# ClusterControl Reference Manual

***Release 2.4.0***

**Severalnines**

**Oct 07, 2026**

**Abstract**

This is the ClusterControl Reference Manual. It documents ClusterControl through version 2.4.0, including installation, configuration, cluster deployment, monitoring, and management features.

For notes detailing the changes in each release, see the ClusterControl Release Notes.

For help with using ClusterControl, please visit the Severalnines support channels, where you can discuss your issues with other users.

Document generated on: October 7, 2026

# TABLE OF CONTENTS

**1 Getting Started** **23**
1.1 Getting Started with ClusterControl . . . 23
1.1.1 What is ClusterControl? . . . 23
1.1.2 Editions and releases . . . 24
1.1.3 Target users . . . 24
1.1.4 Next steps . . . 25
1.2 How it Works? . . . 25
1.2.1 Components . . . 25
1.2.2 Architecture . . . 27
1.2.3 Management and automation operations . . . 29
1.2.4 Monitoring operation . . . 29
1.2.5 Multiple controllers management . . . 31
1.3 Supported databases . . . 32
1.3.1 Database servers . . . 32
1.3.2 Auxiliary tools . . . 34
1.4 Key Features . . . 34
1.4.1 All cluster types . . . 34
1.4.2 Cluster-specific features . . . 36
1.5 Requirements . . . 44
1.5.1 Hardware . . . 44
1.5.2 Operating system . . . 46
1.5.3 Software dependencies . . . 47
1.5.4 Firewall and security groups . . . 47
1.5.5 Operating system user . . . 48
1.5.6 SSH key-based authentication . . . 51
1.5.7 Timezone . . . 54
1.6 General . . . 54
1.6.1 Is ClusterControl free? . . . 54
1.6.2 Is there a live demo site where I can try out ClusterControl? . . . 55
1.6.3 Can I purchase the license for the first year only? . . . 55
1.6.4 My 30-day demo license has expired but I haven't had enough time to test ClusterControl. Can I request for an extension? . . . 55
1.6.5 I would like to subscribe as a paid user but have a limited budget. What should I do? 55
1.6.6 Can I resell your product? . . . 55
1.6.7 Who uses ClusterControl? . . . 55
1.7 Technical . . . 55
1.7.1 Do I need to be a DBA to use ClusterControl? . . . 55
1.7.2 Does ClusterControl run on Windows? . . . 55

1.7.3 How do I upgrade ClusterControl? . . . 55
1.7.4 I'm using ClusterControl community version. Which support channel should I use? 56
1.7.5 Can I install ClusterControl on RHEL-based system but my database cluster is running on Debian? . . . 56
1.7.6 Can I use ClusterControl in an offline environment (without Internet)? . . . 56
1.7.7 Can I have a mix of different SSH users per cluster? . . . 56
1.7.8 Can I use a non-privileged SSH user for ClusterControl? . . . 56
1.7.9 What is difference of MySQL Galera Cluster, Percona XtraDB Cluster and MariaDB Galera Cluster? . . . 56
1.7.10 I found a bug. Where should I report it? . . . 56
1.7.11 My cluster is down. What should I do? . . . 56
1.7.12 How can I submit a feature request? . . . 56
1.8 Quickstart . . . 57
1.8.1 Step 1: Install ClusterControl . . . 57
1.8.2 Step 2: Create the first admin user . . . 58
1.8.3 Step 3: Set up SSH key-based authentication . . . 59
1.8.4 Step 4: Deploy or import database cluster/server . . . 60
1.8.5 Step 5: Start managing . . . 64
1.8.6 Step 6: Additional resources . . . 66
1.9 Online Installation . . . 66
1.9.1 Installer script (install-cc) . . . 66
1.9.2 Helm Chart . . . 68
1.9.3 Ansible Role . . . 71
1.9.4 Puppet module . . . 73
1.10 Offline Setup . . . 77
1.10.1 Step 1: Build and export the repository . . . 77
1.10.2 Step 2: Remove default repositories . . . 85
1.10.3 Step 3: Import the repository to the air-gapped host . . . 86
1.10.4 Step 4: Install ClusterControl in offline mode . . . 88
1.10.5 Step 5: Post-installation . . . 90
1.11 Installer Setup . . . 91
1.12 Troubleshooting Issues . . . 96
1.12.1 Error page "Not Found" after installation . . . 96
1.13 Manual Installation . . . 98
1.13.1 Requirements . . . 98
1.13.2 Installation Steps . . . 98
1.13.3 Troubleshooting Issues . . . 106
1.14 ClusterControl repository . . . 109
1.15 ClusterControl CLI repository . . . 110
1.16 Legacy GUI Installation . . . 111
1.16.1 Caveats . . . 111
1.16.2 Installation steps . . . 111
1.17 ClusterControl on Marketplaces . . . 119
1.17.1 Current Marketplaces . . . 120
1.17.2 Passwordless SSH connections . . . 120
1.17.3 Launching the ClusterControl Application . . . 120
1.17.4 Getting the Public SSH Key . . . 120
1.17.5 Launching the Hosts for the Database Nodes . . . 121
1.17.6 Deploying a Database Cluster . . . 121
1.18 Tutorials . . . 121
1.18.1 Day-1 Operations . . . 121

1.18.2 Day-2 Operations . . . 122
1.19 Standalone MySQL (MariaDB, Percona, Oracle) . . . 122
1.19.1 Prerequisites . . . 123
1.19.2 Architecture . . . 123
1.19.3 Step 1: Set up SSH key-based authentication . . . 124
1.19.4 Step 2: Deploy a new cluster (single node) . . . 125
1.19.5 Step 3: Monitor your cluster . . . 126
1.19.6 Step 4: Import data . . . 126
1.19.7 Step 5: (Optional) Deploy a new load balancer . . . 126
1.19.8 Step 6: Connect to the database server . . . 127
1.19.9 Step 7: Enable automatic backups . . . 127
1.19.10 Step 8: Configure alerts . . . 128
1.19.11 Step 9: Manage your node . . . 128
1.19.12 Step 10: Scale your cluster . . . 129
1.19.13 Conclusion . . . 130
1.20 MySQL Replication (MariaDB, Percona, Oracle) . . . 130
1.20.1 Prerequisites . . . 131
1.20.2 Architecture . . . 131
1.20.3 Step 1: Set up SSH key-based authentication . . . 132
1.20.4 Step 2: Deploy a new cluster . . . 133
1.20.5 Step 3: Monitor your cluster . . . 134
1.20.6 Step 4: Import data . . . 135
1.20.7 Step 5: Deploy load balancers . . . 135
1.20.8 Step 6: Deploy virtual IP address . . . 137
1.20.9 Step 7: Connect to the database cluster . . . 137
1.20.10 Step 8: Enable automatic backups . . . 138
1.20.11 Step 9: Configure alerts . . . 139
1.20.12 Step 10: Manage your cluster and load balancers . . . 140
1.20.13 Step 11: Scale your cluster . . . 140
1.20.14 Conclusion . . . 141
1.21 MySQL Group Replication . . . 142
1.21.1 Prerequisites . . . 142
1.21.2 Architecture . . . 142
1.21.3 Step 1: Set up SSH key-based authentication . . . 143
1.21.4 Step 2: Deploy a new cluster . . . 144
1.21.5 Step 3: Monitor your cluster . . . 145
1.21.6 Step 4: Import data . . . 146
1.21.7 Step 5: Deploy a new load balancer . . . 147
1.21.8 Step 6: Connecting to the database cluster . . . 148
1.21.9 Step 7: Enable automatic backups . . . 149
1.21.10 Step 8: Configure alerts . . . 150
1.21.11 Step 9: Manage your cluster and load balancer . . . 150
1.21.12 Step 10: Scale your cluster . . . 151
1.21.13 Conclusion . . . 152
1.22 Percona XtraDB Cluster . . . 152
1.22.1 Prerequisites . . . 153
1.22.2 Architecture . . . 153
1.22.3 Step 1: Set up SSH key-based authentication . . . 154
1.22.4 Step 2: Deploy a new cluster . . . 155
1.22.5 Step 3: Monitor your cluster . . . 156
1.22.6 Step 4: Import data . . . 157

1.22.7 Step 5: Deploy a new load balancer . . . 158
1.22.8 Step 6: Connect to the database cluster . . . 159
1.22.9 Step 7: Enable automatic backups . . . 159
1.22.10 Step 8: Configure alerts . . . 161
1.22.11 Step 9: Manage your cluster and load balancer . . . 161
1.22.12 Step 10: Scale your cluster . . . 162
1.22.13 Conclusion . . . 163
1.23 MariaDB Galera Cluster . . . 163
1.23.1 Prerequisites . . . 164
1.23.2 Architecture . . . 164
1.23.3 Step 1: Set up SSH key-based authentication . . . 165
1.23.4 Step 2: Deploy a new cluster . . . 165
1.23.5 Step 3: Monitor your cluster . . . 167
1.23.6 Step 4: Import data . . . 167
1.23.7 Step 5: Deploy load balancers and virtual IP . . . 168
1.23.8 Step 6: Connect to the database cluster . . . 170
1.23.9 Step 7: Enable automatic backups . . . 171
1.23.10 Step 8: Configure alerts . . . 172
1.23.11 Step 9: Manage your cluster and load balancer . . . 172
1.23.12 Step 10: Scale your cluster . . . 173
1.23.13 Conclusion . . . 174
1.24 PostgreSQL Streaming Replication . . . 174
1.24.1 Prerequisites . . . 175
1.24.2 Architecture . . . 175
1.24.3 Step 1: Set up SSH key-based authentication . . . 176
1.24.4 Step 2: Deploy a new cluster . . . 177
1.24.5 Step 3: Monitor your cluster . . . 178
1.24.6 Step 4: Import data . . . 179
1.24.7 Step 5: Deploy a new load balancer . . . 179
1.24.8 Step 6: Connecting to the database cluster . . . 181
1.24.9 Step 7: Enable automatic backups . . . 181
1.24.10 Step 8: Configure alerts . . . 182
1.24.11 Step 9: Manage your cluster and load balancer . . . 183
1.24.12 Step 10: Scale your cluster . . . 184
1.24.13 Conclusion . . . 184
1.25 Prerequisites . . . 185
1.26 Architecture . . . 185
1.27 Step 1: Set up SSH key-based authentication . . . 186
1.28 Step 2: Deploy a new cluster . . . 187
1.29 Step 3: Monitor your cluster . . . 188
1.30 Step 4: Deploy PgBouncer . . . 188
1.31 Step 5: Creating connection pool for PgBouncer . . . 189
1.32 Step 6: Connecting to the database cluster . . . 189
1.33 Step 7: Connecting to the database cluster via PgBouncer . . . 190
1.34 Step 8: Enable automatic backups . . . 190
1.35 Step 9: Configure alerts . . . 191
1.36 Step 10: Manage your cluster . . . 192
1.37 Conclusion . . . 192
1.38 PostgreSQL Logical Replication . . . 192
1.38.1 Introduction . . . 192
1.38.2 Architecture . . . 193

1.38.3 Prerequisites . . . 194
1.38.4 Step 1: Set up SSH key-based authentication . . . 194
1.38.5 Step 2: Deploy a new cluster . . . 195
1.38.6 Step 3: Publication and Subscription . . . 197
1.39 Prerequisites . . . 198
1.40 Architecture . . . 198
1.41 Step 1: Set up SSH key-based authentication . . . 199
1.42 Step 2: Deploy MongoDB replica set . . . 199
1.43 Step 3: Monitor your cluster . . . 201
1.44 Step 4: Schedule automatic backups . . . 201
1.45 Step 5: Configure alerts and notifications . . . 202
1.46 Step 6 Connect to the cluster . . . 202
1.47 Step 7: Cluster management operations . . . 203
1.48 Conclusion . . . 203
1.49 Prerequisites . . . 203
1.50 Architecture . . . 204
1.51 Step 1: Set up SSH key-based authentication . . . 204
1.52 Step 2: Deploy MongoDB Enterprise Replica Set . . . 205
1.53 Step 3: Monitor your cluster . . . 206
1.54 Step 4: Schedule automatic backups . . . 206
1.55 Step 5: Configure alerts and notifications . . . 207
1.56 Step 6: Connect to the cluster . . . 208
1.57 Step 7: Verify Enterprise installation . . . 208
1.58 Step 8: Cluster management operations . . . 208
1.59 Conclusion . . . 208
1.60 Prerequisites . . . 209
1.61 Architecture . . . 209
1.62 Step 1: Set up SSH key-based authentication . . . 210
1.63 Step 2: Deploy MongoDB Sharded Cluster . . . 210
1.64 Step 3: Monitor your cluster . . . 212
1.65 Step 4: Schedule automatic backups . . . 212
1.66 Step 5: Configure alerts and notifications . . . 213
1.67 Step 6 Connect to the cluster . . . 213
1.68 Step 7: Cluster management operations . . . 214
1.69 Conclusion . . . 214
1.70 Redis Replication with Sentinel . . . 214
1.70.1 Prerequisites . . . 215
1.70.2 Architecture . . . 215
1.70.3 Step 1: Set up SSH key-based authentication . . . 215
1.70.4 Step 2: Deploy a Redis Sentinel Cluster . . . 216
1.70.5 Step 3: Monitor your cluster . . . 217
1.70.6 Step 4: Connecting to the database cluster . . . 217
1.70.7 Step 5: Enable automatic backups . . . 218
1.70.8 Step 6: Configure alerts . . . 219
1.70.9 Step 7: Manage your cluster . . . 219
1.70.10 Step 8: Scale your cluster . . . 220
1.70.11 Conclusion . . . 221
1.71 Valkey Replication with Sentinel . . . 221
1.71.1 Prerequisites . . . 221
1.71.2 Architecture . . . 222
1.71.3 Step 1: Set up SSH key-based authentication . . . 222

1.71.4 Step 2: Deploy a Valkey Sentinel Cluster . . . 222
1.71.5 Step 3: Monitor your cluster . . . 224
1.71.6 Step 4: Connecting to the database cluster . . . 224
1.71.7 Step 5: Enable automatic backups . . . 224
1.71.8 Step 6: Configure alerts . . . 226
1.71.9 Step 7: Manage your cluster . . . 226
1.71.10 Step 8: Scale your cluster . . . 227
1.71.11 Conclusion . . . 228
1.72 Redis Cluster . . . 228
1.72.1 Prerequisites . . . 228
1.72.2 Architecture . . . 229
1.72.3 Step 1: Set up SSH key-based authentication . . . 229
1.72.4 Step 2: Deploy a Redis Cluster . . . 229
1.72.5 Step 3: Monitor your cluster . . . 231
1.72.6 Step 4: Connecting to the database cluster . . . 231
1.72.7 Step 5: Enable automatic backups . . . 232
1.72.8 Step 6: Configure alerts . . . 233
1.72.9 Step 7: Manage your cluster . . . 234
1.72.10 Step 8: Scale your cluster . . . 234
1.72.11 Conclusion . . . 235
1.73 Valkey Cluster . . . 235
1.73.1 Prerequisites . . . 236
1.73.2 Architecture . . . 236
1.73.3 Step 1: Set up SSH key-based authentication . . . 236
1.73.4 Step 2: Deploy a Valkey Cluster . . . 237
1.73.5 Step 3: Monitor your cluster . . . 239
1.73.6 Step 4: Connecting to the database cluster . . . 239
1.73.7 Step 5: Enable automatic backups . . . 239
1.73.8 Step 6: Configure alerts . . . 240
1.73.9 Step 7: Manage your cluster . . . 241
1.73.10 Step 8: Scale your cluster . . . 242
1.73.11 Conclusion . . . 242
1.74 SQL Server standalone . . . 243
1.74.1 Prerequisites . . . 243
1.74.2 Architecture . . . 243
1.74.3 Step 1: Set up SSH key-based authentication . . . 244
1.74.4 Step 2: Deploy SQL Server standalone . . . 245
1.74.5 Step 3: Monitor your cluster . . . 246
1.74.6 Step 4: Connecting to the database cluster . . . 246
1.74.7 Step 5: Enable automatic backups . . . 246
1.74.8 Step 6: Configure alerts . . . 247
1.74.9 Step 7: Manage your cluster . . . 248
1.74.10 Conclusion . . . 248
1.75 SQL Server Always On availability group . . . 249
1.75.1 Prerequisites . . . 249
1.75.2 Architecture . . . 250
1.75.3 Step 1: Set up SSH key-based authentication . . . 250
1.75.4 Step 2: Deploy a SQL Server Always On availability group . . . 251
1.75.5 Step 3: Connecting to the database cluster . . . 252
1.75.6 Step 4: Create a database and add into availability group . . . 252
1.75.7 Step 5: Monitor your cluster . . . 253

1.75.8 Step 6: Enable automatic backups . . . 254
1.75.9 Step 7: Configure alerts . . . 255
1.75.10 Step 8: Manage your cluster . . . 255
1.75.11 Step 9: Scale your cluster . . . 256
1.75.12 Conclusion . . . 256
1.76 Elasticsearch Standalone Cluster . . . 257
1.76.1 Prerequisites . . . 257
1.76.2 Architecture . . . 257
1.76.3 Step 1: Set up SSH key-based authentication . . . 258
1.76.4 Step 2: Deploy a new cluster (single node) . . . 259
1.76.5 Step 3: Monitor your cluster . . . 260
1.76.6 Step 4: Import data . . . 260
1.76.7 Step 5: Connect to the database . . . 262
1.76.8 Step 6: Enable automatic backups . . . 262
1.76.9 Step 7: Configure alerts . . . 263
1.76.10 Step 8: Manage your node . . . 263
1.76.11 Conclusion . . . 264
1.77 Elasticsearch High-Availability Cluster . . . 264
1.77.1 Prerequisites . . . 265
1.77.2 Architecture . . . 265
1.77.3 Step 1: Set up SSH key-based authentication . . . 266
1.77.4 Step 2: Deploy a new cluster (high-availability) . . . 266
1.77.5 Step 3: Monitor your cluster . . . 268
1.77.6 Step 4: Import data . . . 268
1.77.7 Step 5: Connect to the database . . . 269
1.77.8 Step 6: Enable automatic backups . . . 269
1.77.9 Step 7: Configure alerts . . . 270
1.77.10 Step 8: Manage your node . . . 271
1.77.11 Step 9: Scale your cluster . . . 271
1.77.12 Conclusion . . . 272
1.78 ClickHouse Standalone . . . 273
1.78.1 Prerequisites . . . 273
1.78.2 Architecture . . . 274
1.78.3 Step 1: Set up SSH key-based authentication . . . 274
1.78.4 Step 2: Deploy a new cluster (single node) . . . 275
1.78.5 Step 3: Monitor your cluster . . . 276
1.78.6 Step 4: Import data . . . 276
1.78.7 Step 5: Connect to the database . . . 277
1.78.8 Step 6: Enable automatic backups . . . 277
1.78.9 Step 7: Configure alerts . . . 278
1.78.10 Step 8: Manage your node . . . 279
1.78.11 Conclusion . . . 279
1.79 ClickHouse Replicated Cluster . . . 280
1.79.1 Prerequisites . . . 280
1.79.2 Architecture . . . 281
1.79.3 Step 1: Set up SSH key-based authentication . . . 283
1.79.4 Step 2: Deploy a new cluster (replicated) . . . 283
1.79.5 Step 3: Monitor your cluster . . . 285
1.79.6 Step 4: Import data . . . 285
1.79.7 Step 5: Connect to the database . . . 286
1.79.8 Step 6: Enable automatic backups . . . 287

1.79.9 Step 7: Configure alerts . . . 288
1.79.10 Step 8: Manage your nodes . . . 288
1.79.11 Step 9: Scale your cluster . . . 289
1.79.12 Conclusion . . . 290
1.80 ClusterControl and Database Clusters Deployment in an Offline Environment . . . 290
1.80.1 Setting up Satellite Repository Server . . . 290
1.80.2 Setting Up Database Nodes . . . 298
1.80.3 ClusterControl Offline Installation . . . 302
1.80.4 Offline Database Deployment using ClusterControl . . . 303
1.81 Use cases . . . 306
1.82 Deploying a Secondary Cluster . . . 307
1.82.1 Replicate from Primary . . . 307
1.82.2 Replicate from Backups . . . 308
1.82.3 Clone cluster . . . 308
1.83 Failover . . . 309
1.84 Cluster Replication Status using CLI . . . 309
1.85 Use cases . . . 310
1.86 Important considerations . . . 310
1.87 Deploying a PostgreSQL cluster . . . 310
1.88 Major upgrades for PostgreSQL . . . 311
1.88.1 Upgrade methods . . . 311
1.88.2 Monitoring upgrade process . . . 312
1.89 Using the ClusterControl CLI . . . 312
1.89.1 Monitoring the upgrade . . . 312
1.90 Use cases . . . 313
1.91 Configuring Cloud Storage . . . 313
1.92 Creating a Backup in the Cloud . . . 314
1.93 Restoring from the Cloud . . . 314
1.94 Automating Backups . . . 314
1.95 Verifying Backups . . . 315
1.96 Accessing Cloud Backups via CLI . . . 315
1.97 Use cases . . . 316
1.98 Configuring point-in-time recovery . . . 316
1.98.1 Enable transaction log archiving . . . 316
1.98.2 Enable full backup . . . 317
1.99 Performing point-in-time recovery . . . 317
1.99.1 Using the ClusterControl GUI . . . 317
1.99.2 Using the ClusterControl CLI . . . 318
1.99.3 Verifying the recovery . . . 318
1.100 Use cases . . . 319
1.101 Creating a read-only user . . . 319
1.102 CLI client configuration . . . 321
1.103 Accessing the RPC API . . . 321
1.104 Limitations of This Method . . . 323
1.105 Pre-Migration Checks . . . 323
1.106 Installing ClusterControl v2 . . . 324
1.107 Importing Existing Clusters into ClusterControl v2 . . . 324
1.107.1 Prerequisites . . . 324
1.107.2 Importing a MySQL/MariaDB standalone or replication . . . 325
1.107.3 Importing a PostgreSQL/TimescaleDB cluster . . . 327
1.107.4 Importing a MongoDB replica set . . . 329

1.108 Importing Configurations from ClusterControl v1 . . . 331
1.108.1 Users and Access . . . 332
1.108.2 Notification Settings . . . 333
1.108.3 SSL Certificates . . . 334
1.108.4 Backup Schedules . . . 335
1.108.5 Scheduled Operational Reports . . . 336
1.109 Post-Migration Checks . . . 337
1.110 User and Access Management . . . 339
1.110.1 List Functions . . . 339
1.110.2 Example Prompt . . . 339
1.111 Cluster Inventory and Health . . . 340
1.111.1 List Functions . . . 340
1.111.2 Example Prompt . . . 340
1.112 Performance Analysis . . . 341
1.112.1 List Functions . . . 341
1.112.2 Example Prompt . . . 341
1.113 Jobs and Operations . . . 342
1.113.1 List Functions . . . 342
1.113.2 Example Prompt . . . 342
1.114 Backups and Scheduling . . . 342
1.114.1 List Functions . . . 342
1.114.2 Example Prompt . . . 343
1.115 ProxySQL . . . 343
1.115.1 List Functions . . . 343
1.115.2 Example Prompt . . . 343
1.116 Configuration . . . 343
1.116.1 List Functions . . . 344
1.116.2 Example Prompt . . . 344
1.117 Maintenance, Audit, and Other Tools . . . 344
1.117.1 List Functions . . . 345
1.117.2 Example Prompt . . . 345

**2 User Guide** **347**
2.1 ClusterControl User Guide . . . 347
2.1.1 What's in this guide . . . 347
2.1.2 Who should use this guide . . . 347
2.2 Overview . . . 347
2.2.1 ClusterControl GUI . . . 348
2.2.2 ClusterControl CLI . . . 349
2.2.3 ClusterControl RPC API . . . 350
2.2.4 Terraform provider for ClusterControl . . . 350
2.3 ClusterControl GUI . . . 351
2.3.1 Accessing the GUI . . . 352
2.3.2 User registration page . . . 352
2.3.3 Login page . . . 352
2.3.4 Home . . . 353
2.3.5 Navigation bar . . . 355
2.3.6 Sidebar . . . 356
2.3.7 Footer . . . 363
2.3.8 Give Us Feedback . . . 363
2.4 ClusterControl CLI . . . 363

2.4.1 Overview . . . 364
2.4.2 Installation . . . 364
2.4.3 Configuring CLI user . . . 365
2.4.4 Using the CLI . . . 368
2.5 Accessing the RPC API . . . 379
2.6 Authenticating with the API . . . 380
2.7 Successful request . . . 381
2.8 Failed request . . . 382
2.9 Addons . . . 382
2.9.1 Access the Addons tab . . . 383
2.9.2 Ops-Center . . . 383
2.9.3 Kubernetes . . . 384
2.10 Deployment . . . 385
2.11 Create Database Cluster . . . 385
2.11.1 Support Matrix . . . 386
2.11.2 Multi-threaded Deployment . . . 386
2.11.3 Prerequisites . . . 387
2.11.4 Common Wizard Steps . . . 387
2.11.5 Database-Specific Settings . . . 389
2.12 Import Database Cluster . . . 434
2.12.1 Support Matrix . . . 434
2.12.2 Prerequisites . . . 434
2.12.3 Common Wizard Steps . . . 435
2.12.4 Database-Specific Settings . . . 437
2.13 Create Load Balancer . . . 464
2.13.1 Support Matrix . . . 465
2.13.2 Prerequisites . . . 465
2.13.3 ProxySQL . . . 465
2.13.4 HAProxy . . . 468
2.13.5 MariaDB MaxScale . . . 472
2.13.6 Garbd . . . 474
2.13.7 PgBouncer . . . 475
2.13.8 Keepalived . . . 477
2.14 Import Load Balancer . . . 478
2.14.1 Prerequisites . . . 479
2.14.2 Support Matrix . . . 479
2.14.3 ProxySQL . . . 479
2.14.4 HAProxy . . . 480
2.14.5 MariaDB MaxScale . . . 482
2.14.6 Garbd . . . 483
2.14.7 PgBouncer . . . 484
2.14.8 Keepalived . . . 485
2.15 Observability . . . 486
2.16 Database Status & Health Checks . . . 487
2.16.1 Cluster Summary . . . 487
2.16.2 Dashboard Metrics . . . 487
2.16.3 Performance Advisors . . . 489
2.17 Cluster Node Summary . . . 490
2.18 Node Health Status . . . 491
2.18.1 CPU Utilization . . . 491
2.18.2 Memory Utilization . . . 491

2.18.3 Disk IO Utilization . . . 491
2.19 Node Menu . . . 491
2.20 Configure Slow Query Log . . . 492
2.21 Slow Query Monitor . . . 493
2.22 RealTime Query Monitor . . . 493
2.23 ProxySQL for Query Monitor . . . 493
2.24 Reset Query Monitoring . . . 494
2.25 DB Growth . . . 494
2.26 DB Status . . . 494
2.27 DB Variables . . . 495
2.28 Advisors . . . 495
2.29 Schema Analyzer . . . 495
2.30 Transaction Deadlocks . . . 496
2.31 Running Operations . . . 496
2.32 System Logs . . . 496
2.33 Audit Logs . . . 497
2.34 Audit Logging . . . 497
2.34.1 MariaDB . . . 497
2.34.2 PostgreSQL . . . 498
2.35 Watchlist - Monitoring dashboards in kiosk-mode . . . 499
2.35.1 Access Watchlist management section . . . 500
2.35.2 Add clusters directly into watchlist . . . 500
2.35.3 Create watchlist . . . 501
2.35.4 Monitor in kiosk-mode . . . 501
2.35.5 Share watchlist with others . . . 501
2.35.6 Delete watchlist . . . 502
2.36 Alarms . . . 502
2.37 Jobs . . . 503
2.38 Audit Log . . . 504
2.39 Report templates . . . 504
2.40 Generating a report . . . 505
2.41 Listing and viewing reports . . . 507
2.42 Scheduling a report . . . 507
2.43 Management . . . 508
2.44 Failover and Recovery . . . 509
2.44.1 Node Recovery . . . 509
2.44.2 Cluster Recovery . . . 510
2.45 Cluster Management . . . 513
2.45.1 Cluster Actions . . . 514
2.45.2 Cluster-specific actions . . . 528
2.46 Nodes Management . . . 529
2.46.1 Access the Nodes section . . . 530
2.46.2 Node Actions . . . 530
2.46.3 Replication actions . . . 534
2.46.4 Read-only and binary logging . . . 537
2.46.5 Galera-specific actions . . . 538
2.46.6 MongoDB-specific actions . . . 540
2.46.7 PostgreSQL-specific actions . . . 540
2.46.8 ProxySQL-specific actions . . . 542
2.46.9 ClickHouse-specific actions . . . 543
2.46.10 Schedule maintenance for a node . . . 544

2.47 Configuration Management . . . 545
2.47.1 Access the configuration management section . . . 545
2.47.2 Refresh configuration files . . . 547
2.47.3 View and edit configuration files . . . 547
2.47.4 Compare new configuration with existing configuration . . . 549
2.47.5 Apply configuration changes across nodes . . . 549
2.47.6 Monitor configuration consistency . . . 552
2.47.7 Configuration template . . . 552
2.48 Upgrades and Patches . . . 556
2.48.1 Upgrade procedures . . . 556
2.48.2 Minor upgrade . . . 557
2.48.3 Major upgrade . . . 559
2.49 Alerts and Notifications . . . 561
2.49.1 Alarms . . . 561
2.49.2 Email notifications . . . 562
2.49.3 Forward alarms to third-party services . . . 565
2.50 Database Users and Privileges . . . 565
2.50.1 Best practices . . . 565
2.50.2 Access the database users management . . . 566
2.50.3 MySQL/MariaDB-based clusters . . . 566
2.50.4 PostgreSQL-based clusters . . . 575
2.50.5 Redis-based clusters . . . 586
2.51 MySQL/MariaDB Replication . . . 592
2.52 MySQL Galera . . . 592
2.53 MySQL Cluster (NDB) . . . 593
2.54 MongoDB ReplicaSet . . . 593
2.55 PostgreSQL/TimeScaleDB Replication . . . 594
2.56 Maintenance . . . 594
2.56.1 Type of maintenance mode . . . 595
2.56.2 Configuring maintenance mode . . . 595
2.56.3 Checking maintenance status . . . 596
2.56.4 Deactivating maintenance mode . . . 597
2.57 Certificate Management . . . 597
2.57.1 Best practices and management notes . . . 598
2.57.2 Access Certificate Management . . . 598
2.57.3 View the certificate list . . . 600
2.57.4 Generate a self-signed Certificate Authority (CA) . . . 601
2.57.5 Generate a client or server certificate . . . 602
2.57.6 Import an existing certificate . . . 603
2.57.7 Move or rename a certificate . . . 605
2.57.8 Revoke a certificate . . . 606
2.57.9 Generate a Certificate Revocation List (CRL) . . . 606
2.57.10 Delete a certificate . . . 607
2.58 Backup and Restore . . . 608
2.59 Overview . . . 608
2.59.1 What you can do . . . 608
2.59.2 Backup jobs . . . 609
2.59.3 Backup encryption and decryption . . . 610
2.59.4 Backup retention . . . 611
2.59.5 Backup storage . . . 611
2.59.6 Backup settings . . . 612

2.59.7 Supported backup methods . . . 613
2.60 Create Backup . . . 625
2.60.1 Supported backup methods . . . 626
2.60.2 Prerequisites . . . 626
2.60.3 Backup jobs . . . 627
2.60.4 Common Wizard Steps . . . 627
2.60.5 Database-Specific Settings . . . 631
2.60.6 RPC API job_data Reference . . . 653
2.60.7 Advanced Example . . . 664
2.61 Schedule Backup . . . 666
2.61.1 Supported backup methods . . . 666
2.61.2 Prerequisites . . . 667
2.61.3 Backup jobs & scheduling . . . 668
2.61.4 Common Wizard Steps . . . 668
2.61.5 Database-Specific Settings . . . 674
2.61.6 Manage Existing Schedules . . . 700
2.61.7 RPC API job_data Reference . . . 705
2.61.8 Advanced Example . . . 718
2.62 Restore Backup . . . 720
2.62.1 Supported backup methods . . . 720
2.62.2 Prerequisites . . . 720
2.62.3 Restoring vs. Verifying . . . 721
2.62.4 Point-in-Time Recovery (PITR) . . . 721
2.62.5 Common Wizard Steps . . . 723
2.62.6 Database-Specific Settings . . . 725
2.63 Restore External Backup . . . 735
2.63.1 Restoring internal vs. external backups . . . 735
2.63.2 Supported backup methods . . . 736
2.63.3 Prerequisites . . . 736
2.63.4 Common Wizard Steps . . . 737
2.64 Backup Verification . . . 739
2.64.1 Supported backup methods . . . 739
2.64.2 Prerequisites . . . 740
2.64.3 Common Wizard Steps . . . 740
2.64.4 Database-Specific Settings . . . 742
2.65 Create Cluster from Backup . . . 748
2.65.1 Support Matrix . . . 748
2.65.2 Prerequisites . . . 749
2.65.3 Creating the Cluster from Backup . . . 749
2.66 Upload Backup to Cloud . . . 752
2.66.1 Uploading backups to the cloud . . . 752
2.66.2 Support Matrix . . . 753
2.66.3 Prerequisites . . . 753
2.66.4 Default Configuration . . . 753
2.66.5 Steps to Upload a Backup . . . 753
2.66.6 Steps to Re-upload a Backup (for backups already in the cloud) . . . 755
2.67 Download Backup from Cloud . . . 756
2.67.1 Support Matrix . . . 756
2.67.2 Prerequisites . . . 756
2.67.3 Default Configuration . . . 757
2.67.4 Steps to Download a Backup . . . 757

2.68 Create Replica Cluster . . . 757
2.68.1 Support Matrix . . . 758
2.68.2 Prerequisites . . . 758
2.68.3 Create Replica Cluster Options . . . 759
2.68.4 Galera Cluster . . . 759
2.68.5 PostgreSQL . . . 762
2.68.6 TimescaleDB . . . 765
2.69 Add Database Node . . . 767
2.69.1 Support Matrix . . . 768
2.69.2 Prerequisites . . . 768
2.69.3 MySQL Galera . . . 768
2.69.4 MongoDB Replica Set . . . 771
2.69.5 MongoDB Sharded Cluster . . . 773
2.69.6 Redis Sentinel . . . 776
2.69.7 Redis/Valkey Cluster . . . 778
2.69.8 SQL Server . . . 780
2.69.9 ClickHouse . . . 782
2.70 Add Replica Node . . . 786
2.70.1 Support Matrix . . . 786
2.70.2 Prerequisites . . . 786
2.70.3 MySQL Replication . . . 787
2.70.4 Galera Cluster . . . 790
2.70.5 PostgreSQL . . . 793
2.70.6 TimescaleDB . . . 796
2.70.7 Redis/Valkey Cluster . . . 798
2.70.8 Post Deployment . . . 800
2.71 Remove Database Node (or Remove Node) . . . 800
2.71.1 Support Matrix . . . 801
2.71.2 Prerequisites . . . 801
2.71.3 Deployment steps . . . 801
2.72 Single-Controller vs Multi-Controller . . . 803
2.73 Activating ClusterControl Ops-C . . . 803
2.74 Deactivating ClusterControl Ops-C . . . 805
2.75 Controller Management . . . 805
2.75.1 Add Controller . . . 806
2.75.2 Edit Controller . . . 807
2.75.3 Remove Controller . . . 808
2.76 Monitoring Dashboard . . . 809
2.77 ClusterControl Manager . . . 810
2.78 ClusterControl Kubernetes proxy . . . 811
2.79 GitOps . . . 813
2.79.1 Argo CD integration . . . 815
2.79.2 GitHub integration . . . 815
2.80 Deploy your first Kubernetes cluster . . . 816
2.80.1 1. Install ClusterControl Ops-C . . . 816
2.80.2 2. Install the required tools . . . 816
2.80.3 3. Create a Kubernetes cluster . . . 816
2.80.4 4. Connect to a Kubernetes cluster . . . 816
2.80.5 5. Configure the agent . . . 817
2.80.6 6. Set up the repository . . . 818
2.80.7 7. Addons . . . 818

2.80.8 8. Monitoring . . . 819
2.80.9 9. Prometheus Agent . . . 819
2.80.10 10. Install Argo CD . . . 820
2.80.11 11. Install the agent . . . 820
2.80.12 12. Wait for environment detection . . . 821
2.80.13 10. Pull requests . . . 821
2.80.14 11. GitOps synchronization . . . 822
2.80.15 12. Confirm activation . . . 822
2.81 Create a Kubernetes namespace . . . 822
2.82 Deploy a database operator . . . 823
2.82.1 Supported operators . . . 823
2.82.2 Kubernetes and operator version compatibility . . . 824
2.83 Create a new database cluster . . . 824
2.83.1 Database connection details . . . 826
2.84 Resource and database configuration templates . . . 827
2.85 Retrieve and view logs . . . 827
2.86 Purge resources . . . 827
2.87 Observability . . . 828
2.88 Troubleshooting issues . . . 828
2.88.1 Restart agent . . . 828
2.88.2 Agent with debug log . . . 828
2.88.3 Basic troubleshooting commands . . . 829
2.89 Example issues and solutions . . . 829
2.89.1 Failed database operators . . . 829
2.89.2 Failed backups . . . 833
2.89.3 Node maintenance and troubleshooting . . . 834
2.90 Cloud Providers . . . 835
2.90.1 Components . . . 835
2.90.2 Prerequisites . . . 835
2.90.3 Accessing the Cloud Storage Credentials section . . . 836
2.90.4 Creating a New S3 Integration . . . 837
2.91 Requirements . . . 843
2.92 How it works . . . 843
2.93 Configuration guide . . . 843
2.93.1 Part 1 - Install and initialize Vault . . . 845
2.93.2 Part 2 - Configure Vault for ClusterControl . . . 846
2.93.3 Part 3 - Configure ClusterControl . . . 847
2.94 Path to secrets in Vault . . . 848
2.94.1 Secret paths . . . 848
2.94.2 Vault GUI . . . 850
2.95 Reverting Vault integration . . . 856
2.96 Legacy Vault integration . . . 856
2.97 Limitations . . . 856
2.98 Requirements . . . 857
2.99 Providers . . . 857
2.100 Resources . . . 857
2.101 Quick start . . . 857
2.101.1 Installing and configuring ClusterControl for API access . . . 857
2.101.2 Setting up the Terraform provider for ClusterControl . . . 858
2.101.3 Deploying database clusters using Terraform . . . 859
2.101.4 Destroying a deployed database cluster . . . 860

2.102 LDAP settings . . . 863
2.103 Map LDAP group . . . 864
2.104 Delete mapping group . . . 864
2.105 Supported integrations . . . 865
2.106 Notification behavior . . . 865
2.107 Set up notification . . . 866
2.107.1 Slack . . . 866
2.107.2 Telegram . . . 867
2.107.3 ServiceNow . . . 868
2.107.4 PagerDuty . . . 868
2.107.5 Splunk On-Call (formerly known as VictorOps) . . . 869
2.107.6 OpsGenie . . . 870
2.107.7 ilert . . . 871
2.108 Webhook integration . . . 872
2.109 Events . . . 874
2.109.1 Warning Events . . . 874
2.109.2 Critical Events . . . 877
2.110 Setting up dedicated Prometheus node . . . 879
2.110.1 Host Selection . . . 882
2.110.2 Scrape Interval . . . 882
2.110.3 Data Retention Size . . . 882
2.110.4 Data Directory . . . 882
2.110.5 Scrape Interval . . . 882
2.110.6 Arguments . . . 882
2.110.7 Scrape Interval . . . 883
2.110.8 Arguments . . . 883
2.111 Logging . . . 886
2.112 Debugging . . . 886
2.112.1 Debugging ClusterControl Controller (CMON) . . . 886
2.112.2 Install debugging components . . . 886
2.112.3 Debugging Steps . . . 887
2.113 CMON on Foreground . . . 888
2.114 ClusterControl GUI diagnostic . . . 889
2.115 New deployment abd import cluster . . . 889
2.116 Existing database cluster . . . 889
2.117 Overview . . . 890
2.118 Generating an Error Report . . . 890
2.119 ClusterControl GUI diagnostic . . . 891
2.120 Common Issues . . . 892
2.120.1 ClusterControl Installation . . . 892
2.121 Known Limitations . . . 897
2.121.1 General . . . 897
2.121.2 SSH . . . 897
2.121.3 MySQL/MariaDB . . . 898
2.121.4 Containerization . . . 898
2.121.5 Monitoring . . . 898

**3 Administration Guide** **899**
3.1 Upgrading ClusterControl . . . 899
3.1.1 Online upgrade . . . 899
3.1.2 Offline upgrade . . . 902

3.1.3 Upgrading from ClusterControl v1 to v2 . . . 906
3.1.4 Upgrading to v2.1.0 for Legacy Operating Systems . . . 912
3.1.5 Activating web-based SSH for ClusterControl GUI v2 . . . 914
3.2 Users and Teams . . . 917
3.2.1 Users . . . 917
3.2.2 Teams . . . 920
3.2.3 LDAP . . . 922
3.3 Backup and Restore . . . 925
3.3.1 Prerequisites . . . 926
3.3.2 Back up all clusters . . . 926
3.3.3 Back up an individual cluster . . . 927
3.3.4 Restore an individual cluster to another server . . . 927
3.3.5 Restore the entire controller to another server . . . 928
3.3.6 Known issues and limitations . . . 928
3.4 Configuration Files . . . 929
3.5 Runtime configuration . . . 930
3.6 Applying configuration changes . . . 930
3.7 Securing ClusterControl . . . 931
3.7.1 Firewall and security group . . . 931
3.7.2 SSH . . . 931
3.7.3 File permission . . . 931
3.7.4 HTTPS . . . 932
3.7.5 Access control and identity . . . 932
3.7.6 Secrets and encryption . . . 932
3.8 Secondary Standby . . . 932
3.8.1 Installing standby server . . . 933
3.8.2 Activating the secondary standby . . . 934
3.9 Scalable Controllers Pool . . . 936
3.9.1 How it works . . . 936
3.9.2 Key runtime concepts . . . 937
3.9.3 Shared configuration requirement . . . 937
3.9.4 Deployment and configuration model . . . 937
3.9.5 Example startup pattern for non-Kubernetes environment shared config . . . 938
3.9.6 Example startup pattern for Kubernetes-backed shared config . . . 938
3.9.7 Non-Kubernetes deployment model . . . 938
3.9.8 Capacity and cold-start hardening . . . 938
3.9.9 Managing the Controller Pool . . . 939
3.9.10 Inspecting the controller pool with the CLI . . . 946
3.9.11 Operational guidance . . . 951
3.9.12 Common gotchas . . . 951
3.10 ClusterControl Replication . . . 952
3.10.1 Key Components . . . 952
3.10.2 Architecture . . . 952
3.10.3 Known limitations . . . 953
3.10.4 Requirements . . . 954
3.10.5 Deployment steps . . . 954
3.10.6 Deploy/Import the first cluster . . . 975
3.10.7 Management . . . 975
3.11 Reference Architecture . . . 979
3.11.1 1. Recommended server specs (capacity planning) . . . 979
3.11.2 2. Networking setup . . . 982

3.11.3 3. Storage configuration and partitioning . . . 984
3.11.4 4. SSH key management and rotation . . . 985
3.11.5 5. Using an external CMON database and external Prometheus . . . 986
3.11.6 6. ClusterControl deployment architectures . . . 987
3.11.7 7. Multi-site / multi-datacenter deployment . . . 993
3.11.8 8. Offline or restricted-network deployment . . . 997
3.11.9 9. Cloud deployment considerations . . . 997
3.11.10 10. Example deployments . . . 999
3.11.11 11. Database cluster reference topologies . . . 1002
3.12 Tuning for Large Deployment . . . 1010
3.12.1 Tuning InnoDB buffer pool size . . . 1010
3.12.2 Using MySQL socket file . . . 1011
3.12.3 CMON pool size . . . 1012
3.12.4 File descriptors . . . 1012
3.12.5 OpenSSH with SSSD . . . 1012
3.12.6 File System and Partitioning . . . 1013
3.13 Prometheus storage retention . . . 1013
3.14 CMON data history . . . 1013
3.15 Log rotation . . . 1014
3.16 Architecture at a glance . . . 1015
3.17 Prerequisites . . . 1016
3.18 Installing cmon-telemetry . . . 1016
3.18.1 Generate a signing key and Bearer token . . . 1016
3.18.2 Write the configuration file . . . 1017
3.18.3 Start the service . . . 1018
3.19 Provisioning the cmon-proxy service token . . . 1019
3.19.1 Path A: `ccmgradm serviceauth-bootstrap` (recommended) . . . 1019
3.19.2 Path B: manual via `s9s` . . . 1020
3.19.3 Verify provisioning landed . . . 1021
3.20 Wire the Billing UI passthrough . . . 1021
3.21 Verifying the pipeline end to end . . . 1022
3.21.1 Troubleshooting if data is not flowing . . . 1023
3.22 The Billing page . . . 1024
3.22.1 The Show public key button . . . 1025
3.23 Generating a report . . . 1025
3.24 Viewing a report . . . 1025
3.25 Downloading JSON / CSV . . . 1027
3.26 Deleting a report . . . 1027
3.27 Scoping reports to a customer or tenant . . . 1027
3.28 REST API reference . . . 1028
3.29 Report seals and verification . . . 1030
3.29.1 HMAC-SHA256: internal integrity . . . 1030
3.29.2 Ed25519: asymmetric, third-party verifiable . . . 1030
3.29.3 What the two signatures prove together . . . 1031
3.29.4 Verifying an exported bundle with cmon-report-verify . . . 1031
3.29.5 What verification proves and what it does not . . . 1033
3.30 Retention and data lifecycle . . . 1033
3.31 Frequently asked questions . . . 1033
3.32 Migrating ClusterControl to Another Server . . . 1035
3.32.1 Example use cases . . . 1035
3.32.2 Prerequisites . . . 1035

3.32.3 Migration steps . . . 1035
3.32.4 Post-migration checklist . . . 1040
3.33 Uninstall or Reinstall ClusterControl . . . 1040
3.33.1 Reinstall . . . 1040
3.33.2 Uninstall . . . 1041

**4 Reference Manuals 1043**
4.1 What You'll Find in This Manual . . . 1043
4.1.1 ClusterControl Components . . . 1043
4.1.2 Feature Reference . . . 1043
4.1.3 Command and API Reference . . . 1043
4.1.4 Troubleshooting and Support . . . 1043
4.2 Components . . . 1043
4.3 ClusterControl Controller . . . 1044
4.3.1 Command Line Arguments . . . 1044
4.3.2 Startup File . . . 1045
4.3.3 Configuration Options . . . 1046
4.3.4 LDAP . . . 1069
4.3.5 CMON Database . . . 1073
4.4 ClusterControl GUI . . . 1073
4.4.1 Introduction . . . 1073
4.4.2 Files and location . . . 1074
4.4.3 Managing multiple controllers . . . 1074
4.5 ClusterControl CLI . . . 1076
4.5.1 s9s-account . . . 1076
4.5.2 s9s-alarm . . . 1079
4.5.3 s9s-backup . . . 1080
4.5.4 s9s-cluster . . . 1090
4.5.5 s9s-container . . . 1110
4.5.6 s9s-controller . . . 1115
4.5.7 s9s-job . . . 1117
4.5.8 s9s-log . . . 1122
4.5.9 s9s-maintenance . . . 1124
4.5.10 s9s-metatype . . . 1126
4.5.11 s9s-node . . . 1128
4.5.12 s9s-process . . . 1137
4.5.13 s9s-replication . . . 1139
4.5.14 s9s-report . . . 1141
4.5.15 s9s-script . . . 1143
4.5.16 s9s-server . . . 1145
4.5.17 s9s-tree . . . 1148
4.5.18 s9s-user . . . 1152
4.6 ClusterControl SSH . . . 1159
4.7 ClusterControl Notifications . . . 1160
4.8 ClusterControl Cloud . . . 1160
4.8.1 Commands . . . 1161
4.8.2 Options . . . 1161
4.9 ClusterControl Cloud File Manager . . . 1161
4.9.1 Usage . . . 1162
4.9.2 Commands . . . 1162
4.9.3 Options . . . 1162

4.9.4 Examples . . . 1162
4.10 ClusterControl Proxy . . . 1163
4.10.1 ClusterControl Manager . . . 1163
4.10.2 ClusterControl Manager Admin CLI . . . 1164
4.10.3 Configuration file . . . 1165
4.10.4 Configure custom SSL certificates . . . 1167
4.10.5 Web server: headers and simple examples . . . 1167
4.11 ClusterControl Agent . . . 1169
4.11.1 Query Monitor Agents . . . 1169
4.12 ClusterControl CMON RPC v2 API Reference . . . 1171
4.12.1 Introduction . . . 1171
4.12.2 Available Endpoints . . . 1172
4.12.3 Alarms . . . 1175
4.12.4 Information about Controller's audit log . . . 1181
4.12.5 User Authentication . . . 1183
4.12.6 Information about Backups . . . 1190
4.12.7 Certificate Authority (CA) . . . 1203
4.12.8 Cloud credentials API . . . 1213
4.12.9 Cluster Information and Manipulation . . . 1222
4.12.10 Configuration . . . 1248
4.12.11 Controllers and HA . . . 1276
4.12.12 Discovery . . . 1280
4.12.13 Hosts and Containers . . . 1291
4.12.14 Job Information and Manipulation . . . 1301
4.12.15 Log Related Calls . . . 1315
4.12.16 PostgreSQL Logical Replication . . . 1324
4.12.17 Maintenance . . . 1334
4.12.18 Mysql databaseperformance . . . 1342
4.12.19 Reports . . . 1345
4.12.20 Repositories . . . 1357
4.12.21 Cmon Events . . . 1360
4.12.22 Statistical Data Calls . . . 1362
4.12.23 CMON Directory Tree . . . 1389
4.12.24 User Database manipulation . . . 1402
4.12.25 Controllers manipulation on database . . . 1436
4.12.26 Watchlists manipulation on database . . . 1440
4.13 ClusterControl Domain Specific Language (CCDSL) . . . 1444
4.13.1 1. Language basics . . . 1444
4.13.2 2. Program structure . . . 1447
4.13.3 3. Advisors, programs, and widgets . . . 1447
4.13.4 4. Cluster and host objects . . . 1450
4.13.5 5. Executing SQL commands and queries . . . 1454
4.13.6 6. Obtaining and processing statistical information . . . 1455
4.13.7 7. Advice: `CmonAdvice` . . . 1456
4.13.8 8. Alarms . . . 1457
4.13.9 9. Configuration files: `CmonClusterConfig` . . . 1458
4.13.10 10. Files: `CmonFile` . . . 1459
4.13.11 11. Jobs: `CmonJob` . . . 1459
4.13.12 12. Graphs: `CmonGraph` . . . 1460
4.13.13 13. JSON functions and the Cmon database . . . 1461
4.13.14 14. Date and time: `CmonDateTime` . . . 1461

4.13.15 15. General value functions . . . 1462
4.13.16 16. String functions . . . 1463
4.13.17 17. Mathematical, array, and statistical functions . . . 1464
4.13.18 18. Regular expression functions . . . 1465
4.13.19 19. Cmon, license, and introspection functions . . . 1465
4.13.20 20. Advisor bundle structure . . . 1466
4.13.21 21. Shared helper library (`common/*.js`) . . . 1468
4.14 ClusterControl Client SDK . . . 1470
4.14.1 SDK for various programming languages . . . 1471
4.14.2 Generating Sources from the API definition . . . 1471
4.15 Provider Configuration . . . 1473
4.16 Resources . . . 1473
4.16.1 clustercontrol_db_cluster . . . 1473
4.16.2 clustercontrol_db_cluster_backup . . . 1478
4.16.3 clustercontrol_db_cluster_backup_schedule . . . 1479
4.16.4 clustercontrol_db_cluster_maintenance . . . 1480
4.17 Common fields in the resource definition . . . 1480
4.17.1 Resource – clustercontrol_db_cluster . . . 1480
4.17.2 Scheduling Backups using the clustercontrol_db_cluster_backup_scheduler resource 1480
4.17.3 Taking adhoc backups using the clustercontrol_db_cluster_backup resource . . . 1481
4.17.4 Setting a maintenance window using the clustercontrol_db_cluster_maintenance resource . . . 1481
4.18 Supported backup methods for the respective database types (and vendors) . . . 1482
4.19 Toggling cluster auto-recovery option . . . 1482
4.20 Build . . . 1482
4.21 Create a CMON user for the cmon_exporter . . . 1482
4.22 How to run . . . 1483
4.22.1 Running in foreground . . . 1483
4.22.2 Systemd . . . 1483
4.22.3 Docker . . . 1483
4.23 Access the metrics . . . 1483
4.24 Current metrics . . . 1483
4.25 Install . . . 1485
4.26 Options . . . 1486
4.27 Usage . . . 1486
4.28 Run as systemd . . . 1487
4.29 Sample prometheus.yml . . . 1488
4.30 ClusterControl MCP . . . 1488
4.30.1 How It Works . . . 1488
4.30.2 Tool Annotations . . . 1490
4.30.3 Configuration . . . 1490
4.30.4 Connecting to AI Clients . . . 1495
4.30.5 Verifying the Connection . . . 1497

CHAPTER

ONE

# GETTING STARTED

## 1.1 Getting Started with ClusterControl

Welcome to the **ClusterControl** documentation! This guide will help you get started with ClusterControl, a powerful and user-friendly tool designed to simplify the deployment, management, and monitoring of high-availability database clusters, whether you're working with open-source databases such as MySQL, MariaDB, PostgreSQL, TimescaleDB, MongoDB, Redis, Valkey, and ClickHouse, or proprietary options like SQL Server, MongoDB Enterprise, and EDB. ClusterControl provides a unified interface and advanced features to automate and secure your database operations.




### 1.1.1 What is ClusterControl?

**ClusterControl** is an all-in-one database management platform developed by Severalnines that centralizes the deployment, monitoring, scaling, and security of database clusters into a single interface (GUI, CLI, or API), so managing a cluster no longer means juggling manual configuration, monitoring, and scaling across disconnected tools.

It supports multiple database technologies and offers a single platform for:

- **Automated deployment**: Deploy database clusters with a few clicks, across topologies like primary-replica replication, Galera clusters, and sharded environments.
- **Comprehensive monitoring**: Real-time visibility into cluster health, resource usage, and performance, with customizable dashboards and alerts.
- **Backup and recovery**: Schedule automatic backups, configure retention policies, and restore clusters with ease.
- **Scaling and load balancing**: Scale clusters automatically or manually to absorb workload spikes while keeping them stable and highly available.
- **High availability and failover**: Automated failover and node recovery minimize downtime and keep databases available.
- **Security and compliance**: Manage user permissions, enforce SSL, and track access logs to maintain a secure database environment.

With its focus on ease of use and powerful automation, ClusterControl is ideal for both seasoned database administrators and those new to database cluster management.

### 1.1.2 Editions and releases

ClusterControl comes in 3 editions - **Community**, **Advanced**, and **Enterprise** - within the same binary. Please review the ClusterControl product page for a features comparison between these editions.

All new ClusterControl installations will be automatically configured with a 30-day fully functional Enterprise edition trial license (1). When the license expires, ClusterControl defaults to the Community Edition. To upgrade from Community to Advanced or Enterprise, you need a valid software license. For commercial information, please contact us. { .annotate }

1. This requires an internet connection for automatic license retrieval and activation. For an offline environment, reach out to our Sales department at sales@severalnines.com to request for a free trial license.

Starting from ClusterControl 2.2.0 (September 2024), ClusterControl provides 2 releases:

| Release branch | Version | Release date |
|---|---|---|
| Yearly long-term support (LTS) | 2.4.0 | February 2026 |
| Latest (default) | 2.5.0 | July 2026 |

See Release Notes for details of all ClusterControl's releases.

### 1.1.3 Target users

ClusterControl is a database management and automation platform designed for DBAs (Database Administrators), DevOps teams, system administrators, and IT operations professionals who are responsible for deploying, managing, and maintaining database clusters. Its target users typically have a need for high-availability, scalability, performance monitoring, and automation in database environments.

| Target user | Typical use case |
|---|---|
| DBAs (Database Administrators) | Automating repetitive tasks such as backups, failover management, and upgrades across complex database systems. |
| DevOps engineers | Deploying and managing database clusters as infrastructure-as-code, as part of CI/CD pipelines. |
| System administrators / IT operations | Monitoring database uptime, configuring automated alerts, and ensuring disaster-recovery protocols are in place. |
| Development teams (for large-scale applications) | Scaling databases to handle increased traffic and high-performance queries as application usage grows. |
| Companies managing multiple databases | Centrally managing hybrid environments that run several database engines (MySQL, PostgreSQL, MongoDB, and others) on one platform. |
| Enterprises with high-availability requirements | Ensuring zero-downtime for databases running business-critical applications through automated failover, replication, and clustering. |

### 1.1.4 Next steps

- *Quickstart*: Get a ClusterControl instance up and running in minutes.
- *Installation*: Detailed installation options for your environment.
- *Tutorials*: Step-by-step walkthroughs for deploying your first cluster.

## 1.2 How it Works?

At its core, ClusterControl operates as a central management layer, integrating with multiple database technologies to streamline complex tasks and ensure high availability and performance. Using automation and intelligent monitoring, it allows you to deploy, scale, back up, and recover clusters with minimal manual intervention.

This page details the underlying mechanisms and processes that allow ClusterControl to function as a reliable, secure, and efficient solution for managing and optimizing your entire database infrastructure.

### 1.2.1 Components

Primarily, ClusterControl consists of the following components:

| Component | Package name | Description |
|---|---|---|
| ClusterControl Controller (cmon) | clusterc controll | The brain of ClusterControl. A backend service performing automation, management, monitoring and scheduling tasks on the managed nodes. |
| ClusterControl GUI | clusterc mcc | A modern web user interface to visualize and manage the cluster. It also supports for centralized management of multiple ClusterControl Controller instances, also known as ClusterControl Operations Center (Ops-C). It interacts with the ClusterControl controller via the `cmon-proxy` service and remote procedure call (RPC). |
| ClusterControl CLI | s9s-tools | A command-line tool to manage and monitor clusters provisioned by ClusterControl. |
| ClusterControl SSH | clusterc ssh | A web-based SSH client module implemented using JavaScript and WebSockets. |
| ClusterControl Notifications | clusterc notificat | A service and interface module for integration with third-party messaging and incident management tools. |
| ClusterControl Cloud | clusterc cloud | A service and interface module for integration with cloud providers. |
| ClusterControl Cloud File Manager | clusterc clud | A command-line interface module to interact with cloud storage objects. |
| ClusterControl Proxy | clusterc proxy | A controller proxying service for ClusterControl GUI. |
| ClusterControl Kubernetes Proxy | clusterc kuber-proxy | A proxying service to interact with database clusters running on Kubernetes platform. |
| CusterControl Telemetry | clusterc telemetr | A billing-grade usage metering and accounting of the database-server estate for single or multiple controllers (optional - requires separate installation). |

For a large deployment, ClusterControl provides an additional suite to provision multiple ClusterControl instances at once, called ClusterControl Operations Center (Ops-C). This feature is now merged within ClusterControl GUI and can be activated under the **License** section.

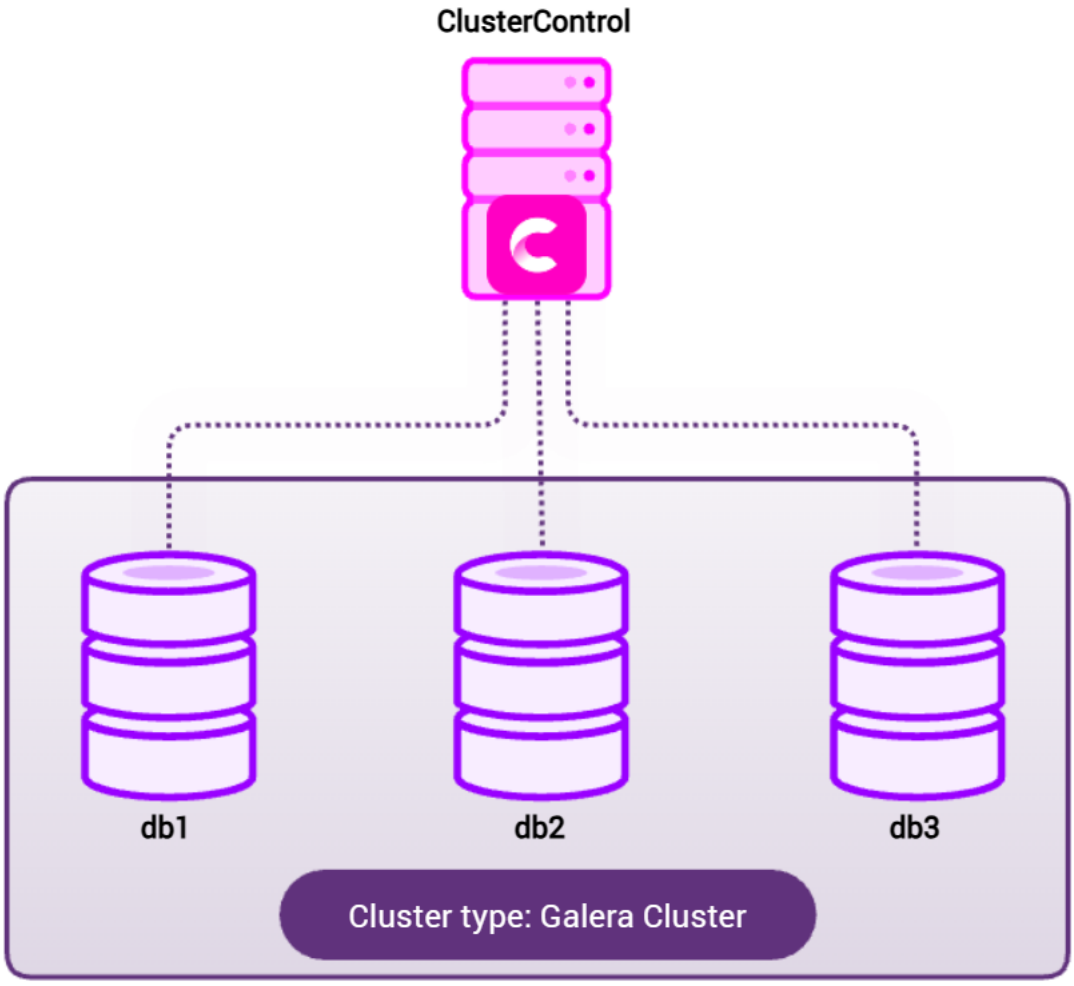
ClusterControl also provides a number of integration components:

| Component | Description |
|---|---|
| Terraform Provider for ClusterControl | Terraform provider that enables interaction with ClusterControl database platform. |
| cmon_exporter | Prometheus exporter for ClusterControl Controller (cmon) service. |
| cmon_sd | An HTTP endpoint that exposes monitoring targets for external Prometheus service discovery. |

See *Components* for more details on every ClusterControl component.

### 1.2.2 Architecture

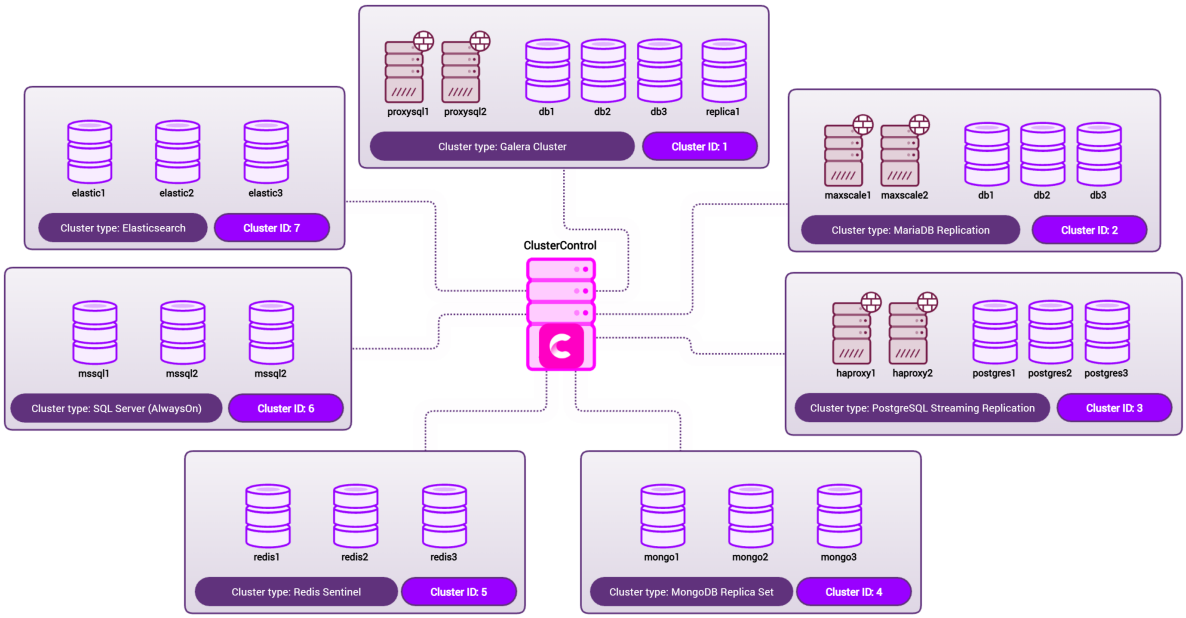
ClusterControl components must reside on a node separate from the database cluster. For instance, a three-node Galera Cluster requires ClusterControl on a fourth, independent node. Following is an example deployment of having a Galera cluster with ClusterControl:




Once the cmon service is initiated, it loads its configuration parameters into the CMON database from `/etc/cmon.cnf` and `/etc/cmon.d/cmon_*.cnf`. Each configuration file corresponds to a unique cluster ID. The service then registers hosts, gathers information, and performs regular checks and scheduled tasks on all managed nodes via SSH. This agentless management relies on passwordless SSH (key-based authentication), utilizing the `ssh_identity` key and `ssh_user` defined in the CMON configuration. For

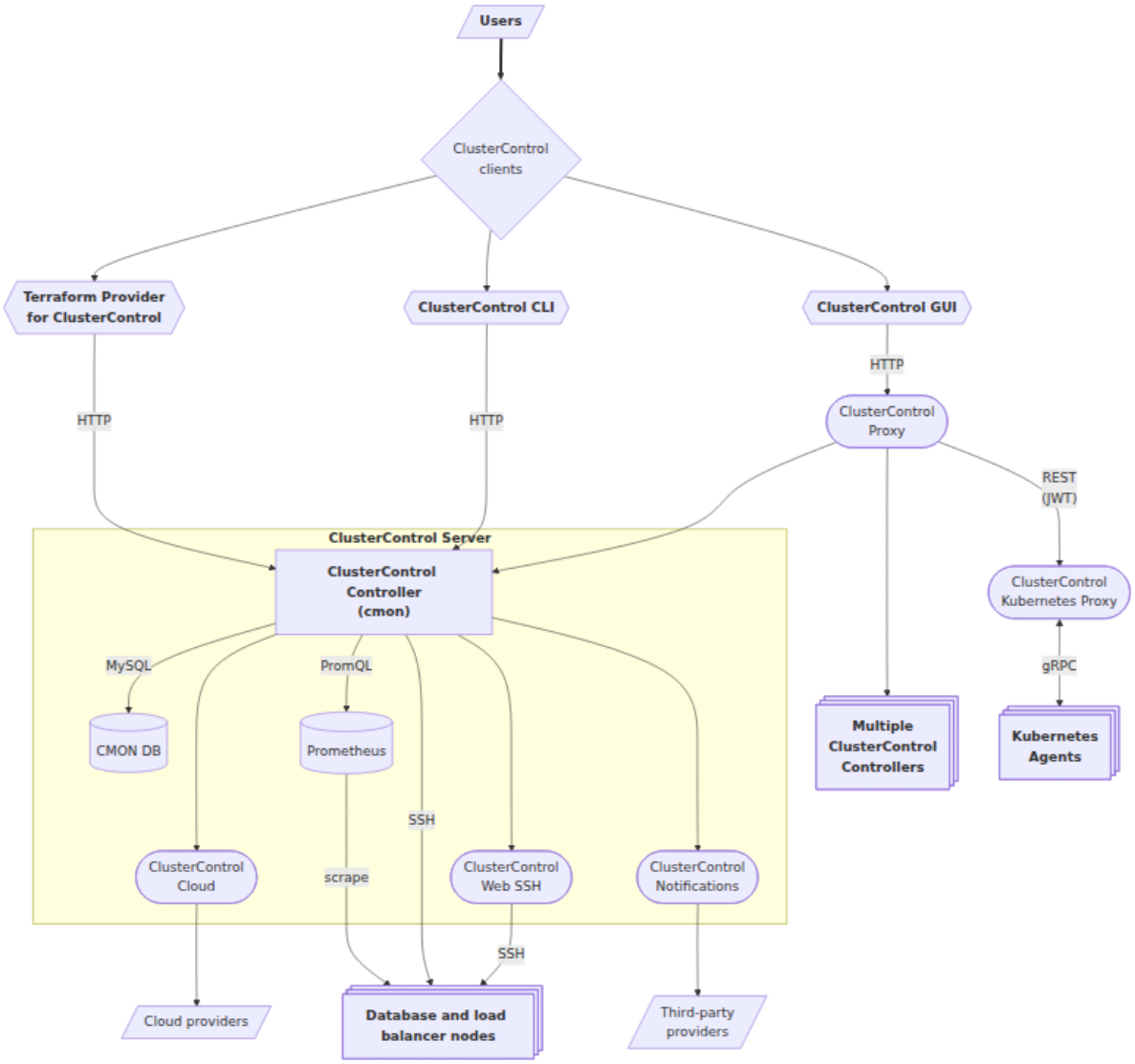
monitoring, ClusterControl employs an agent-based approach using Prometheus and exporters, including `cmonagent` for query monitoring.

Management of the database infrastructure is performed through the ClusterControl GUI, accessible at `https://<ClusterControl_host>`, or via the ClusterControl CLI. Users can begin by importing an existing database cluster or creating a new database server or cluster, either on-premises or in the cloud, provided the target nodes are reachable via SSH. A single ClusterControl instance supports managing multiple clusters of various types, as shown in the following figure:




ClusterControl controller exposes all functionalities through remote procedure calls (RPC) on port 9500 (authenticated by an RPC token), port 9501 (RPC with TLS) and integrates with a number of modules like notifications (9510), cloud (9518) and web SSH (9511). The client components, ClusterControl GUI or ClusterControl CLI interact with those interfaces to retrieve monitoring data (cluster load, host status, alarms, backup status, etc.) or to send management commands (add/remove nodes, run backups, upgrade a cluster, etc.).

The following diagram illustrates the internal architecture of ClusterControl and how the components interact with each other:




ClusterControl has minimal performance impact and will not cause any downtime to your database server or cluster. In fact, it will perform automatic recovery (if enabled) when it finds a failed database node or cluster. See *Failover and Recovery*.

### 1.2.3 Management and automation operations

For tasks such as management and deployment, ClusterControl executes jobs by sending remote commands via SSH to the designated target node. Users only need to install the ClusterControl controller package on the ClusterControl host, and ensure the SSH key-based authentication and the CMON database user grants are properly set up on each of the managed hosts.

This mechanism requires no agent which simplifies the configuration of the whole infrastructure. Agent-based mode of operation is only supported for monitoring jobs, as described in the next section.

### 1.2.4 Monitoring operation

For monitoring operation, ClusterControl uses Prometheus and exporters to store time-series monitoring data. For query monitoring, ClusterControl uses either a query agent (ClusterControl Agent, `cmnd`) or SSH pulling, depending on the cluster type.

Other operations like management, scaling, and deployment are performed through an agentless approach

as described in the *Management and Deployment Operations*.

**Info**

ClusterControl used to support both agentless (SSH sampling) and agent-based monitoring. However, since v1.9.8, ClusterControl defaults to agent-based monitoring using Prometheus. Agentless monitoring is no longer recommended.

### Monitoring tools

ClusterControl uses a Prometheus server (default to port 9090) to store time-series monitoring data, and all monitored nodes will be configured with at least three exporters (depending on the node's role):

| Node type | Exporter name | Port | Package |
|---|---|---|---|
| All | Process exporter | 9011 | process-exporter |
| | Node/system metrics exporter | 9100 | node_exporter |
| MySQL/MariaDB | MySQL exporter | 9104 | mysqld_exporter |
| MongoDB | MongoDB exporter | 9216 | mongodb_exporter |
| PostgreSQL/TimescaleDB | Postgres exporter | 9187 | postgres_exporter |
| Redis/Valkey | Redis exporter | 9121 | redis_exporter |
| SQL Server | SQL exporter | 9999 | sql_exporter |
| Elasticsearch | Elasticsearch exporter | 9114 | elasticsearch_exporter |
| ProxySQL | ProxySQL exporter | 42004 | proxysql_exporter |
| HAProxy | HAProxy exporter | 9101 | haproxy_exporter |

On every monitored host, ClusterControl will configure and daemonize the exporter process using `systemd`. It is recommended to have an Internet connection when installing the necessary packages and automate the Prometheus deployment. For offline installation, the packages must be pre-downloaded into `/var/cache/cmon/packages` on the ClusterControl node. For the list of required packages and links, please refer to `/usr/share/cmon/templates/packages.conf`.

Apart from the Prometheus scrape process, ClusterControl also connects to the process exporter via HTTP calls to determine the process state of the node, and PromQL to visualize and trend the monitoring data.

**Note**

Note ClusterControl depends on a working Prometheus for accurate reporting on management and monitoring data. Therefore, Prometheus and exporter processes are managed by the internal process manager thread. A non-operational Prometheus will have a significant impact on the CMON process.

ClusterControl uses a conservative approach to prevent port conflicts when monitoring multiple instances of the same process on a single host (currently limited to PostgreSQL-based clusters). It automatically assigns an incremental port number for each subsequent instance's exporter.

**Example**

If a host has two PostgreSQL instances (e.g., on ports 5432 and 5437), the first instance's exporter will use the default port (e.g., 9187). The second instance's exporter will be automatically configured to use

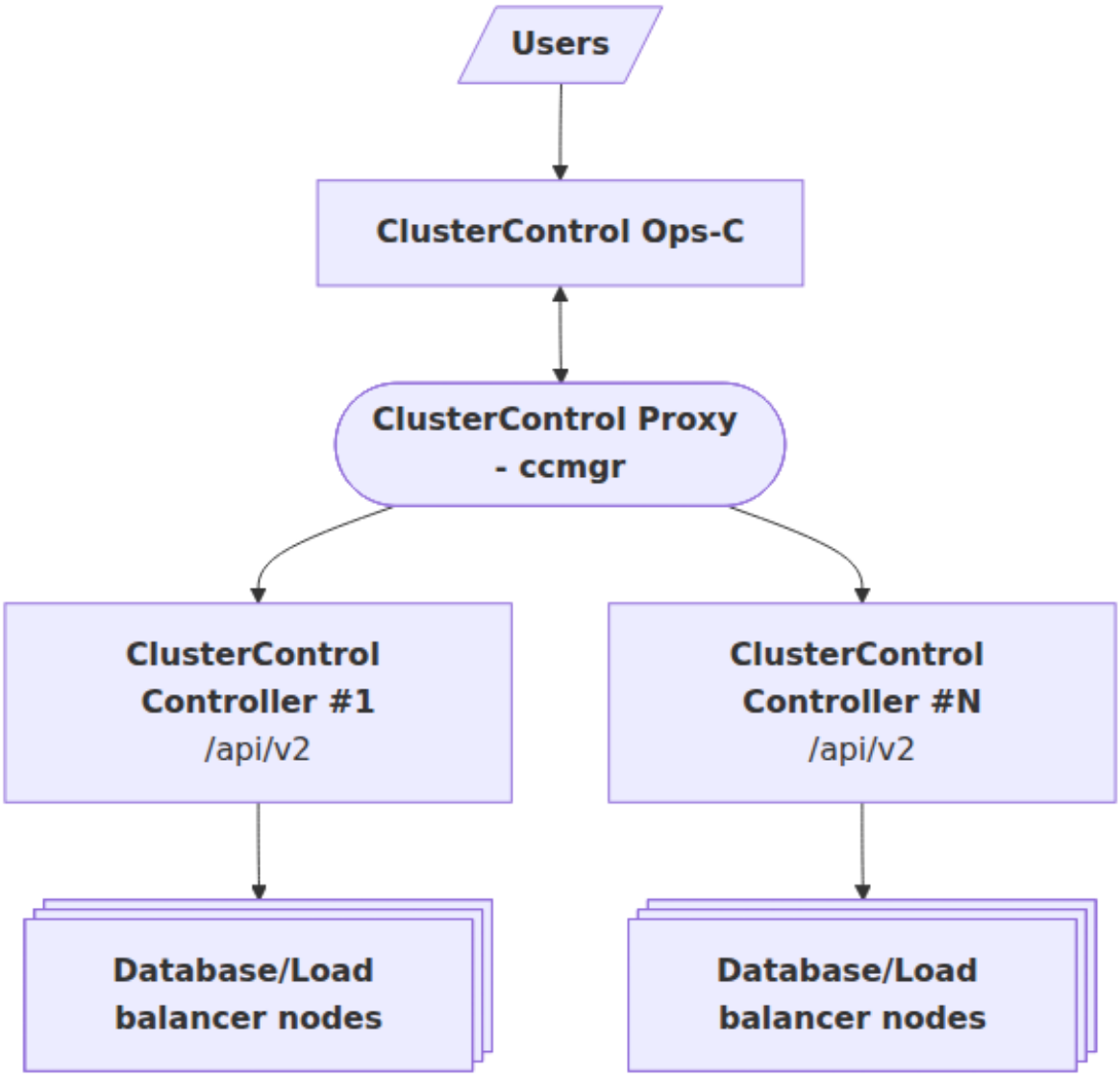
the next available port (e.g., 9188).

### 1.2.5 Multiple controllers management

For large deployments, it is possible to have multiple ClusterControl instances managing your fleet of database clusters. For example, for 1000+ database nodes, you probably need around 5 to 8 ClusterControl instances (depending on the hardware size) to manage and automate the whole set. You can use ClusterControl Ops-C (shortform of "Operation Center") to have a centralized management system to oversee the whole clusters.

ClusterControl Ops-C utilizing the ClusterControl Proxy service (`ccmgr`) to connect to multiple ClusterControl instances to retrieve status and information about all clusters managed by every ClusterControl controller. It aggregates and visualizes overall view of all clusters in real-time, allowing you to have a centralized view of what is happening without the hassle to check and find the corresponding ClusterControl controller.

The following diagram shows how a single ClusterControl Ops-C installation ommunicates with multiple ClusterControl instances:



By default, the ClusterControl Ops-C GUI is disabled and can be activated under the ***Addons*** tab (on ClusterControl versions before 2.5.0 this toggle lived on the License page). When activating, additional configuration is required to create a privileged user to securely integrate and manage multiple ClusterControl controllers. Apart from the standard ClusterControl users authentication method, it also supports authenticating using the existing directory service like LDAP or Active Directory. Simply add a controller by specifying the username and password of ClusterControl user (must belong to admins group) and you are good to go.

**See Also**

- *User Guide → Integration → ClusterControl Ops-C*
- *User Guide → Overview → ClusterControl GUI*
- *Reference Manuals → ClusterControl Proxy*

## 1.3 Supported databases

ClusterControl provides comprehensive support for a wide range of open-source and proprietary databases, enabling users to manage, monitor, and automate their database clusters from a unified interface.

### 1.3.1 Database servers

The following table shows the supported database product with its respective vendor and supported major version:

| Database type | Product name | Vendor | Supported major version | Recommended minimum number of nodes |
|---|---|---|---|---|
| MySQL | MySQL Community Server | Oracle | 8.0, 8.4 | Standalone: 1 node (1 primary) Replication: 2 nodes (1 primary + 1 primary/replica) Group replication: 3 nodes (1 primary + 2 replicas) |
| | Percona Server for MySQL | Percona | 8.0, 8.4 | Standalone: 1 node (1 primary) Replication: 2 nodes (1 primary + 1 primary/replica) |
| | Percona Server for MySQL Pro | Percona | 8.0, 8.4 | Standalone: 1 node (1 primary) Replication: 2 nodes (1 primary + 1 primary/replica) |
| | Percona XtraDB Cluster | Percona | 8.0, 8.4 | Cluster: 3 nodes (3 primaries) |
| | Percona XtraDB Cluster Pro | Percona | 8.4 | Cluster: 3 nodes (3 primaries) |
| MariaDB | MariaDB Community Server | MariaDB | 10.5, 10.6, 10.11, 11.4, 11.8 | Standalone: 1 node (1 primary) Replication: 2 nodes (1 primary + 1 primary/replica) Cluster: 3 nodes (3 primaries) |
| PostgreSQL | PostgreSQL Server | PostgreSQ | 14, 15, 16, 17, 18 | Standalone: 1 node (1 primary) Replication: 2 nodes (1 primary + 1 replica) |
| | PostgreSQL Server | EDB | 14, 15, 16, 17 | Standalone: 1 node (1 primary) Replication: 2 nodes (1 primary + 1 replica) |
| | TimescaleDB Community (TSL) | Times | 14, 15, 16, 17, 18 | Standalone: 1 node (1 primary) Replication: 2 nodes (1 primary + 1 replica) |
| MongoDB | MongoDB Community Server | MongoDB | 7.0, 8.0 | Replica set: 3 nodes (1 primary + 2 replicas) Sharded cluster: 7 nodes (3 config servers + 3 shard servers + 1 mongos) |
| | MongoDB Enterprise Server | MongoDB | 7.0, 8.0 | Replica set: 3 nodes (1 primary + 2 replicas) Sharded cluster: 7 nodes (3 config servers + 3 shard servers + 1 mongos) |
| | Percona Server for MongoDB | Percona | 7.0, 8.0 | Replica set: 3 nodes (1 primary + 2 replicas) Sharded cluster: 7 nodes (3 config servers + 3 shard servers + 1 mongos) |
| Redis | Redis Community Edition | Redis | 6, 7, 8 | Sentinel: 3 nodes ( 1 primary with Sentinel + 2 replicas with Sentinel) Cluster: 6 nodes ( 3 primaries + 3 replicas) |
| | Valkey | LF Projec | 7, 8 | Sentinel: 3 nodes ( 1 primary with Sentinel + 2 replicas with Sentinel) Cluster: 6 nodes ( 3 primaries + 3 replicas) |
| | Valkey | Canonical | 7 | Sentinel: 3 nodes ( 1 primary with Sentinel + 2 replicas with Sentinel) Cluster: 6 nodes ( 3 primaries + 3 replicas) |
| SQL Server | Microsoft SQL Server for Linux | Microsof | 2019 (15.x), 2022 (16.x), 2025(17.x) | Standalone: 1 node (1 primary) Always On availability group: 2 nodes (1 primary + 1 replica) |
| Elasticsearch | Elasticsearch | Elastic | 7.17, 8.15, 8.3 | Standalone cluster: 1 node (1 eligible-master/data node) High availability cluster: 5 nodes (3 eligible-master nodes + 2 data nodes) |
| Column-oriented OLAP | ClickHouse | ClickHouse | 24.8, 25.3 | Standalone: 1 node (1 replica) Replicated cluster: 3 nodes (3 replicas with embedded Keeper) |

**Note**

Note ClusterControl only supports managing x86-64 architecture hosts (x86_64, x64, AMD64, Intel64).

**Warning**

Attention

Some database products are only supported on selected operating systems. See *Operating System*.

### 1.3.2 Auxiliary tools

Apart from the database software, ClusterControl also supports the following open-source auxiliary tools:

| Type | Product name | Repository |
|---|---|---|
| Load balancer | ProxySQL | Percona |
| | MariaDB MaxScale | MariaDB |
| | HAProxy | EPEL/OS |
| | Keepalived | EPEL/OS |
| | garbd | Percona/MariaDB |
| Connection pooler | PgBouncer | PGDG |
| Backup | pgBackRest | PGDG |
| | Percona Backup for MongoDB | Percona |
| Monitoring | Prometheus | Prometheus |

## 1.4 Key Features

Explore ClusterControl's key features below to understand how it can support your database management needs from end to end.

### 1.4.1 All cluster types

The following are features that are available across all cluster types supported by ClusterControl:

#### Security and control

- LDAP/Active Directory authentication. ^==Enterprise==^{: .badge-enterprise}
- Key and certificate management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Role-based access control. ^==Enterprise==^{: .badge-enterprise}
- Integration with Hashicorp Vault. ^==Enterprise==^{: .badge-enterprise}
- Multi-controller management using ClusterControl Operations Center (Ops-C) ^==Enterprise==^{: .badge-enterprise}

### Monitoring and Alerting

- Host monitoring.
- Topology visualizer.
- Advisors.
- Email notification with fine-grained delivery configuration based on severity. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Integration with messaging tools like Slack, Telegram and webhook. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Integration with incident management platform like Pagerduty, Splunk On-Call (VictorOps), ServiceNow, OpsGenie and ilert. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Jobs monitoring.
- Automatic alarms for various critical and warning events.
- Watchlist - Monitoring dashboards in full-screen view (kiosk-mode).

### Cloud storage

- Amazon S3. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Google Cloud Storage. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Microsoft Azure Blob Storage. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Any S3-compatible storage providers. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}

### Node management

- Schedule maintenance mode and tracking.
- Web-based SSH console.
- Reboot host. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Stop/Start/Restart services. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}

### Audit and reporting

- Usage metering and operator billing. ^==Enterprise==^{: .badge-enterprise}
- Operational reports (system report, database availability report, backup report, upgrade report, database growth report, schema change report).
- ClusterControl audit log and tracking.
- Activity center.
- Error reports.
- Centralized database and load balancer logs.

### 1.4.2 Cluster-specific features

#### MySQL/MariaDB/Percona

- Deployment automation:
  - Deploy MySQL/MariaDB standalone, Galera Cluster, replication (primary-replica and primary-primary).
  - Deploy MySQL group replication.
  - Support for various MySQL flavors (Percona Server for MySQL, Percona Server for MySQL Pro, Oracle MySQL Community Server, MariaDB Community Server).
  - Import existing MySQL/MariaDB standalone, replication or Galera Cluster.
  - Cluster-to-Cluster replication (Galera Cluster). ^==Enterprise==^{: .badge-enterprise}
  - Deploy MySQL replication on Kubernetes.
- Backup management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - On-demand or scheduled logical or physical backups (partial, full and incremental).
  - Support for mysqldump, Percona Xtrabackup, Mariabackup.
  - Point-in-Time Recovery (PITR).
  - Backup to local disk and remote storage (S3-compatible object storage, NFS).
  - Backup verification.
  - Backup binary logs.
  - Restore from local backup or cloud.
  - Restore external backup.
  - Stream backup directly to cloud.
- Failover and recovery: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Automated failover for primary-replica and Galera clusters.
  - Node auto-recovery and healing.
  - Promote replica to primary.
  - Rebuild replica from primary.
  - Change replication source.
  - Read-only and read-write management.
- Scaling: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Scale-out by adding new database nodes (replicas, delayed replicas or Galera nodes).
  - Scale-out load balancer nodes.
  - Scaling support for MySQL Replication and Galera.
- Monitoring and alerting:
  - Database stats monitoring.

    - Load balancer stats monitoring.
    - Track replication lag, Galera state, disk space, queries per second.
    - Customizable threshold and alerting.
    - Database and load balancer logs.
- Load balancing: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Integrates with HAProxy, ProxySQL and MariaDB MaxScale for automated load balancing.
    - ProxySQL manager (query rules, users, query cache, routing)
    - Integrates with Keepalived for virtual IP address.
    - Integrates with garbd (Galera arbitrator daemon).
- Security: ^==Enterprise==^{: .badge-enterprise}
    - SSL encryption for client-server communication.
    - SSL encryption between server-server communication (Galera Cluster and replication).
    - User management, audit logging, and access control.
    - Backup encryption.
- Performance management:
    - Query monitoring, slow query detection, schema analyzer, query optimizer.
    - Database growth.
    - Advisors for tuning performance and query improvement.
    - Database and load balancer configuration management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Database user management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Database upgrades and patching: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Automated rolling upgrades for minor versions.
    - Schema upgrade and migration management.

### PostgreSQL/TimescaleDB/EDB

- Deployment automation:
    - Deploy PostgreSQL standalone or streaming replication (primary-replica).
    - Deploy PostgreSQL logical replication with possibility of bi-directional replication.
    - Support for various PostgreSQL flavors (PostgreSQL, TimescaleDB, EDB Postgres Server).
    - Import existing PostgreSQL standalone and streaming replication.
    - Cluster-to-Cluster replication.^==Enterprise==^{: .badge-enterprise}
    - Deploy PostgreSQL streaming replication on Kubernetes.

- Backup management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - On-demand and scheduled logical or physical backups (partial, full and incremental).
  - Support for pgdump, pg_basebackup, PgBackRest.
  - Point-in-Time Recovery (PITR).
  - Backup to local disk and remote storage (S3-compatible object storage, NFS).
  - Backup verification.
  - Restore from local backup or cloud.
  - WAL archiving.
  - Dedicated backup repository host.
  - Stream backup directly to cloud.
- Failover and recovery: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Automated failover for primary-replica streaming replication clusters.
  - Node automatic recovery and healing.
  - Promote replica to primary.
  - Rebuild replica from primary.
  - Change replication source.
- Scaling: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Scale-out by adding new database nodes (replicas).
  - Scale-out load balancer nodes.
  - Scaling support for streaming replication.
- Monitoring and alerting:
  - Database stats monitoring.
  - Load balancer stats monitoring.
  - Track replication lag, database state, disk space, queries per second.
  - Customizable threshold and alerting.
  - Database and load balancer logs.
- Load balancing: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Integrates with HAProxy for automated load balancing.
  - Integrates with Keepalived for virtual IP address.
  - pgBouncer pool management.
- Security: ^==Enterprise==^{: .badge-enterprise}
  - SSL encryption for client-server communication.
  - SSL encryption between server-server communication (replication).

    - User management, audit logging, and access control.
    - Backup encryption.
- Performance management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Query monitoring, slow query detection, query optimizer.
    - Database growth.
    - Advisors for tuning performance and query improvement.
    - Database and load balancer configuration management.
    - Database user management.
- Database upgrades and patching: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Automated rolling upgrades for major and minor versions.
- Plugins/extensions management:
    - TimescaleDB
    - pgvector
    - PostGIS

### MongoDB

- Deployment automation:
    - Deploy a new MongoDB standalone, replica set or sharded cluster (single or multiple shards).
    - Support for various MongoDB flavors (MongoDB Community, MongoDB Enterprise, Percona Server for MongoDB).
    - Import existing MongoDB Replica Set or Sharded Cluster.
- Backup management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - On-demand and scheduled logical backups.
    - Support for mongodump and Percona Backup for MongoDB.
    - Backup to local disk and remote storage (S3-compatible object storage, NFS).
    - Restore from local backup or cloud.
- Failover and recovery: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Automated failover for primary-replica streaming replication clusters.
    - Node auto-recovery and healing.
    - Step down primary for auto election.
    - Freeze node.
- Scaling: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Scale-out by adding new mongod or arbiter (standalone or replica set)

    - Scale-out by adding new mongodb, config or mongos nodes (sharded cluster).
    - Scale-out by adding new replica set shards (sharded cluster)
- Monitoring and alerting:
    - Database stats monitoring.
    - Mongos router monitoring.
    - Track replication lag, database state, disk space, queries per second.
    - Customizable threshold and alerting.
- Security: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - SSL encryption for client-server communication.
    - SSL encryption between server-server communication (replication).
    - User management, audit logging, and access control.
    - Backup encryption.
- Performance management:
    - Ops monitoring.
    - Database growth.
    - Advisors for tuning performance and query improvement.
    - Database configuration management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Database upgrades and patching: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Automated rolling upgrades for minor versions.

### Redis/Valkey

- Deployment automation:
    - Deploy a new Redis/Valkey Sentinel or cluster.
    - Support for various Redis flavors (Redis Community and Valkey).
    - Import existing Redis/Valkey Sentinel or cluster.
- Backup management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - On-demand and scheduled RDB (Redis Database) and AOF (Append Only File) backups.
    - Backup to local disk and remote storage (S3-compatible object storage, NFS).
    - Restore from local backup or cloud.
    - Backup verification.
- Failover and recovery: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Automated failover for primary-replica replication clusters (Sentinel).

    - Node auto-recovery and healing.
    - Promote replica to primary.
    - Change replication source.
- Scaling: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Scale-out by adding new Redis node (sentinel or cluster mode)
- Monitoring and alerting:
    - Database stats monitoring.
    - Track database state, disk space, queries per second.
    - Customizable threshold and alerting.
- Security: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - SSL encryption for client-server communication.
    - SSL encryption between server-server communication.
    - User management, audit logging, and access control.
    - Backup encryption.
- Performance management:
    - Database growth.
    - Advisors for tuning performance and query improvement.
    - Database configuration management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Database user management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
- Database upgrades and patching: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Automated rolling upgrades for minor versions.

### SQL Server

- Deployment automation:
    - Deploy a new SQL Server standalone or Always On Availability Group cluster.
    - Import existing SQL Server standalone or Always On Available Group cluster.
- Backup management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - On-demand and scheduled physical backups.
    - Support for full, differential, and transaction log backups.
    - Backup to local disk and remote storage (S3-compatible object storage, NFS).
    - Restore from local backup or cloud.
    - Point-in-time recovery.

- Failover and recovery: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Automated failover for primary-replica AlwaysOn availability group.
  - Node auto-recovery and healing.
  - Promote replica to primary.
  - Rebuild replica.
- Scaling: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Scale-out by adding new SQL Server node.
- Monitoring and alerting:
  - Database stats monitoring.
  - Track database state, disk space, queries per second.
  - Customizable threshold and alerting.
- Security: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - SSL encryption for client-server communication.
  - SSL encryption between server-server communication.
  - User management, audit logging, and access control.
- Performance management:
  - Database growth.
  - Advisors for tuning performance and query improvement.
  - Database configuration management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}

### Elasticsearch

- Deployment automation:
  - Deploy a new Elasticsearch single-node cluster or high-availability cluster.
  - Import existing Elasticsearch single-node cluster or high-availability cluster.
- Backup management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - On-demand and scheduled Elastic snapshot backup.
  - Backup to local disk and remote storage (S3-compatible object storage, NFS).
  - Restore from local backup or cloud.
- Failover and recovery: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
  - Automated failover for high availability setup.
  - Node auto-recovery and healing.
- Scaling: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}

    - Scale-out by adding new data node.
- Monitoring and alerting:
    - Database stats monitoring.
    - Track database state, disk space, queries per second.
    - Customizable threshold and alerting.
- Security: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - SSL encryption for client-server communication.
    - SSL encryption between server-server communication.
    - User management, audit logging, and access control.
- Performance management:
    - Database growth.
    - Advisors for tuning performance and query improvement.
    - Database configuration management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}

### ClickHouse

- Deployment automation:
    - Deploy a new ClickHouse single-node, replicated (multi-node with embedded Keeper) or sharded cluster.
    - Import existing ClickHouse single-node, replicated or sharded cluster.
- Backup management: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - On-demand and scheduled backups using ClickHouse native BACKUP/RESTORE (full and incremental).
    - Backup to local disk and remote storage (S3-compatible object storage, NFS).
    - Restore from local backup or cloud.
    - Restore to the same or a different cluster.
- Failover and recovery: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Automated failover for replicated clusters via embedded ClickHouse Keeper quorum.
    - Node auto-recovery and healing.
- Scaling: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - Scale-out by adding new replica node (replicated cluster).
    - Scale-out by adding new shard (sharded cluster).
- Monitoring and alerting:
    - Database stats monitoring.
    - Track replication lag, Keeper quorum, disk space, queries per second and data part count.

    - Customizable threshold and alerting.
- Security: ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}
    - SSL encryption for client-server communication.
    - SSL encryption between server-server communication.
    - User management, audit logging, and access control.
- Performance management:
    - Database growth.
    - Advisors for tuning performance and query improvement.
    - Database configuration management. ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}

**See also**

- Feature comparison: ClusterControl product page

## 1.5 Requirements

Before setting up ClusterControl, it's essential to ensure that your environment meets the necessary requirements for a smooth installation and optimal performance. ClusterControl is designed to work across a variety of infrastructure setups, from on-premise servers to cloud environments, and supports multiple database technologies.

This page outlines the hardware, software, network, and database-specific requirements needed to successfully deploy and run ClusterControl.

### 1.5.1 Hardware

The following table shows the recommended hardware specifications for a ClusterControl host:

| Aspect | Minimum | Small (3-15 nodes) | Medium (15-50 nodes) | Large (50-150 nodes) |
|---|---|---|---|---|
| CPU architecture | x86_64 only | | | |
| CPU | 2 cores | 4 cores | 16 cores | 24 cores |
| RAM | 2 GB | 8 GB | 16 GB | 32 GB |
| Disk space | 20 GB | 50 GB | 150 GB | 300 GB |

**Tip**

For very large deployments, you probably need more than one ClusterControl instances to manage the whole fleet. Use ClusterControl Operational Center (Ops-C) to manage these ClusterControl instances. See *Integration → ClusterControl Ops-C*.

### Capacity planning

In regards to capacity planning, to simply put into perspective, we have seen real-world use-case where a single ClusterControl server in production with 24 CPUs, 32GB RAM, 150 GB disk space (currently 92% usage) to manage 67 clusters with around 130 nodes (database + load balancers).

We have also covered optimizing the performance of ClusterControl in this blog post, How to Optimize Performance of ClusterControl and Its Components.

### CPU

The CPU capacity of the ClusterControl server plays a crucial role in handling various tasks such as monitoring, managing, and orchestrating database clusters. To accommodate a significant workload and ensure scalability for future growth, we recommend a minimum of 8 to 24 CPU cores with multi-threading capabilities.

### RAM

Memory availability directly impacts the performance and responsiveness of ClusterControl, especially when handling a large number of database clusters and nodes. For optimal operation, we recommend 8 to 32 GB of RAM. However, considering the increasing complexity of database environments and potential future expansions, provisioning additional memory is advisable.

During the installation stage, the installer script tunes the necessary parameters for MySQL and Apache according to the amount of RAM in the host. If you have increased the RAM after the installation, it is recommended to revisit the tuning by looking at this blog post, How to Optimize Performance of ClusterControl and Its Components.

### Disk space

Disk space requirements primarily depend on the size of the monitoring data, logs, backups, and other associated files generated by ClusterControl. Given the example use-case disk utilization of 92% for 150GB, it is recommended to allocate at least 300GB of disk space for the ClusterControl server. This ensures sufficient headroom for data growth and temporary storage needs.

If you use ClusterControl as the centralized backup repository, consider adding more disk space, adding another disk or mounting a networked file system like NFS, SMB or iSCSI to cater to this.

If you use Prometheus and agent-based monitoring, you may also use a dedicated Prometheus server (by default Prometheus will be installed on the ClusterControl server) to scale up the monitoring resources. Prometheus defaults to 15 days retention period and if you require longer retention period, adjust the disk allocation for `/var/lib/prometheus` partition accordingly.

### Network bandwidth

ClusterControl relies on network communication for monitoring, provisioning, and managing database clusters. Therefore, it's essential to ensure adequate network bandwidth to support the expected workload. A Gigabit Ethernet connection or higher is recommended for optimal performance. For inter-WAN communication, consider placing the ClusterControl server at the nearest of the most critical segment (production/primary).

#### Additional considerations

#### High availability

For mission-critical environments, implementing high availability (HA) for the ClusterControl server is highly recommended. This involves deploying redundant ClusterControl nodes in an active-passive or active-active configuration to ensure continuous operation and fault tolerance. See *Redundancy & High Availability*.

#### Storage performance

While sufficient disk space is essential, equally important is the performance of the storage subsystem. Utilizing fast and reliable storage technologies such as solid-state drives (SSDs) or high-performance RAID arrays can significantly improve the responsiveness of ClusterControl, especially during data-intensive operations like backups, restores, scaling and reporting.

#### Monitoring and maintenance

Regular monitoring of hardware resources, including CPU, RAM, disk space, and network utilization, is essential to identify potential bottlenecks or capacity issues proactively. Additionally, periodic maintenance tasks such as disk cleanup, log rotation, and performance tuning should be scheduled to optimize the ClusterControl server's performance and stability.

### 1.5.2 Operating system

#### ClusterControl

ClusterControl has been tested on the following operating systems:

- Red Hat Enterprise Linux 8.x/9.x
- Rocky Linux 8.x/9.x
- AlmaLinux 8.x/9.x
- Ubuntu 22.04/24.04 LTS
- Debian 10.x/11.x/12.x

#### Monitored nodes

For the monitored nodes, the following operating systems are supported:

- Red Hat Enterprise Linux 8.x/9.x
- Rocky Linux 8.x/9.x
- AlmaLinux 8.x/9.x
- Ubuntu 20.04/22.04/24.04 LTS
- Debian 10.x/11.x/12.x

Should the cluster deployment or import job fail due to unsupported operating system, an informational message will display the operating systems supported by ClusterControl for the selected database technology.

**Warning**

Attention

- Mixing Red Hat-based and Debian-based nodes in a database cluster is not supported.

### 1.5.3 Software dependencies

The following software and packages are required by ClusterControl:

- MySQL client/server (5.7 or later) or MariaDB client/server (10.0 or later)
- OpenSSH server/client
- NTP server – All servers' time must be synced under one time zone
- `socat` or `netcat` – for streaming backups

**Note**

Note If ClusterControl is installed via installation script (install-cc) or package manager (yum/dnf/apt), all dependencies will be automatically satisfied.

### 1.5.4 Firewall and security groups

It is important to secure the ClusterControl host and the database cluster. It is recommended for users to isolate the database infrastructure from the public Internet and just whitelist the known hosts or networks to reach the database cluster.

ClusterControl requires ports used by the following services to be accessible or enabled:

- ICMP (echo reply/request).
- SSH (default is 22).
- HTTPS (default is 443 - Web server is handled by ClusterControl Proxy).
- MySQL (default is 3306).
- CMON RPC (default is 9500).
- CMON RPC TLS (default is 9501).
- CMON Events (default is 9510).
- CMON SSH (default is 9511).
- CMON Cloud (default is 9518).
- Streaming port for backups through socat/netcat (default is 9999, configurable under ***Backup Settings***).

ClusterControl supports various database and application vendors and each has its own set of standard ports that need to be reachable. Following ports and services need to be reachable from ClusterControl node to the managed nodes (unless the target services are deliberately configured on non-default ports):

| Node type | Default port (Service) |
|---|---|
| *All managed nodes* | - 22 (SSH) - ICMP (echo reply/request) - 9100 (Node exporter) - 9011 (Process exporter) |
| MySQL/MariaDB (standalone and replication) | - 3306 (MySQL) - 9104 (MySQL exporter) - 4433 (query monitor) |
| Galera Cluster (MariaDB Galera Cluster/Percona XtraDB Cluster) | - 3306 (MySQL) - 4444 (SST) - 4567 TCP/UDP (Galera) - 4568 (Galera IST) - 9200 (HAProxy health check) - 9104 (MySQL exporter) - 4433 (query monitor) |
| MongoDB replica set | - 27017 (mongod) - 9126 (MongoDB exporter) |
| MongoDB sharded cluster | - 27018 (mongod) - 27017 (mongos) - 27019 (config server) - 9216 (MongoDB exporter) |
| PostgreSQL/TimescaleDB | - 5432 (PostgreSQL) - 9200 (HAProxy health check) - 9187 (PostgreSQL exporter) - 4433 (query monitor) |
| HAProxy | - 9600 (HAProxy stats) - 3307 (MySQL load-balanced) - 3308 (MySQL load-balanced read-only) - 5433 (PostgreSQL load-balanced) - 5434 (PostgreSQL load-balanced read-only) - 9101 (HAProxy exporter) |
| MariaDB MaxScale | - 6603 (MaxCtrl - CLI) - 4006 (Round robin listener) - 4008 (Read/Write split listener) - 4442 (Debug information) |
| Keepalived | - IP protocol 112 (VRRP) |
| Galera Arbitrator (garbd) | - 4567 (Galera) |
| ProxySQL | - 6032 (ProxySQL Admin) - 6033 (MySQL load-balanced) - 42004 (ProxySQL exporter) |
| Prometheus | - 9090 (Prometheus) |
| Microsoft SQL Server | - 1433 (SQL Server) - 9999 (SQL Server exporter) |
| Redis/Valkey (Sentinel) | - 6379 (Redis) - 26379 (Sentinel) - 9121 (Redis exporter) |
| Redis/Valkey (Cluster) | - 6379 (Redis) - 16379 (Cluster bus) - 9121 (Redis exporter) |
| Elasticsearch | - 9200 (Elastic HTTP/Transfer) - 9114 (Elasticsearch exporter) |
| ClickHouse | - 9440 (Native protocol) - 8443 (HTTP API) - 9010 (Interserver replication) - 9281 (Keeper client/coordination) - 9234 (Keeper Raft) |

### 1.5.5 Operating system user

ClusterControl controller (cmon) process requires a dedicated operating system user to perform various management and monitoring commands on the managed nodes. The value of `os_user` or `sshuser` in CMON configuration file, must exist on all managed nodes and it should have the ability to perform superuser commands.

You are recommended to install ClusterControl as 'root', and running as root is the easiest option. If you perform the installation using another user other than 'root', the following must be true:

- The OS user must exist on all nodes
- The OS user must not be `mysql`
- The `sudo` or `pbrun` or `doas` program must be installed on all hosts
- The OS user must be allowed to do sudo, i.e, it must be in sudoers
- The OS user must be configured with the proper `PATH` environment variable. The following environment `PATH` is expected for user `myuser`:

```
PATH=/
↪usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/myuser/.local/bin:/hom
```

**Warning**

Attention

ClusterControl recommends configuring passwordless sudo access for the SSH user to ensure full functionality. Restricting `sudo` to a subset of commands is not supported and may cause some operations to fail, including cluster deployment, service control, failover, recovery, backup restoration, and upgrades.

This recommendation is based on how the controller executes privileged commands on managed hosts:

- **Commands are executed through a shell.** ClusterControl runs privileged operations using `sudo bash -c '<command>'`. As a result, `sudo` authorizes the shell (`bash`) rather than the underlying commands (such as `systemctl`, `xtrabackup`, or `yum`).
- **Commands are generated dynamically.** Many commands are built at runtime and vary depending on the operation, host configuration, file paths, package versions, and other environment-specific information. Because these command strings are not fixed, maintaining a per-command `sudoers` allowlist is not practical.
- **A single SSH user is used for all management tasks.** ClusterControl uses the configured SSH user for all privileged operations, including deployment, backups, recovery, upgrades, monitoring, and service management. Different `sudo` permissions cannot currently be assigned to individual features.

Given this execution model, limiting `sudo` to specific commands is not supported and may prevent some management operations from completing successfully.

For sudoers, using passwordless sudo is recommended (privilege escalation without password). To set up a passwordless sudo user, open `/etc/sudoers` via text editor and add the following line at the end. Replace `<os_user>` with the sudo username of your choice:

```
  <os_user> ALL=(ALL) NOPASSWD: ALL
```

Open a new terminal to verify if it works. You should now be able to run the following command without entering a password:

```
sudo ls /usr
```

**Note**

Note If using passwordless sudo is not an option, do not forget to specify the sudo password when deploying a new or importing an existing database cluster.

You can also verify this with the SSH command line used by CMON (assuming SSH key-based authentication has been set up correctly):

```
ssh -qt <os_user>@<ip_address_or_hostname> "sudo ls /usr"
```

Where `<os_user>` is the name of the user you intend to use by ClusterControl for management purposes,

and <ip_address_or_hostname> is the IP address or hostname of a node in your cluster.

For pbrun, a commercial access management solution that provides controlled elevation of privileges in a Unix/Linux environment, implementing strong passwords for users is a best practice. The installation of pbrun as follow:

```
# Extract installation packages
  tar -xzvf powerbroker-x.x.x.tar.gz

# Run Installer
  cd powerbroker-x.x.x

# Verify installation
  pbrun -v

# Initialize configuration
  pbrun --init
```

Roles and users are defined in /etc/pb.conf. This involves configuring pbrun and registering <os_user> with pbrun.

```
# Add user to PowerBroker
  pb_useradd os_user

# Assign roles
  pb_roleadd os_user role_name
```

Verify if the <os_user> can run some root command using pbrun:

```
ssh user@remote-host "pbrun /usr/bin/systemctl restart nginx"
```

Where <os_user> is the name of the user you intend to use by ClusterControl for management purposes, and <ip_address_or_hostname> is the IP address or hostname of a node in your cluster.

For doas, a lightweight command-line tool, similar to sudo, for running commands as another user (typically root). It is simpler, more secure, uses a minimal /etc/doas.conf file, and is common on OpenBSD and minimalist Linux. The installation of doas as follow:

### Red Hat/CentOS

Installation of doas command line utility in Redhat based:

1. Install the package from repository.

   ```
   dnf install opendoas
   ```

2. Create config file.

   ```
   touch /etc/doas.conf
   chmod 600 /etc/doas.conf
   ```

3. Add <os_user> in the doas.conf config file.

```
permit os_user
```

#### Debian/Ubuntu

Installation of `doas` command line utility in Debian based:

1. Install the package from repository.

   ```
   apt update
   apt install doas
   ```

   If the package not available, try:

   ```
   apt install opendoas
   ```

2. Create config file.

   ```
   touch /etc/doas.conf
   chmod 600 /etc/doas.conf
   ```

3. Add `<os_user>` in the `doas.conf` config file.

   ```
   permit os_user
   ```

Verify if the `<os_user>` can execute a root command:

```
ssh os_user@remote-host "doas systemctl restart nginx"
```

Where `<os_user>` is the name of the user you intend to use by ClusterControl for management purposes, and `<ip_address_or_hostname>` is the IP address or hostname of a node in your cluster.

### 1.5.6 SSH key-based authentication

Proper SSH key-based authentication setup from ClusterControl node to all managed nodes is mandatory. Before performing any operation on the managed node, the node must be accessible via SSH without using a password by using the key-based authentication instead.

ClusterControl uses `libssh` (a multiplatform C library using SSHv2 protocol) which supports the following public-key algorithms:

- ssh-rsa
- rsa-sha2-512
- rsa-sha2-256
- ssh-dss
- ssh-ed25519
- ecdsa-sha2-nistp256
- ecdsa-sha2-nistp384
- ecdsa-sha2-nistp521

**Note**

Note Take note that ClusterControl is fully tested with the RSA public key. Other supported key types should work in most cases.

### Setting up key-based authentication

To set up an SSH key-based authentication, make sure you generate SSH keys (private and public keys) and copy the public key from the ClusterControl host as the designated user to the target host.

**Tip**

It is not necessary to setup two-way key-based authentication SSH, e.g: from the managed database node to the ClusterControl node.

To generate an SSH key, use the `ssh-keygen` command which is available with the OpenSSH-client package. On ClusterControl node:

```
$ whoami
root
$ ssh-keygen -t rsa      # press Enter on all prompts
```

The above command will generate SSH RSA private and public keys under the user's home directory, `/root/.ssh/`. The private key, `id_rsa` has to be kept secure on the node. The public key, `id_rsa.pub` should be copied over to all nodes that want to be accessed by ClusterControl passwordlessly.

The next step is to copy the SSH public key to all nodes. You may use the `ssh-copy-id` command to achieve this if the destination node supports password authentication:

```
ssh-copy-id <os_user>@<target_node>
```

**Example**

```
$ whoami
root
$ ls -1 ~/.ssh/id*
/root/.ssh/id_rsa
/root/.ssh/id_rsa.pub
$ ssh-copy-id root@192.168.0.10     # specify the root password of␣
↪192.168.0.10 if prompted
```

The command `ssh-copy-id` will simply copy the public key from the source server and add it into the destination server's authorized key list, default to `~/.ssh/autohorized_keys` of the authenticated SSH user. If password authentication is disabled, and the target node requires a key-based authentication, you can use the `-o` flag to customize the SSH option:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.0.10
```

Alternatively, you can manually copy the public key to the target nodes. On ClusterControl node, copy the content of SSH public key located at `~/.ssh/id_rsa.pub` and paste it into `~/.ssh/authorized_keys` of all target nodes.

**Example**

The following example shows how a root user on the ClusterControl host (192.168.0.10) generates and copies an SSH key to databases hosts (192.168.0.11, 192.168.0.12, 192.168.0.13):

```
$ whoami
root
$ ssh-keygen -t rsa            # press Enter on all prompts
$ ls -1 ~/.ssh/id*
/root/.ssh/id_rsa
/root/.ssh/id_rsa.pub
$ ssh-copy-id 192.168.0.11     # specify the root password of 192.168.0.11␣
↪if prompted
$ ssh-copy-id 192.168.0.12     # specify the root password of 192.168.0.12␣
↪if prompted
$ ssh-copy-id 192.168.0.13     # specify the root password of 192.168.0.13␣
↪if prompted
```

If you are running as a sudo user e.g "sysadmin", here is an example:

```
$ whoami
sysadmin
$ ssh-keygen -t rsa            # press Enter on all prompts
$ ls -1 ~/.ssh/id*
/home/sysadmin/.ssh/id_rsa
/home/sysadmin/.ssh/id_rsa.pub
$ ssh-copy-id 192.168.0.11     # specify the sysadmin password of␣
↪192.168.0.11 if prompted
$ ssh-copy-id 192.168.0.12     # specify the sysadmin password of␣
↪192.168.0.12 if prompted
$ ssh-copy-id 192.168.0.13     # specify the sysadmin password of␣
↪192.168.0.13 if prompted
```

You should be able to SSH from ClusterControl to the other server(s) without a password:

```
ssh sysadmin@192.168.0.11
ssh sysadmin@192.168.0.12
ssh sysadmin@192.168.0.13
```

For cloud users, you can use the corresponding key pair generated by the cloud provider by uploading it onto the ClusterControl host and specify the physical path when configuring the SSH-related parameters in the ClusterControl UI (deploy a cluster, import nodes, etc). ClusterControl will then use this key to perform tasks that require SSH key-based authentication and store the path via `ssh_identity` variable inside CMON configuration file:

```
ssh_identity=/path/to/keypair/cloud.pem
```

If you use other public-key algorithms (ClusterControl defaults to RSA), make sure the public key generated on the ClusterControl node is copied and allowed on all managed nodes under `~/.ssh/`

`autohorized_keys`. You can use `ssh-copy-id` command (as shown in the examples above), or simply copy the public key to all managed nodes manually.

#### Sudo password

Sudo with or without a password is possible. If undefined, ClusterControl will escalate to sudoer without a password. When deploying a new or importing an existing cluster into ClusterControl, the user will be asked to specify the sudo password in the deployment dialog. The specified sudo password is then stored inside the CMON configuration file under `sudo` variable:

```
sudo="echo 'thesudopassword' | sudo -S 2>/dev/null"
```

**Warning**

Attention

Having `2>/dev/null` in the sudo command is compulsory to strip out stderr from the response.

### 1.5.7 Timezone

ClusterControl requires all servers' time to be synchronized and to run within the same time zone. Verify this by using the following command:

```
$ date
Mon Sep 17 22:59:24 UTC 2013
```

To change time zone, e.g from UTC to Pacific time:

```
rm /etc/localtime
ln -sf /usr/share/zoneinfo/US/Pacific localtime
```

UTC is however recommended. Configure NTP client for each host with a working time server to avoid time drifting between hosts which could cause inaccurate reporting or incorrect graphs plotting. To immediately sync a server's time with a time server, use the following command:

```
ntpdate -u time.apple.com     # or any other NTP server close to the server
```

Frequently asked questions related to ClusterControl and Severalnines.

## 1.6 General

### 1.6.1 Is ClusterControl free?

ClusterControl has a Community Edition that is free forever, available features include deployment and monitoring of clusters/servers. See the ClusterControl product page or *Key Features*.

After installation, you will get a 30-day free trial license (no credit card or signing required) with all enterprise features enabled. After the trial license expires, it will default back to the Community edition.

### 1.6.2 Is there a live demo site where I can try out ClusterControl?

There is no live demo site available at the moment, although we have some webinars that show the product in detail. We recommend users to install ClusterControl on a test server/VM and give it a try.

You can also request a demo by filling up our Request Demo form so we can prepare a demo for you – usually, we can do this within 24 to 48 hours after the initial inquiry.

### 1.6.3 Can I purchase the license for the first year only?

Subscriptions are annual and include software licenses, all updates, and technical support. At the end of the subscription period, you can choose to renew the subscription or continue with the Community Edition. When the license expires, ClusterControl defaults to the Community Edition.

### 1.6.4 My 30-day demo license has expired but I haven't had enough time to test ClusterControl. Can I request for an extension?

Of course. Please use our contact us page or email directly to sales[at]severalnines.com.

### 1.6.5 I would like to subscribe as a paid user but have a limited budget. What should I do?

We'll be happy to understand your specific requirements. Please contact us at sales[at]severalnines.com.

### 1.6.6 Can I resell your product?

Yes, we have a reseller program. Please email us at sales[at]severalnines.com.

### 1.6.7 Who uses ClusterControl?

We have over 100 paying customers, including the likes of Cisco, BT, CNRS, Ping Identity, Technicolor, and Eurovision. A list of references and case studies can be found on the Case Studies page. See also *Target users*.

## 1.7 Technical

### 1.7.1 Do I need to be a DBA to use ClusterControl?

No. We build ClusterControl to be used by DBAs as well as SysAdmins and DevOps to easily manage their databases.

### 1.7.2 Does ClusterControl run on Windows?

No. ClusterControl only runs on RHEL-based and Debian-based with x86-64 architecture.

### 1.7.3 How do I upgrade ClusterControl?

Follow the upgrade steps as described in our offical documentation page. See *Upgrading ClusterControl*.

### 1.7.4 I'm using ClusterControl community version. Which support channel should I use?

You can open a thread in our public forum. In case you have to share files or logs that might contain sensitive information, please submit a support request.

### 1.7.5 Can I install ClusterControl on RHEL-based system but my database cluster is running on Debian?

Yes. Starting from 1.2.11, ClusterControl does not depend on the DB nodes' operating system any longer.

### 1.7.6 Can I use ClusterControl in an offline environment (without Internet)?

Yes. However some features might not be working well, as explained in *Offline Installation* page. Also, see this tutorial, *Deploy Database Cluster in Offline Environment*.

### 1.7.7 Can I have a mix of different SSH users per cluster?

It's possible but not recommended since it will make the deployment more complex. If you choose root as the SSH user, you should use root user for all nodes across the cluster.

### 1.7.8 Can I use a non-privileged SSH user for ClusterControl?

Unfortunately, no. ClusterControl requires super-privileged SSH user to perform tasks like deployment, administration and management of the database nodes. See *Operating system user*.

### 1.7.9 What is difference of MySQL Galera Cluster, Percona XtraDB Cluster and MariaDB Galera Cluster?

The Galera technology is developed by Codership Oy and is available as a patch for standard MySQL and InnoDB. Percona and MariaDB leverage the Galera library in Percona XtraDB Cluster (PXC) and MariaDB Galera Cluster respectively.

Since they all leverage the same Galera library, replication performance should be fairly similar. The Codership build usually has the latest version of Galera, although that could change in the future. See Galera Cluster Comparison – Codership vs Percona vs MariaDB.

### 1.7.10 I found a bug. Where should I report it?

Please report the bug by submitting a support request. For Enterprise customers, bug fixes are provided soonest possible with custom builds.

### 1.7.11 My cluster is down. What should I do?

Please submit a support request at our support site. If you are a paying customer, please do this with your company email address so the ticket can be assigned to the appropriate support SLA. Please also include an *Error Report*.

### 1.7.12 How can I submit a feature request?

You can either use our Feature Request page or submit a ticket at our support site.

## 1.8 Quickstart

Follow these steps to quickly get ClusterControl up and running in your infrastructure.

### 1.8.1 Step 1: Install ClusterControl

On the ClusterControl host, as root or sudo user, run:

```
wget https://severalnines.com/downloads/cmon/install-cc
chmod +x install-cc
sudo ./install-cc
```

**Info**

All new installations come with a 30-day fully functional Enterprise edition trial license (online installation only). Once expired, ClusterControl will default to Community edition.

**Example**

On Ubuntu 22.04 LTS:

```
    $ wget https://severalnines.com/downloads/cmon/install-cc
    Resolving severalnines.com (severalnines.com)... 104.21.94.86,␣
↪172.67.221.146, 2606:4700:3034::6815:5e56, ...
    Connecting to severalnines.com␣
↪(severalnines.com)|104.21.94.86|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: unspecified [text/plain]
    Saving to: 'install-cc'
    install-cc                                  [ <=>                   ␣
↪                                                  ]  79.83K  --.-KB/s    ␣
↪in 0.07s
    2024-09-17 07:47:12 (1.05 MB/s) - 'install-cc' saved [81747]

    $ chmod +x install-cc

    $ sudo ./install-cc
    2024-09-17 07:47:17 UTC -- Installing lsb-release ...
    Reading package lists...
    Building dependency tree...
    Reading state information...
    lsb-release is already the newest version (11.1.0ubuntu4).
    lsb-release set to manually installed.
    0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
    !!
    2024-09-17 07:47:29 UTC -- Only RHEL/RockyLinux/AlmaLinux 8|9, Debian␣
↪11|12, Ubuntu 20.04|22.04|24.04 LTS versions are supported
    2024-09-17 07:47:29 UTC -- System RAM is > 1.5G
    2024-09-17 07:47:29 UTC -- Setting MySQL innodb_buffer_pool_size to␣
↪50% of system RAM
    2024-09-17 07:47:29 UTC -- MySQL innodb_buffer_pool_size set to 1955M
```

```
    2024-09-17 07:47:29 UTC -- Severalnines would like your help␣
↪improving our installation process.
    2024-09-17 07:47:29 UTC -- Information such as OS, memory and install␣
↪success helps us improve how we onboard our users.
    2024-09-17 07:47:29 UTC -- None of the collected information␣
↪identifies you personally.
    2024-09-17 07:47:29 UTC -- !!
    => Would you like to help us by sending diagnostics data for the␣
↪installation? (Y/n): Y

    This script will add Severalnines repository server for deb and rpm␣
↪packages and
    install the ClusterControl Web Applicaiton and Controller.
    An Apache and MySQL server will also be installed. An existing MySQL␣
↪Server on this host can be used.

    => Enter a MySQL root password to use: 2024-09-17 08:31:53 UTC --
    => Enter the MySQL root password again:
    => Enter a CMON MySQL user password to use: 2024-09-17 08:31:58 UTC --
    => Enter the CMON user password again:

    ...
    ...

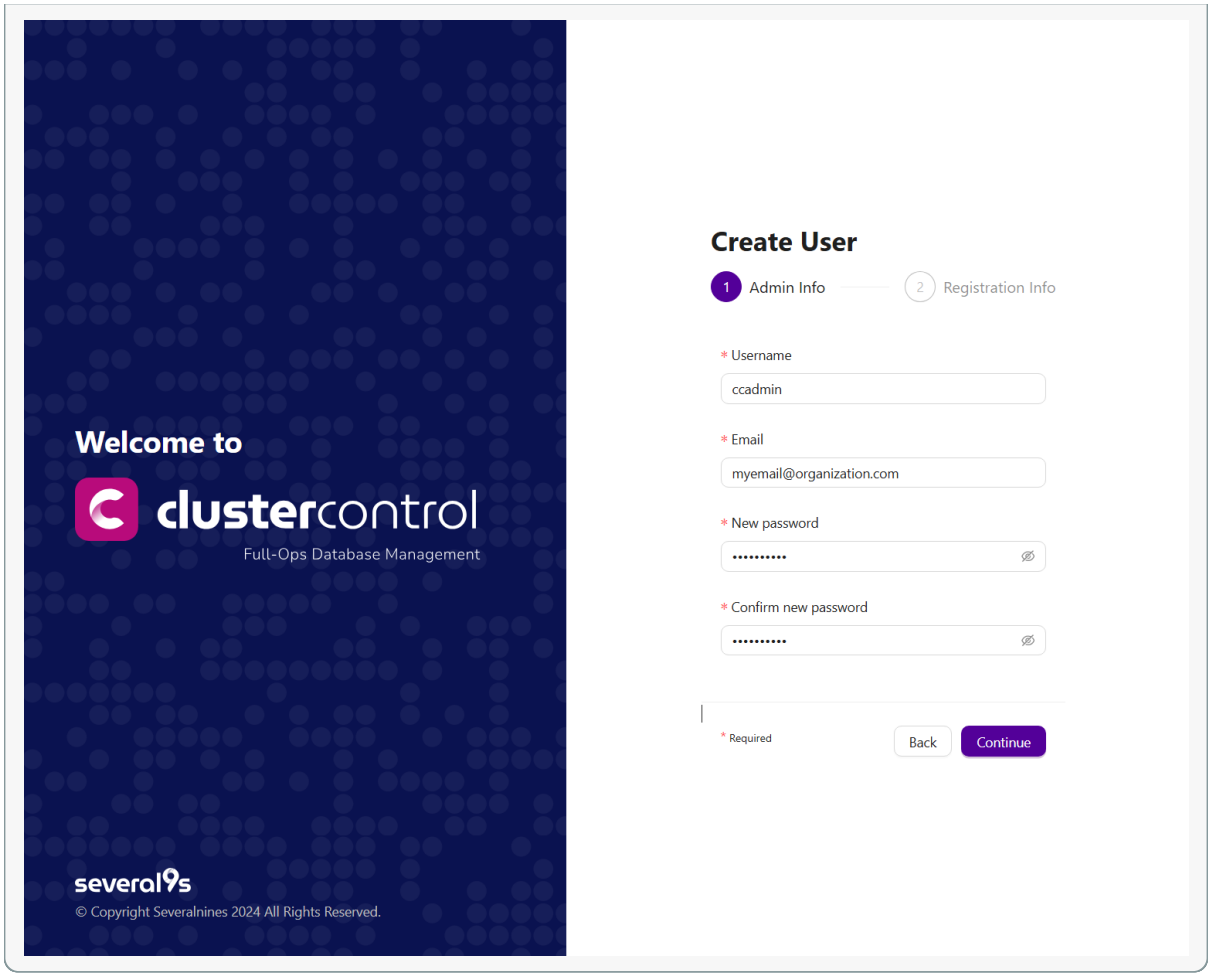
    2024-09-17 08:37:12 UTC -- ClusterControl installation completed!
    Open your web browser to https://178.62.210.68 and create a default␣
↪Admin User.
    If you want to uninstall ClusterControl then please follow the␣
↪instructions here,␣
↪https://severalnines.com/docs/administration.html#uninstall
```

For more information on the installation methods, see *Installation*.

### 1.8.2 Step 2: Create the first admin user

Open `https://<ClusterControl_host>/` in your browser to create the first admin user.

**Example**




### 1.8.3 Step 3: Set up SSH key-based authentication

Set up passwordless SSH (key-based authentication) to all target nodes. Run the following commands on the ClusterControl server:

```
# as root or sudo user
ssh-keygen -t rsa                                    # press Enter on␣
↪all prompts
ssh-copy-id -i ~/.ssh/id_rsa {target_node_IP_address}    # repeat this for␣
↪all target nodes
```

???+ Note

```
You could also add the generated public key, `~/.ssh/id_rsa.pub` to all␣
↪target nodes' `~/.ssh/authorized_keys` file manually.
```

#### As root

```
$ whoami
root

$ ssh-keygen -t rsa
```

(continues on next page)

(continued from previous page)

```
$ ssh-copy-id -i /root/.ssh/id_rsa root@192.168.17.101    # db1
$ ssh-copy-id -i /root/.ssh/id_rsa root@192.168.17.102    # db2
$ ssh-copy-id -i /root/.ssh/id_rsa root@192.168.17.103    # db3
```

#### As sudoer (ubuntu)

```
$ whoami
ubuntu

$ ssh-keygen -t rsa
$ ssh-copy-id -i /home/ubuntu/.ssh/id_rsa ubuntu@192.168.17.101    # db1
$ ssh-copy-id -i /home/ubuntu/.ssh/id_rsa ubuntu@192.168.17.102    # db2
$ ssh-copy-id -i /home/ubuntu/.ssh/id_rsa ubuntu@192.168.17.103    # db3
```

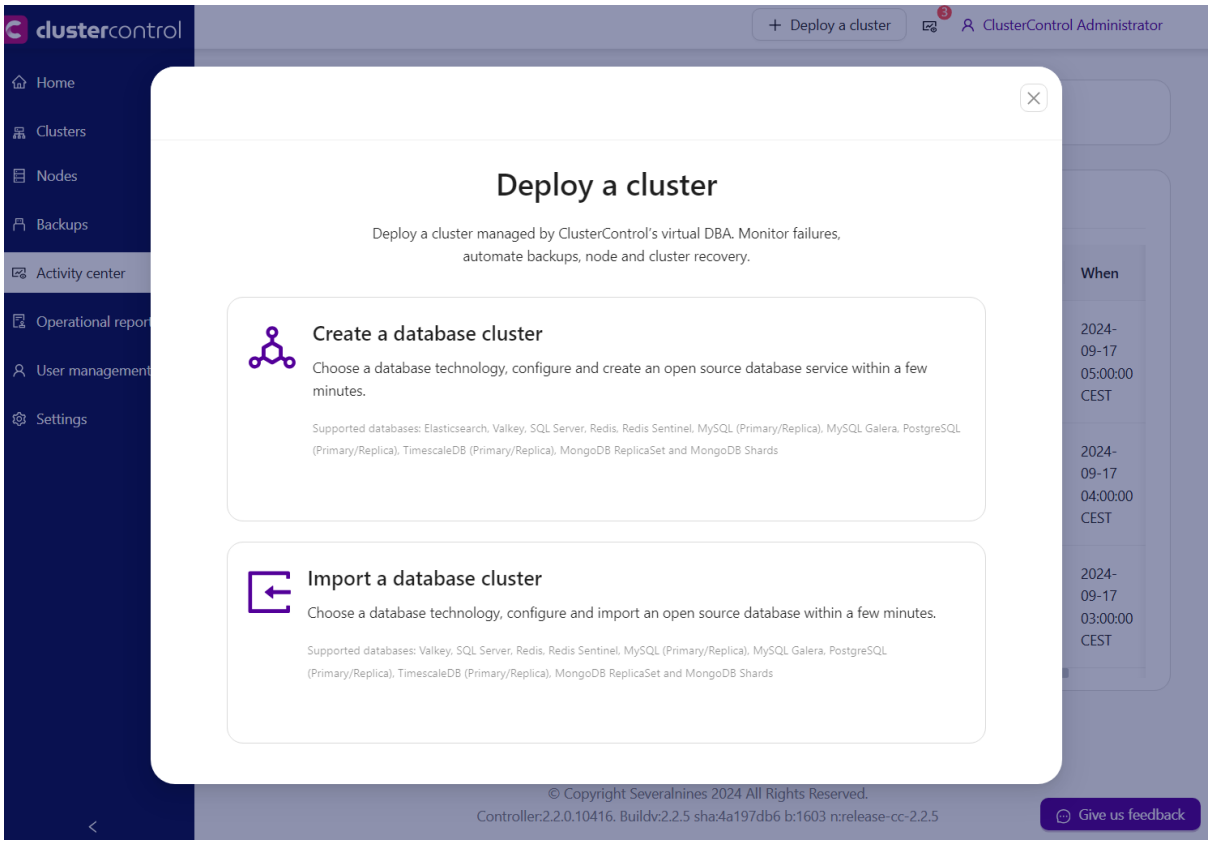
???+ Note

```
For cloud instances, you can use the corresponding private key pair␣
↪generated by the cloud provider and upload it onto the ClusterControl␣
↪host and specify the physical path when configuring the SSH-related␣
↪parameters in the ClusterControl GUI (deploy a cluster, import nodes,␣
↪etc).
```

### 1.8.4 Step 4: Deploy or import database cluster/server

You can deploy a new database cluster or import an existing database cluster into ClusterControl.

### ClusterControl GUI




### ClusterControl CLI

```
# create a 3-node Galera cluster (Percona XtraDB Cluster) v8.0
$ s9s cluster --create \
    --cluster-type=galera \
    --nodes="192.168.17.101;192.168.17.102;192.168.17.103" \
    --vendor=percona \
    --provider-version=8.0 \
    --db-admin-passwd='Str0nGpa$$word' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='Percona XtraDB Cluster 8.0'
```

### ClusterControl RPC API

First, authenticate and store the session cookie:

```
curl -k 'https://<ClusterControl_host>:9501/v2/auth' \
-XPOST -d \
    '{"operation":"authenticateWithPassword",
        "user_name":"admin",
        "password":"Str0nGpa$$word"
    }' \
-c cookies.jar
```

Then submit the deployment job:

```
curl -k 'https://<ClusterControl_host>:9501/v2/jobs' \
    -XPOST -d '{
        "operation": "createJobInstance",
        "job": {
            "class_name": "CmonJobInstance",
            "job_spec": {
                "command": "create_cluster",
                "job_data": {
                    "cluster_name": "Percona XtraDB Cluster 8.0",
                    "cluster_type": "galera",
                    "config_template": "my.cnf.80-pxc",
                    "datadir": "/var/lib/mysql",
                    "db_password": "Str0nGpa55word",
                    "db_user": "root",
                    "disable_firewall": true,
                    "disable_selinux": true,
                    "enable_ssl": true,
                    "generate_token": true,
                    "install_software": true,
                    "port": "3306",
                    "ssh_keyfile": "/root/.ssh/id_rsa",
                    "ssh_port": "22",
                    "ssh_user": "root",
                    "sudo_password": "",
                    "type": "mysql",
                    "vendor": "percona",
                    "version": "8.0",
                    "nodes": [
                        {
                            "hostname": "192.168.17.101",
                            "hostname_data": "192.168.17.101",
                            "hostname_internal": "",
                            "port": "3306"
                        },
                        {
                            "hostname": "192.168.17.102",
                            "hostname_data": "192.168.17.102",
                            "hostname_internal": "",
                            "port": "3306"
                        },
                        {
                            "hostname": "192.168.17.103",
                            "hostname_data": "m192.168.17.103",
                            "hostname_internal": "",
                            "port": "3306"
                        }
                    ],
                    "with_tags": [
                        "galera",
```

(continues on next page)

(continued from previous page)

```
                        "multimaster",
                        "mysql"
                    ]
                }
            }
        }
        }' \
    -b cookies.jar
```

### Terraform Provider for ClusterControl

```
resource "clustercontrol_db_cluster" "my-galera-cluster" {

    db_cluster_create = true
    db_cluster_name = "Percona XtraDB Cluster 5.7"
    db_cluster_type = "galera"
    db_version = "8.0"
    db_vendor = "percona"
    db_admin_user_password = "Str0nGpa$$word"
    db_host {
        hostname = "192.168.17.101"
    }
    db_host {
        hostname = "192.168.17.102"
    }
    db_host {
        hostname = "192.168.17.103"
    }
    ssh_key_file = "/root/.ssh/id_rsa"
    ssh_user = "root"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

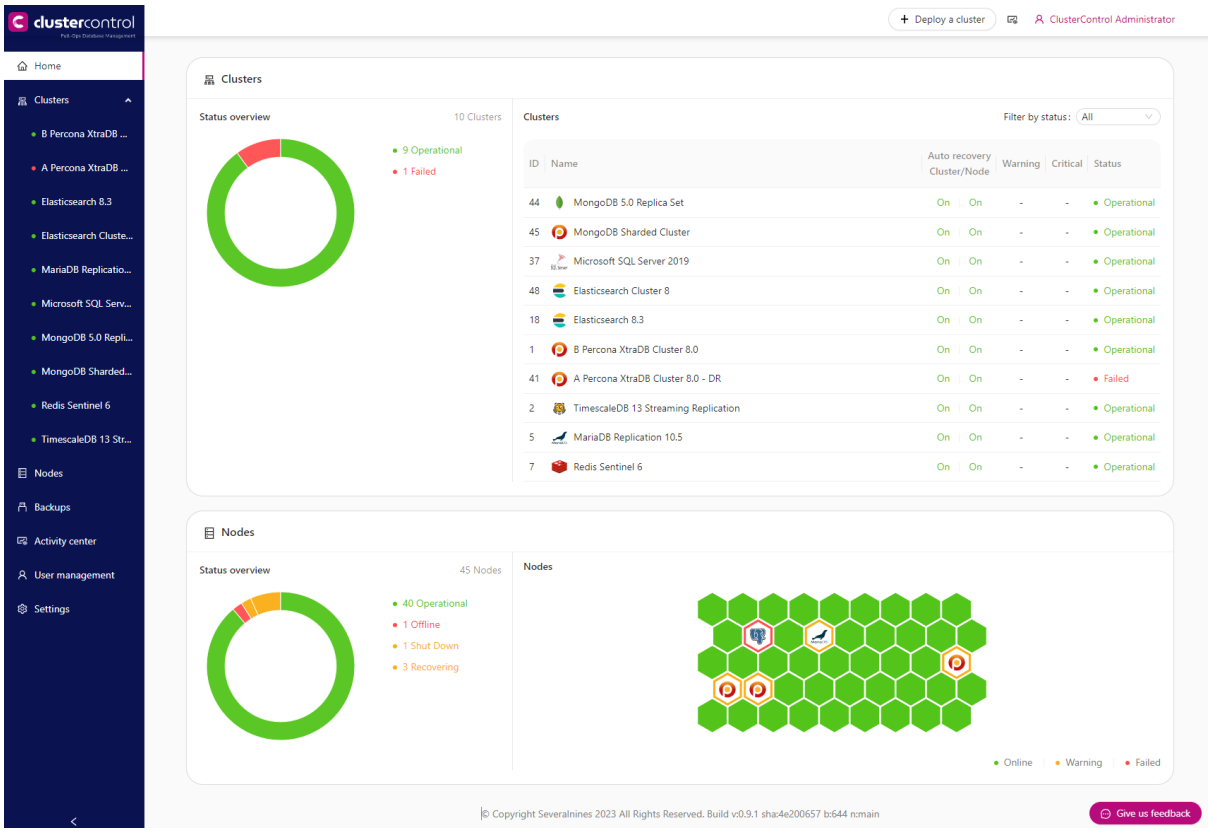
**More info**

- *Getting Started → Tutorials*
- *User Guide → Deployment → Create Database Cluster*
- *User Guide → Deployment → Import Database Cluster*

### 1.8.5 Step 5: Start managing

Use the ClusterControl GUI or CLI to monitor and manage your database cluster/server.

#### ClusterControl GUI




#### ClusterControl CLI

```
# list all database clusters
$ s9s cluster --list --long

# list all managed database nodes
$ s9s node --list --long

# scale out a cluster by adding a new node
$ s9s cluster --add-node \
    --nodes=192.168.17.104 \
    --cluster-id=1 \
    --wait
```

(continues on next page)

(continued from previous page)

```
# perform a cluster rolling restart in 20 minutes
$ s9s cluster --rolling-restart \
    --cluster-id=1 \
    --schedule="$(date -d 'now + 20 min')"
```

### ClusterControl RPC API

List clusters and nodes (using the session cookie obtained in Step 4):

```
curl -k 'https://<ClusterControl_host>:9501/v2/clusters' \
    -XPOST -d '{"operation":"getAllClusterInfo"}' \
    -b cookies.jar
```

Trigger a rolling restart of a cluster:

```
curl -k 'https://<ClusterControl_host>:9501/v2/jobs' \
    -XPOST -d '{
        "cluster_id": 1,
        "job": {
            "class_name": "CmonJobInstance",
            "title": "Rolling Restart",
            "job_spec": {
                "command": "rolling_restart"
            }
        },
        "operation": "createJobInstance"
    }' \
    -b cookies.jar
```

See *Reference Manuals → API/SDK → ClusterControl RPC API* for the full list of available operations, including adding nodes.

### Terraform Provider for ClusterControl

Scale the cluster by declaring an additional `db_host` block and re-applying:

```
resource "clustercontrol_db_cluster" "my-galera-cluster" {
    # ... existing configuration from Step 4 ...
    db_host {
        hostname = "192.168.17.101"
    }
    db_host {
        hostname = "192.168.17.102"
    }
    db_host {
        hostname = "192.168.17.103"
    }
    db_host {
        hostname = "192.168.17.104"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

```
$ terraform apply
```

Terraform manages the cluster's declared topology; operational tasks like rolling restarts are not part of its resource model and should be performed through the GUI, CLI, or RPC API.

**More info**

- *User Guide*
- *Administration Guide*

### 1.8.6 Step 6: Additional resources

- *User Guide*
- *Administration Guide*
- Follow ClusterControl blog
- Contact Support
- Contact Sales

## 1.9 Online Installation

There are several ways to get ClusterControl installed and running in your infrastructure:

- Installer script (install-cc)
- ClusterControl Helm Charts
- Ansible Role
- Puppet Module

### 1.9.1 Installer script (install-cc)

The installer script is the recommended way to install ClusterControl, and it is the easiest way to get ClusterControl up and running on the supported operating system. For example, see *Quickstart*.

The script must be downloaded and executed on the ClusterControl node, which performs all necessary steps to install and configure ClusterControl's packages and dependencies on that particular host. It also supports offline installation with `OFFLINE=1` variable exported, however, you need to have a mirrored repository enabled beforehand. See *Offline Installation* for details. The script assumes that the host can install all dependencies via the operating system's repositories.

On the ClusterControl server, run the following commands:

```
wget https://severalnines.com/downloads/cmon/install-cc
chmod +x install-cc
sudo ./install-cc     # omit sudo if you run as root
```

??? Info

```
The installation script will attempt to automate the following tasks:

- Install and configure a MySQL or MariaDB server.
- Install ClusterControl dependencies via the package manager.
- Install and configure ClusterControl packages via package manager.
- Configure ClusterControl RPC API user and token.
- Configure ClusterControl Controller (CMON) with minimal configuration␣
↪options.
- Enable the ClusterControl-related services on boot and start them up.
```

After the installation completes, open a web browser and go to `https://<ClusterControl_host>/` and create the default admin user by specifying a username (username "admin" is reserved) and password on the welcome page.

### Environment variables

The installer script also understands a number of environment variables if defined. Supported environment variables are:

| Variables | Description |
|---|---|
| `S9S_CMON_PASSWORD` | MySQL cmon user password. |
| `S9S_ROOT_PASSWORD` | MySQL root user password. |
| `S9S_DB_PORT` | MySQL port for cmon to connect. |
| `HOST` | Primary IP address or FQDN of the host. Useful if the host has multiple IP addresses. |
| `OFFLINE` | Special flag to indicate an offline installation. A mirrored repository must be enabled. |
| `INNODB_BUFFER_POOL_SI` | MySQL InnoDB buffer pool size to be configured on the host. The default is 50% of the host's RAM. |
| `CLUSTERCONTROL_BUILD` | ClusterControl builds (other than the controller). Separate each package with space. |
| `CONTROLLER_BUILD` | ClusterControl controller build. |
| `S9S_TOOLS_BUILD` | ClusterControl CLI (a.k.a s9s) build. |

The environment variable can be set through the `export` command or by prefixing the install command as shown in the *Example Use Cases* section.

### Example use cases

If you have multiple network interface cards, you may immediately assign a primary IP address using the `HOST` variable as per the example below (otherwise the script would prompt use to choose one primary interface):

```
HOST=192.168.1.10 ./install-cc
```

By default, the script will allocate 50% of the host's RAM to the InnoDB buffer pool. You may change this by assigning a value in MB for `INNODB_BUFFER_POOL_SIZE` variable as the example below:

```
INNODB_BUFFER_POOL_SIZE=512 ./install-cc
```

If you want to perform a single-liner non-interactive installation, you can assign each mandatory variable with its value beforehand, similar to the example below:

```
S9S_CMON_PASSWORD=cmonP4ss \
S9S_ROOT_PASSWORD=root123 \
S9S_DB_PORT=3306 \
HOST=10.10.10.10 \
./install-cc
```

If you want to install a specific version instead of the latest in the repository, you can use `CLUSTERCONTROL_BUILD`, `CONTROLLER_BUILD` and `S9S_TOOLS_BUILD` environment variables. You can get the available package name and version from the ClusterControl download site.

Examples are as follows:

#### Red Hat/CentOS/AlmaLinux/Rocky Linux

```
CLUSTERCONTROL_BUILD="clustercontrol-1.7.1-5622␣
↪clustercontrol-cloud-1.7.1-163 clustercontrol-clud-1.7.1-163␣
↪clustercontrol-notifications-1.7.1-159 clustercontrol-ssh-1.7.1-70" \
CONTROLLER_BUILD="clustercontrol-controller-1.7.1-2985" \
S9S_TOOLS_BUILD="s9s-tools-1.7-93.1" \
./install-cc
```

#### Debian/Ubuntu

```
CLUSTERCONTROL_BUILD="clustercontrol-1.7.1-5622␣
↪clustercontrol-cloud-1.7.1-163 clustercontrol-clud-1.7.1-163␣
↪clustercontrol-notifications-1.7.1-159 clustercontrol-ssh-1.7.1-70" \
CONTROLLER_BUILD="clustercontrol-controller-1.7.1-2985" \
S9S_TOOLS_BUILD="s9s-tools-1.7.20190117-release1" \
./install-cc
```

### 1.9.2 Helm Chart

This Helm chart is designed to provide everything you need to get ClusterControl running in a vanila Kubernetes cluster. This includes dependencies like:

- Ingress-Nginx Controller
- MySQL Operator
- VictoriaMetrics Operator

If you do not wish to install any of those, see *Helm chart dependencies*.

### Installation using Helm chart

1. Add a helm chart repository with the following commands:

```
helm repo add s9s https://severalnines.github.io/helm-charts/
helm repo update
```

2. Create a namespace for ClusterControl. It's also required for MySQL Operator to run in a custom namespace (not default):

```
kubectl create ns clustercontrol
kubectl config set-context --current --namespace=clustercontrol
```

3. Install ClusterControl using Helm:

```
helm install clustercontrol s9s/clustercontrol --devel
```

The installation will generate an example SSH key for you to use to access your database nodes. However, you can use your own SSH key as described below.

### Providing your own SSH keys for ClusterControl to use

The ClusterControl Helm chart provides an example SSH key for you to use, however, you should provide your SSH keys for ClusterControl to use and connect to your target machines. These should already be configured on the target server's `authorized_keys`.

1. Create k8s secrets with your SSH keys:

```
kubectl create secret generic my-ssh-keys --from-file=key1=/
↪path/to/my/.ssh/id_rsa
```

`key1` is the filename of your SSH key in ClusterControl - this will be created under `/root/.ssh-keys-user`

> **Note**
>
> Note You can use multiple `--from-file` - be sure to provide unique key names - `key1`, `key2`, `key3`

2. Install or upgrade ClusterControl by providing `cmon.sshKeysSecretName` value with our secret name created above:

```
helm upgrade --install clustercontrol s9s/clustercontrol --set␣
↪cmon.sshKeysSecretName=my-ssh-keys --devel
```

### Custom configuration via values.yaml

To create your own override file, export it from Helm using the show command:

```
helm show values s9s/clustercontrol > values.yaml
```

Look at the generated `values.yaml` and customize your configuration here. To install or upgrade using your custom `values.yaml`, include the override file with the `-f` flag:

```
helm install clustercontrol s9s/clustercontrol -f values.yaml --devel
```

**Note**

Note - ClusterControl CMON RPC API is accessible within the cluster via `cmon-master:9501`. - ClusterControl GUI is accessible within the cluster via `cmon-master:3000`. - We highly recommend ingress as ClusterControl GUI requires CMON API to be exposed and available externally.

### Helm chart dependencies

### Oracle MySQL Operator or NGINX ingress controller

If you already have Oracle MySQL Operator or NGINX ingress controller installed, you can set the following flags:

```
helm install clustercontrol s9s/clustercontrol --debug \
  --set fqdn=clustercontrol.example.com \
  --set installMysqlOperator=false \
  --set ingressController.enabled=false --devel
```

This Helm chart has dependencies that make ClusterControl easier to install. None of these is necessary if you provide your equivalent or you already have it installed:

- oracle-mysql-operator

```
- Oracle MySQL operator is required for running the MySQL database␣
↪within the Kubernetes cluster.

- You can disable this by setting: `installMysqlOperator: false`
```

- oracle-mysql-innodbcluster

```
- A MySQL InnoDB Cluster is required for ClusterControl.
- You must provide a different MySQL/MariaDB for ClusterControl to␣
↪use. Please refer to the [official Helm chart documentation for␣
↪MySQL InnoDB␣
↪Cluster](https://github.com/mysql/mysql-operator/blob/trunk/helm/mysql-innodbcluste
- You can disable this by setting: `createDatabases: false`.
```

- nginx-ingress-controller

```
- An ingress controller to access ClusterControl.
- For more information, please refer to the official Helm chart␣
↪documentation for the [NGINX-Ingress␣
↪Controller](https://github.com/kubernetes/ingress-nginx/tree/main/charts/ingress-n
- If you already have an ingress controller installed or wish to use a␣
↪different one, you can disable this by:

  ```yaml
  ingressController:
```

(continues on next page)
```

(continued from previous page)

```
  enabled: false
  ```
```

### VictoriaMetrics or other Prometheus-compatible monitoring

If you wish to use your VictoriaMetrics or other Prometheus-compatible monitoring systems, please refer to victoria-metrics-single parameters. These defaults provide a minimal need for ClusterControl metrics and dashboards to work. Feel free to adjust as needed, however, keep in mind the required labels annotations, and service discovery. If you already have your own VictoriaMetrics or Prometheus cluster and don't want to install this, you can disable it by setting the following inside `values.yaml`:

```
prometheusHostname: my-prometheus-server
monitoring:
  enabled: false
```

## 1.9.3 Ansible Role

If you are automating your infrastructure using Ansible, we have created a role for this purpose and it is available at Ansible Galaxy. This role also supports deploy a new cluster and import an existing cluster into ClusterControl automatically, as shown under *Example Playbooks*.

**See also**

ClusterControl Ansible Github page.

Getting the role is as easy as:

```
ansible-galaxy install severalnines.clustercontrol
```

### Usage

1. Get the ClusterControl Ansible role from Ansible Galaxy or Github.

### Ansible Galaxy

```
ansible-galaxy install severalnines.clustercontrol
```

### Github

```
git clone https://github.com/severalnines/ansible-clustercontrol
cp -rf ansible-clustercontrol /etc/ansible/roles/severalnines.clustercontrol
```

2. Create a playbook. See *Example playbooks* section.
3. Run the playbook.

```
ansible-playbook cc.playbook
```
```

4. Once ClusterControl is installed, go to `https://<ClusterControl_host>/` and create the default admin user/password.
5. On ClusterControl node, set up SSH key-based authentication to all target DB nodes. For example, if ClusterControl node is 192.168.0.10 and DB nodes are 192.168.0.11, 192.168.0.12 and 192.168.0.13:

```
ssh-copy-id 192.168.0.11    # DB1
ssh-copy-id 192.168.0.12    # DB2
ssh-copy-id 192.168.0.13    # DB3
```

6. Start to deploy a new database cluster or add an existing cluster.

### Example playbooks

The simplest playbook would be:

```
- hosts: clustercontrol-server
  roles:
    - { role: severalnines.clustercontrol }
```

If you would like to specify custom configuration values as explained above, create a file called `vars/main.yml` and include it inside the playbook:

```
- hosts: 192.168.10.15
  vars:
    - vars/main.yml
  roles:
    - { role: severalnines.clustercontrol, tags: controller }
```

Inside `vars/main.yml`:

```
controller: true
mysql_root_username: admin
mysql_root_password: super-user-password
cmon_mysql_password: super-cmon-password
cmon_mysql_port: 3307
```

If you are running as another user, ensure the user has the ability to escalate as a superuser via sudo. Example playbook for Ubuntu 20.04 with sudo password enabled:

```
- hosts: ubuntu@192.168.10.100
  become: yes
  become_user: root
  roles:
    - { role: severalnines.clustercontrol, tags: controller }
```

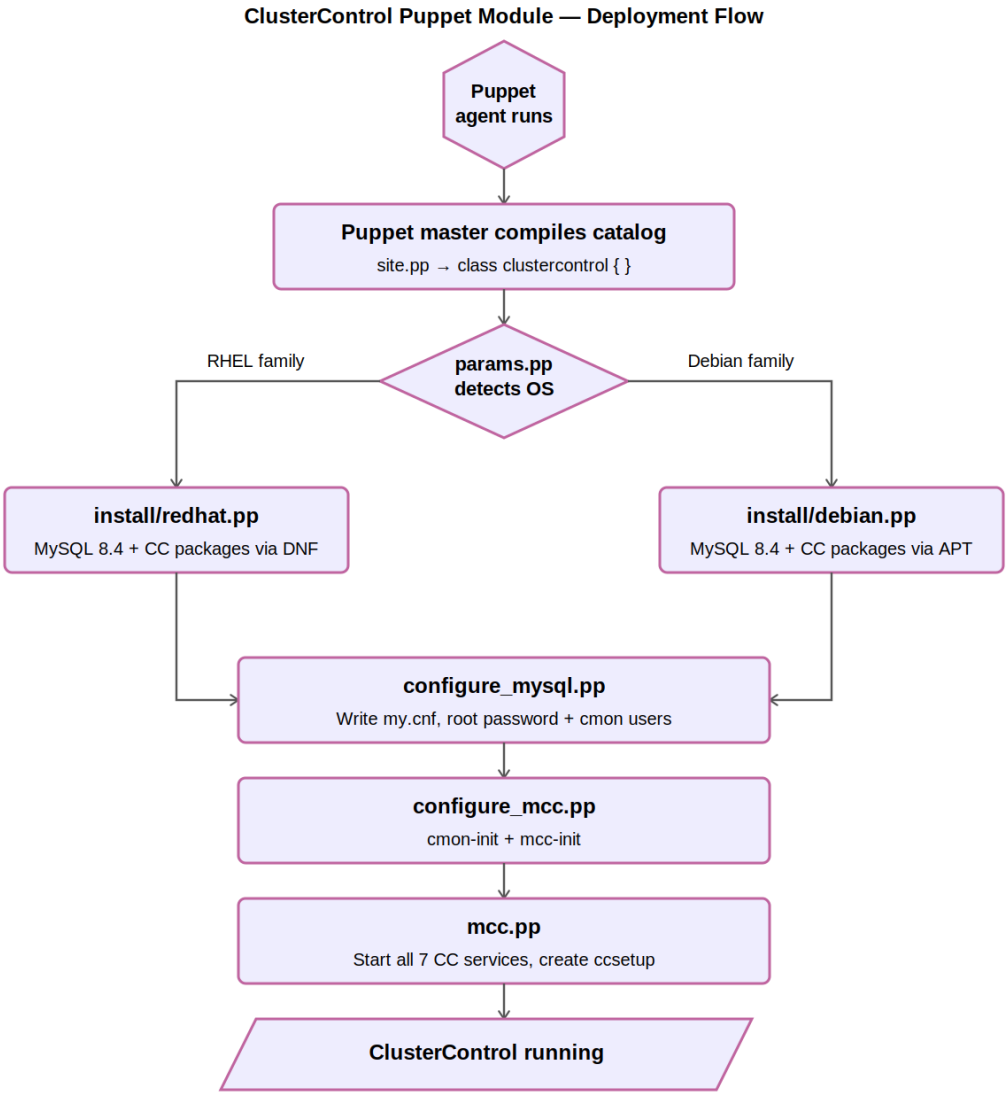
Then, execute the command with the `--ask-become-pass` flag, for example:

```
$ ansible-playbook cc.playbook --ask-become-pass
```

### 1.9.4 Puppet module

This module installs ClusterControl for your new database node/cluster deployment or on top of your existing database node/cluster. The Puppet module for ClusterControl automates the following actions based on the scope of this module:

- Setup ClusterControl required repositories
- Install MariaDB and configure it for ClusterControl
- Create the cmon database schema and grant the cmon database user
- Install all ClusterControl packages (`clustercontrol-controller`, `clustercontrol-mcc`, `clustercontrol-proxy`, `s9s-tools`, and related components)
- Initialize the CMON controller via `cmon --init`
- Initialize the ClusterControl management component via `ccmgradm init`
- Start and enable all ClusterControl services (`cmon`, `cmon-proxy`, `cmon-ssh`, `cmon-events`, `cmon-cloud`, `kuber-proxy`)
- Create the `ccsetup` bootstrap user required for first-time GUI registration




## Requirements

Make sure you meet following criteria prior to the deployment:

- ClusterControl node must run on a clean dedicated host with internet connection.
- Puppet 7 or Puppet 8.
- If you are running as a non-root user, make sure the user is able to escalate to root with the `sudo` command.

## Installation using Puppet

Our ClusterControl module for Puppet is available either on Puppet Forge or from the Severalnines Puppet repository by cloning or downloading it as a zip. Place it under the puppetlabs `modulepath` directory and ensure the module directory is named `clustercontrol`. For example:

```
/etc/puppetlabs/code/environments/production/modules/clustercontrol
```

1. Install the module from Puppet Forge:

   ```
   puppet module install severalnines-clustercontrol
   ```

   Or clone from GitHub:

   ```
   cd /etc/puppetlabs/code/environments/production/modules
   git clone https://github.com/severalnines/puppet clustercontrol
   ```

2. Create a node definition. The module requires only two parameters. All other parameters are optional with sensible defaults.

   ```
   node 'clustercontrol.puppet.local' {
     class { 'clustercontrol':
       mysql_root_password => 'StrongRootPassword',
       cmon_mysql_password => 'StrongCmonPassword',
     }
   }
   ```

   To pin a specific ClusterControl version instead of always installing the latest:

   ```
   node 'clustercontrol.example.com' {
     class { 'clustercontrol':
       mysql_root_password    => 'StrongRootPassword',
       cmon_mysql_password    => 'StrongCmonPassword',
       clustercontrol_version => '2.4.0',
     }
   }
   ```

3. Run the Puppet agent on the ClusterControl node:

   ```
   puppet agent -t
   ```

4. Open the ClusterControl GUI by navigating to `https://<ClusterControl_host>/` in your browser. On first access, log in using the `ccsetup` bootstrap user with password `admin`. You will be prompted to create your real admin account.

5. Configure SSH access to your database nodes. After ClusterControl is installed, set up SSH key-based authentication from the ClusterControl node to all target database nodes. For example:

   ```
   ssh-copy-id -i /root/.ssh/id_rsa root@192.168.0.11    # DB node 1
   ssh-copy-id -i /root/.ssh/id_rsa root@192.168.0.12    # DB node 2
   ssh-copy-id -i /root/.ssh/id_rsa root@192.168.0.13    # DB node 3
   ```

6. Start deploying or importing database clusters from the ClusterControl GUI.

### Parameters

| Parameter | Required | Default | Description |
|---|---|---|---|
| `mysql_root_pas` | Yes | — | MySQL root user password. Used during installation only. |
| `cmon_mysql_pas` | Yes | — | Password for the cmon MariaDB user. Used by CMON at runtime. |
| `clustercontrol_version` | No | latest | ClusterControl package version to install. Set to a specific version (e.g. `2.4.0`) to pin and prevent automatic upgrades. |
| `disable_firewall` | No | true | Disables the OS firewall on the ClusterControl node during installation. |
| `disable_selinux` | No | true | Disables SELinux on RHEL-family systems during installation. |

### Idempotency

This module is fully idempotent. Re-running `puppet agent -t` on an already-installed node will not overwrite or re-initialize any existing configuration. The module uses the following guards:

- MySQL root password: only set if root has no password yet
- cmon database schema: `CREATE SCHEMA IF NOT EXISTS`
- cmon user grants: only granted if the user cannot already connect
- `cmon --init` and `ccmgradm init`: guarded by state marker files under `/var/lib/cmon/`

**More info**

- ClusterControl Puppet Module Github page.
- For more examples of deployments using Puppet, see Puppet Module for ClusterControl - Adding Management and Monitoring to your Existing Database Clusters.

The `install-cc` official installation script has been updated with several helper functions to simplify the installation of ClusterControl in air-gapped (offline) environments. These functions enable the creation of `.deb` or `.rpm` repositories that can be imported into the target system.

The process involves first running the script on a server or virtual machine that matches the Linux distribution of the air-gapped server and has internet access. The script performs the following tasks:

- Downloads the necessary ClusterControl packages along with their dependencies, such as the MySQL server or `gnuplot` for the CMON Controller.
- Builds a local file-based `.deb` or `.rpm` package repository based on the Linux distribution.
- Generates a tarball containing the repository directory and the repository metadata file, which can then be transferred and imported onto another server.

ClusterControl provides two methods for installing ClusterControl in an air-gapped environment without requiring internet access during the installation:

- *Offline Setup*: Install ClusterControl manually by preparing and transferring the required packages and dependencies to the air-gapped environment.

- *Installer Setup*: Install ClusterControl using a standalone offline installer file that contains the required installation components.

Take note that the following ClusterControl features will not work without an Internet connection:

- ***Backup → Create/Schedule Backup → Upload to Cloud*** – requires a connection to cloud providers.
- ***Integrations → Cloud Providers*** – requires a connection to cloud providers.
- ***Manage → Load Balancer*** – requires a connection to EPEL, ProxySQL, HAProxy, MariaDB repository.
- ***Manage → Upgrades*** – requires a connection to the provider's repository.
- ***Deploy Database Cluster*** – requires a connection to the database provider's repository.

## 1.10 Offline Setup

### 1.10.1 Step 1: Build and export the repository

The offline repository package is configured on a temporary host that has internet access. The installer script is able to build the local repository for offline usage on RHEL/Rocky Linux/AlmaLinux 8/9, Debian 11/12 and Ubuntu 20.04/22.04/24.04.

Download the installer script on the temporary host that has an Internet access:

```
wget https://severalnines.com/downloads/cmon/install-cc
chmod +x install-cc
```

Build the local repository for ClusterControl by using the `--airgap-repo` flag:

```
./install-cc --airgap-repo
```

**Example**

The following is an example output of the temporary server (the staging host that has an Internet connection) on Ubuntu 24.04:

```
[root@staging]:/home/vagrant# ./install-cc --airgap-repo
This script will configure the Severalnines repository server for both␣
↪deb and rpm packages, and install the ClusterControl Web Application␣
↪and Controller.
It will install a new MySQL server or utilize an existing MySQL server on␣
↪the host.
2025-05-30 05:52:23 UTC -- Only RHEL/RockyLinux/AlmaLinux 8|9, Debian␣
↪11|12, Ubuntu 20.04|22.04|24.04 LTS versions are supported
2025-05-30 05:52:23 UTC -- Waiting for locks ...
2025-05-30 05:52:23 UTC -- Update distro repos ...
141 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists...
Building dependency tree...
Reading state information...
lsb-release is already the newest version (12.0-2).
lsb-release set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 141 not upgraded.
```

```
2025-05-30 05:52:56 UTC -- DOWNLOAD_DIR=/home/vagrant/cc_packages
2025-05-30 05:52:56 UTC -- REPO_DIR=/usr/local/repo/cc_repo
2025-05-30 05:52:56 UTC -- EXPORT_FILE=cc_repo.tar.gz
2025-05-30 05:52:56 UTC -- IMPORT_REPO_DIR=/usr/local/repo/cc_repo
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wget is already the newest version (1.21.4-1ubuntu4.1).
The following additional packages will be installed:
  dirmngr gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpgconf␣
↪gpgsm gpgv keyboxd
Suggested packages:
  pinentry-gnome3 tor parcimonie xloadimage gpg-wks-server scdaemon
The following packages will be upgraded:
  dirmngr gnupg gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client␣
↪gpgconf gpgsm gpgv keyboxd
11 upgraded, 0 newly installed, 0 to remove and 130 not upgraded.
Need to get 2291 kB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64␣
↪gpg-wks-client amd64 2.4.4-2ubuntu17.2 [70.9 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64␣
↪dirmngr amd64 2.4.4-2ubuntu17.2 [323 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64␣
↪gnupg-utils amd64 2.4.4-2ubuntu17.2 [109 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64 gpgsm␣
↪amd64 2.4.4-2ubuntu17.2 [232 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64␣
↪gpg-agent amd64 2.4.4-2ubuntu17.2 [227 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64 gpg␣
↪amd64 2.4.4-2ubuntu17.2 [565 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64␣
↪gpgconf amd64 2.4.4-2ubuntu17.2 [103 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64 gnupg␣
↪all 2.4.4-2ubuntu17.2 [359 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64␣
↪keyboxd amd64 2.4.4-2ubuntu17.2 [78.3 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64 gpgv␣
↪amd64 2.4.4-2ubuntu17.2 [158 kB]
Get:11 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64␣
↪gnupg-l10n all 2.4.4-2ubuntu17.2 [66.1 kB]
Fetched 2291 kB in 3s (674 kB/s)
(Reading database ... 50146 files and directories currently installed.)
Preparing to unpack .../0-gpg-wks-client_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking gpg-wks-client (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../1-dirmngr_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking dirmngr (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../2-gnupg-utils_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking gnupg-utils (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../3-gpgsm_2.4.4-2ubuntu17.2_amd64.deb ...
```

```
Unpacking gpgsm (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../4-gpg-agent_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking gpg-agent (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../5-gpg_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking gpg (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../6-gpgconf_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking gpgconf (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../7-gnupg_2.4.4-2ubuntu17.2_all.deb ...
Unpacking gnupg (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../8-keyboxd_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking keyboxd (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Preparing to unpack .../9-gpgv_2.4.4-2ubuntu17.2_amd64.deb ...
Unpacking gpgv (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Setting up gpgv (2.4.4-2ubuntu17.2) ...
(Reading database ... 50146 files and directories currently installed.)
Preparing to unpack .../gnupg-l10n_2.4.4-2ubuntu17.2_all.deb ...
Unpacking gnupg-l10n (2.4.4-2ubuntu17.2) over (2.4.4-2ubuntu17) ...
Setting up gnupg-l10n (2.4.4-2ubuntu17.2) ...
Setting up gpgconf (2.4.4-2ubuntu17.2) ...
Setting up gpg (2.4.4-2ubuntu17.2) ...
Setting up gnupg-utils (2.4.4-2ubuntu17.2) ...
Setting up gpg-agent (2.4.4-2ubuntu17.2) ...
Setting up gpgsm (2.4.4-2ubuntu17.2) ...
Setting up dirmngr (2.4.4-2ubuntu17.2) ...
Setting up keyboxd (2.4.4-2ubuntu17.2) ...
Setting up gnupg (2.4.4-2ubuntu17.2) ...
Setting up gpg-wks-client (2.4.4-2ubuntu17.2) ...
Processing triggers for install-info (7.1-3build2) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d␣
 ↪instead (see apt-key(8)).
OK
deb http://repo.severalnines.com/s9s-tools/noble/ ./
--2025-05-30 05:53:55-- ␣
 ↪http://repo.severalnines.com/severalnines-repos.asc
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
 ↪(repo.severalnines.com)|54.247.53.237|:80... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 963 [application/octet-stream]
Saving to: 'STDOUT'

-                           ␣
 ↪100%[=================================================>]     963  --.-
 ↪KB/s    in 0s

2025-05-30 05:53:55 (53.1 MB/s) - written to stdout [963/963]

Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d␣
 ↪instead (see apt-key(8)).
OK
deb [arch=amd64] http://repo.severalnines.com/deb ubuntu main
2025-05-30 05:53:59 UTC -- Added /etc/apt/sources.list.d/s9s-repo.list
2025-05-30 05:53:59 UTC -- Updating repo ...
Get:1 http://repo.severalnines.com/deb ubuntu InRelease [3973 B]
Get:2 http://repo.severalnines.com/s9s-tools/noble ./ InRelease [1876 B]
Hit:3 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu noble InRelease
Hit:5 http://us.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:6 http://us.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:7 http://repo.severalnines.com/deb ubuntu/main amd64 Packages [71.9␣
 ↪kB]
Get:8 http://repo.severalnines.com/s9s-tools/noble ./ Packages [785 B]
Fetched 78.5 kB in 9s (8410 B/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
130 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: http://repo.severalnines.com/deb/dists/ubuntu/InRelease: Key is stored␣
 ↪in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the␣
 ↪DEPRECATION section in apt-key(8) for details.
W: http://repo.severalnines.com/s9s-tools/noble/./InRelease: Key is␣
 ↪stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the␣
 ↪DEPRECATION section in apt-key(8) for details.
Hit:1 http://us.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://repo.severalnines.com/deb ubuntu InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://repo.severalnines.com/s9s-tools/noble ./ InRelease
Hit:5 http://us.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:6 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
130 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: http://repo.severalnines.com/deb/dists/ubuntu/InRelease: Key is stored␣
 ↪in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the␣
 ↪DEPRECATION section in apt-key(8) for details.
W: http://repo.severalnines.com/s9s-tools/noble/./InRelease: Key is␣
```

```
↪stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the␣
↪DEPRECATION section in apt-key(8) for details.
2025-05-30 05:54:34 UTC -- Downloading .deb packages and dependencies...
2025-05-30 05:54:34 UTC -- Processing clustercontrol-mcc...

WARNING: apt does not have a stable CLI interface. Use with caution in␣
↪scripts.

--2025-05-30 05:54:38-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-cloud/clustercontrol-cloud
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 16809906 (16M) [application/octet-stream]
Saving to: 'clustercontrol-cloud_2.3.2-423_x86_64.deb'

clustercontrol-cloud_2.3.2-42␣
↪100%[=================================================>]  16.03M ␣
↪1.05MB/s    in 16s

2025-05-30 05:54:55 (1002 KB/s) -
↪'clustercontrol-cloud_2.3.2-423_x86_64.deb' saved [16809906/16809906]

--2025-05-30 05:54:55-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-clud/clustercontrol-clud_2
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13383006 (13M) [application/octet-stream]
Saving to: 'clustercontrol-clud_2.3.2-423_x86_64.deb'

clustercontrol-clud_2.3.2-423␣
↪100%[=================================================>]  12.76M ␣
↪1.67MB/s    in 8.8s

2025-05-30 05:55:04 (1.45 MB/s) -
↪'clustercontrol-clud_2.3.2-423_x86_64.deb' saved [13383006/13383006]

--2025-05-30 05:55:04-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-proxy/clustercontrol-proxy
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7532004 (7.2M) [application/octet-stream]
Saving to: 'clustercontrol-proxy_2.3.2-78_x86_64.deb'

clustercontrol-proxy_2.3.2-78␣
```

```
↪100%[=================================================>]   7.18M ␣
↪1.34MB/s    in 6.0s

2025-05-30 05:55:11 (1.20 MB/s) -
↪'clustercontrol-proxy_2.3.2-78_x86_64.deb' saved [7532004/7532004]

--2025-05-30 05:55:11-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-kuber-proxy/clustercontrol
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15708012 (15M) [application/octet-stream]
Saving to: 'clustercontrol-kuber-proxy-0.1.0-39-Linux.deb'

clustercontrol-kuber-proxy-0.␣
↪100%[=================================================>]  14.98M ␣
↪2.37MB/s    in 8.3s

2025-05-30 05:55:19 (1.81 MB/s) -
↪'clustercontrol-kuber-proxy-0.1.0-39-Linux.deb' saved␣
↪[15708012/15708012]

--2025-05-30 05:55:19-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-notifications/clustercontr
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3291698 (3.1M) [application/octet-stream]
Saving to: 'clustercontrol-notifications_2.3.2-373_x86_64.deb'

clustercontrol-notifications_␣
↪100%[=================================================>]   3.14M ␣
↪1.03MB/s    in 3.0s

2025-05-30 05:55:23 (1.03 MB/s) -
↪'clustercontrol-notifications_2.3.2-373_x86_64.deb' saved␣
↪[3291698/3291698]

--2025-05-30 05:55:23-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-ssh/clustercontrol-ssh_2.3
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3156366 (3.0M) [application/octet-stream]
Saving to: 'clustercontrol-ssh_2.3.2-213_x86_64.deb'

clustercontrol-ssh_2.3.2-213_␣
```

```
↪100%[=================================================>]   3.01M ␣
↪1.14MB/s    in 2.6s

2025-05-30 05:55:26 (1.14 MB/s) - 'clustercontrol-ssh_2.3.2-213_x86_64.deb
↪' saved [3156366/3156366]

--2025-05-30 05:55:26-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-mcc/clustercontrol-mcc_2.3
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6834822 (6.5M) [application/octet-stream]
Saving to: 'clustercontrol-mcc_2.3.2-457_amd64.deb'

clustercontrol-mcc_2.3.2-457_␣
↪100%[=================================================>]   6.52M ␣
↪1.17MB/s    in 6.3s

2025-05-30 05:55:33 (1.03 MB/s) - 'clustercontrol-mcc_2.3.2-457_amd64.deb
↪' saved [6834822/6834822]

2025-05-30 05:55:33 UTC -- Processing clustercontrol-controller...

WARNING: apt does not have a stable CLI interface. Use with caution in␣
↪scripts.

--2025-05-30 05:55:36-- ␣
↪http://repo.severalnines.com/deb/pool/main/c/clustercontrol-controller/clustercontrol-
Resolving repo.severalnines.com (repo.severalnines.com)... 54.247.53.237
Connecting to repo.severalnines.com␣
↪(repo.severalnines.com)|54.247.53.237|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 42128170 (40M) [application/octet-stream]
Saving to: 'clustercontrol-controller-2.3.2-12981-x86_64.deb'

clustercontrol-controller-2.3␣
↪100%[=================================================>]  40.18M ␣
↪1.37MB/s    in 23s

2025-05-30 05:55:59 (1.76 MB/s) -
↪'clustercontrol-controller-2.3.2-12981-x86_64.deb' saved␣
↪[42128170/42128170]

--2025-05-30 05:55:59-- ␣
↪http://us.archive.ubuntu.com/ubuntu/pool/main/f/fonts-android/fonts-droid-fallback_6.0
Resolving us.archive.ubuntu.com (us.archive.ubuntu.com)... 91.189.91.82,␣
↪91.189.91.81, 91.189.91.83, ...
Connecting to us.archive.ubuntu.com␣
↪(us.archive.ubuntu.com)|91.189.91.82|:80... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 1805156 (1.7M) [application/vnd.debian.binary-package]
Saving to: 'fonts-droid-fallback_6.0.1r16-1.1build1_all.deb'

fonts-droid-fallback_6.0.1r16␣
↪100%[=================================================>]   1.72M  ␣
↪871KB/s    in 2.0s

2025-05-30 05:56:02 (871 KB/s) -
↪'fonts-droid-fallback_6.0.1r16-1.1build1_all.deb' saved␣
↪[1805156/1805156]

--2025-05-30 05:56:02-- ␣
↪http://us.archive.ubuntu.com/ubuntu/pool/main/g/gcc-14/libgomp1_14.2.0-4ubuntu2~24.04_
Resolving us.archive.ubuntu.com (us.archive.ubuntu.com)... 91.189.91.82,␣
↪91.189.91.83, 91.189.91.81, ...
Connecting to us.archive.ubuntu.com␣
↪(us.archive.ubuntu.com)|91.189.91.82|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 147940 (144K) [application/vnd.debian.binary-package]
Saving to: 'libgomp1_14.2.0-4ubuntu2~24.04_amd64.deb'

libgomp1_14.2.0-4ubuntu2~24.0␣
↪100%[=================================================>] 144.47K  ␣
↪142KB/s    in 1.0s

2025-05-30 05:56:04 (142 KB/s) - 'libgomp1_14.2.0-4ubuntu2~24.04_amd64.deb
↪' saved [147940/147940]

--2025-05-30 05:56:04-- ␣
↪http://us.archive.ubuntu.com/ubuntu/pool/main/f/fftw3/libfftw3-double3_3.3.10-1ubuntu3
Resolving us.archive.ubuntu.com (us.archive.ubuntu.com)... 91.189.91.82,␣
↪91.189.91.83, 91.189.91.81, ...
Connecting to us.archive.ubuntu.com␣
↪(us.archive.ubuntu.com)|91.189.91.82|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 838254 (819K) [application/vnd.debian.binary-package]
Saving to: 'libfftw3-double3_3.3.10-1ubuntu3_amd64.deb'

libfftw3-double3_3.3.10-1ubun␣
↪100%[=================================================>] 818.61K  ␣
↪440KB/s    in 1.9s

2025-05-30 05:56:06 (440 KB/s) -
↪'libfftw3-double3_3.3.10-1ubuntu3_amd64.deb' saved [838254/838254]

--2025-05-30 05:56:06-- ␣
↪http://us.archive.ubuntu.com/ubuntu/pool/main/f/fonts-dejavu/fonts-dejavu-mono_2.37-8_
Resolving us.archive.ubuntu.com (us.archive.ubuntu.com)... 91.189.91.83,␣
↪91.189.91.81, 91.189.91.82, ...
```

```
Connecting to us.archive.ubuntu.com␣
↪(us.archive.ubuntu.com)|91.189.91.83|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 502232 (490K) [application/vnd.debian.binary-package]
Saving to: 'fonts-dejavu-mono_2.37-8_all.deb'

fonts-dejavu-mono_2.37-8_all.␣
↪100%[=============================================>] 490.46K  ␣
↪301KB/s    in 1.6s

2025-05-30 05:56:09 (301 KB/s) - 'fonts-dejavu-mono_2.37-8_all.deb' saved␣
↪[502232/502232]

...
...
...

dpkg-scanpackages: info: Wrote 140 entries to output Packages file.
2025-05-30 06:04:58 UTC -- Debian repository created successfully in /
↪usr/local/repo/cc_repo
2025-05-30 06:04:58 UTC -- Exporting repository to /
↪usr/local/repo/cc_repo.tar.gz...
2025-05-30 06:05:29 UTC -- Repository exported!
```

The process of creating a local repository by the installer script involves several steps:

1. Create the local repository
2. Download all packages required by ClusterControl, including dependencies
3. Export the repository into a compressed file before transferring it to the air-gapped host.

The tarball is called `cc_repo.tar.gz` and stored at one level above the repository directory (`/usr/local/repo`). The filename can be changed with the `EXPORT_FILE` environment variable.

### 1.10.2 Step 2: Remove default repositories

To ensure a smooth installation process, remove the default repository files on the ClusterControl host so that the imported offline repository (built on step 1) will be used as a default repository.

#### RHEL/RockyLinux/AlmaLinux

The default repositories are defined under `/etc/yum.repos.d` directory. Remove all repository definition files under `/etc/yum.repos/d` as below:

```
rm -f /etc/yum.repos.d/*.repo
```

Or, disable the repositories:

```
sudo sed -i 's/^enabled=1/enabled=0/' /etc/yum.repos.d/*.repo
```

### Debian/Ubuntu

The default repositories are defined inside `/etc/apt/sources.list`. Remove the default lines by either deleting or commenting them out:

```
sudo sed -i '/^deb /d' /etc/apt/sources.list
```

## 1.10.3 Step 3: Import the repository to the air-gapped host

Importing a repository tarball takes two steps:

1. Copy the `install-cc` and `cc_repo.tar.gz` file to the target host. You may use remote copy tools like SFTP, SCP, rsync or use media device like USB drive or DVD.
2. Run the `install-cc` script with the `--import-repo` parameter.

Before running the `--import-repo` command in the air-gapped environment, disable the default repository since we are going to replace the existing repositories with the one that we prepared in the previous step.

The tarball (`cc_repo.tar.gz`) is located in the directory where the installer script is located. Run the repository import process with the following command:

```
./install-cc --import-repo
```

**Example**

Tranfer the installer script, `install-cc` and the local repository tarball, `cc_repo.tar.gz` from the staging host (with Internet) to the offline ClusterControl host (without Internet):

```
[root@staging ~]# sftp vagrant@10.10.10.11
vagrant@10.10.10.11's password:
Connected to 10.10.10.11.
sftp> put install-cc
Uploading install-cc to /home/vagrant/install-cc
install-cc-dev                                                              ␣
↪             100%   59KB   1.4MB/s   00:00
sftp> exit
[root@staging ~]# cd /usr/local/repo/
[root@staging ~]# sftp vagrant@10.10.10.11
vagrant@10.10.10.11's password:
Connected to 10.10.10.11.
sftp> put cc_repo.tar.gz
Uploading cc_repo.tar.gz to /home/vagrant/cc_repo.tar.gz
cc_repo.tar.gz                                                              ␣
↪             100%  266MB  15.5MB/s   00:17
```

Import the repository on the ClusterControl host:

```
[root@clustercontrol ~]:/home/vagrant# ./install-cc --import-repo
This script will configure the Severalnines repository server for both␣
↪deb and rpm packages, and install the ClusterControl Web Application␣
↪and Controller.
It will install a new MySQL server or utilize an existing MySQL server on␣
↪the host.
```

```
2025-05-30 06:09:51 UTC -- Only RHEL/RockyLinux/AlmaLinux 8|9, Debian␣
↪11|12, Ubuntu 20.04|22.04|24.04 LTS versions are supported
2025-05-30 06:09:51 UTC -- Waiting for locks ...
2025-05-30 06:09:51 UTC -- Update distro repos ...
141 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists...
Building dependency tree...
Reading state information...
lsb-release is already the newest version (12.0-2).
lsb-release set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 141 not upgraded.
2025-05-30 06:10:13 UTC -- Extracting /home/vagrant/cc_repo.tar.gz to /
↪usr/local/repo/cc_repo...
2025-05-30 06:10:14 UTC -- Configuring Debian repository...
Get:1 file:/usr/local/repo/cc_repo stable InRelease
Ign:1 file:/usr/local/repo/cc_repo stable InRelease
Get:2 file:/usr/local/repo/cc_repo stable Release [626 B]
Get:2 file:/usr/local/repo/cc_repo stable Release [626 B]
Get:3 file:/usr/local/repo/cc_repo stable Release.gpg
Ign:3 file:/usr/local/repo/cc_repo stable Release.gpg
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages [45.9 kB]
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
```

```
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 Components
Hit:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:8 http://us.archive.ubuntu.com/ubuntu noble InRelease
Get:9 http://us.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:10 http://us.archive.ubuntu.com/ubuntu noble-backports InRelease
Fetched 126 kB in 4s (32.2 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
141 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: No Hash entry in Release file /
 ↪var/lib/apt/lists/partial/_usr_local_repo_cc%5frepo_dists_stable_Release␣
 ↪which is considered strong enough for security purposes
2025-05-30 06:10:20 UTC -- Debian repository successfully configured.
2025-05-30 06:10:20 UTC -- Repository imported!
```

### 1.10.4 Step 4: Install ClusterControl in offline mode

Once the local ClusterControl repository is imported, install ClusterControl without any access to the public internet. The installation process remains the same — simply run the installation script while setting the `OFFLINE=1` environment variable:

```
OFFLINE=1 INNODB_BUFFER_POOL_SIZE={db_memory_allocation} \
  SEND_DIAGNOSTICS=0 \
  S9S_CMON_PASSWORD={cmon_password} \
  S9S_ROOT_PASSWORD={root_password} ./install-cc
```

Adjust the `INNODB_BUFFER_POOL_SIZE` parameter based on the hardware or VM specifications you are using and fill in the `cmon_password` and `root_password` according to your password policy.

To configure Severalnines repository to use HTTPS URL instead, specify `USE_REPO_TLS=1`.

**Example**

Perform the offline ClusterControl installation on the ClusterControl host:

```
[root@clustercontrol ~]# OFFLINE=1 INNODB_BUFFER_POOL_SIZE=512␣
 ↪SEND_DIAGNOSTICS=0 S9S_CMON_PASSWORD=cmon123 S9S_ROOT_PASSWORD=root123 .
 ↪/install-cc
This script will configure the Severalnines repository server for both␣
 ↪deb and rpm packages, and install the ClusterControl Web Application␣
 ↪and Controller.
It will install a new MySQL server or utilize an existing MySQL server on␣
 ↪the host.
2025-05-30 06:12:33 UTC -- Only RHEL/RockyLinux/AlmaLinux 8|9, Debian␣
 ↪11|12, Ubuntu 20.04|22.04|24.04 LTS versions are supported
Reading package lists...
```

```
Building dependency tree...
Reading state information...
lsb-release is already the newest version (12.0-2).
0 upgraded, 0 newly installed, 0 to remove and 141 not upgraded.
2025-05-30 06:12:34 UTC -- System RAM is > 1.5G
2025-05-30 06:12:34 UTC -- Setting MySQL innodb_buffer_pool_size to 40%␣
↪of system RAM
2025-05-30 06:12:34 UTC -- MySQL innodb_buffer_pool_size set to 512M
2025-05-30 06:12:34 UTC -- Run with 'INNODB_BUFFER_POOL_SIZE=512 ./
↪install-cc' to customize the innodb buffer pool setting.

2025-05-30 06:12:34 UTC -- Severalnines would like your help improving␣
↪our installation process.
2025-05-30 06:12:34 UTC -- Information such as OS, memory and install␣
↪success helps us improve how we onboard our users.
2025-05-30 06:12:34 UTC -- None of the collected information identifies␣
↪you personally.
2025-05-30 06:12:34 UTC -- !!

This script will configure the Severalnines repository server for both␣
↪deb and rpm packages, and install the ClusterControl Web Application␣
↪and Controller.
It will install a new MySQL server or utilize an existing MySQL server on␣
↪the host.


2025-05-30 06:12:34 UTC -- Installing required packages ...
Reading package lists...
Building dependency tree...
Reading state information...

...
...
...

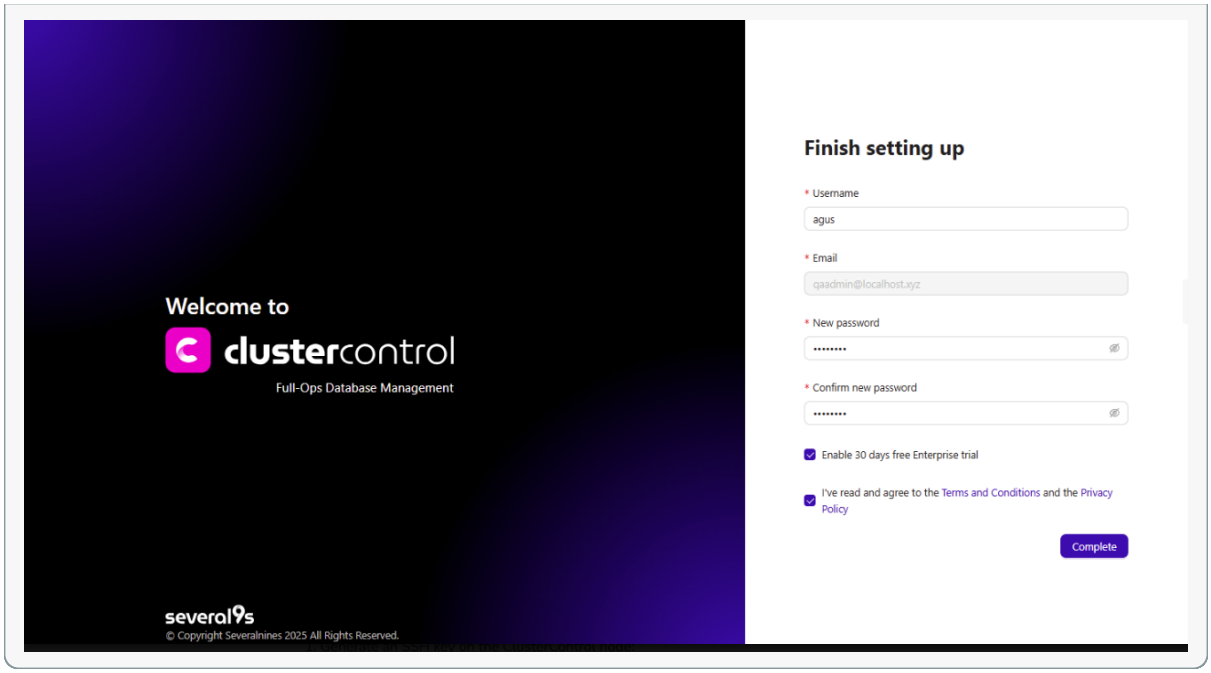
No VM guests are running outdated hypervisor (qemu) binaries on this host.
=> Waiting 60s until the CMON Controller completes its startup cycle ...
2025-05-30 06:18:22 UTC -- Create temporary cc setup user ...
User created.
2025-05-30 06:18:22 UTC -- *** Restarting the Controller process to␣
↪generate CMON API rpc_tls files.
Synchronizing state of cmon-events.service with SysV service script with /
↪usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable cmon-events
Created symlink /
↪etc/systemd/system/multi-user.target.wants/cmon-events.service -> /
↪etc/systemd/system/cmon-events.service.
Synchronizing state of cmon-ssh.service with SysV service script with /
↪usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable cmon-ssh
```

```
Created symlink /
↪etc/systemd/system/multi-user.target.wants/cmon-ssh.service -> /
↪etc/systemd/system/cmon-ssh.service.
Synchronizing state of cmon-cloud.service with SysV service script with /
↪usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable cmon-cloud
Created symlink /
↪etc/systemd/system/multi-user.target.wants/cmon-cloud.service -> /
↪etc/systemd/system/cmon-cloud.service.
2025-05-30 06:18:43 UTC -- Configuring ClusterControl MCC / Ops-C ...
2025-05-30 06:18:43 UTC -- Configuring ccmgr (cmon-proxy) ...
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing port from 19051 to 443
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
2025-05-30 06:18:43 UTC -- ClusterControl MCC / Ops-C frontend installed .
↪..
2025-05-30 06:18:43 UTC -- ClusterControl installation completed!
Open your web browser to https://10.10.10.11 to start using␣
↪ClusterControl.
If you want to uninstall ClusterControl then please follow the␣
↪instructions here,␣
↪https://severalnines.com/docs/administration.html#uninstall
```

### 1.10.5 Step 5: Post-installation

Access the ClusterControl GUI at `https://<ClusterControl_host>` and create a new admin user.

**Example**




Once ClusterControl is up and running, import your existing cluster or deploy a new database cluster and start managing them from one place. Make sure SSH key-based authentication is configured from the ClusterControl node to the database nodes.

1. Generate an SSH key on the ClusterControl node:

```
ssh-keygen -t rsa # press Enter on all prompts
```

2. Setup SSH key-based authentication from ClusterControl to the database and load balancer nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa <os_user>@<IP address/hostname>
```

Repeat step 2 for every database/load balancer host that you are going to manage.

## 1.11 Installer Setup

ClusterControl provides a standalone offline installer that can be used to install ClusterControl in an air-gapped environment without requiring internet access during the installation.

1. Download the latest ClusterControl offline installer from the following URL (requires internet access):

```
wget https://go.severalnines.com/cc/offline-installer
```

2. Transfer the downloaded installer file to the target server in the air-gapped environment using your preferred file transfer method.
3. Grant execute permission to the installer:

```
chmod +x offline-installer
```

Then, run the installer:

```
./offline-installer
```

**Example**

Perform the offline ClusterControl installation using installer file on the ClusterControl host:

```
root@clustercontrol:~# ./offline-installer
Verifying archive integrity...  100%
SHA256 checksums are OK.  100%
MD5 checksums are OK. All good.
Uncompressing ClusterControl Offline Installer v2.5.0 build 6  100%
Detected OS: ubuntu
Version: 24
=== Starting Offline Installation at Fri Sep 25 07:35:02 UTC 2026 ===
Using buffer pool size: 512M
CMON password and root password configured (not shown).
Running commands for Ubuntu 22.04/24.04
Backing up and removing files in /etc/apt/sources.list.d/
Importing offline repository...
This script will configure the Severalnines repository server for both␣
↪deb and rpm packages, and install the ClusterControl Web Application␣
↪and Controller.
It will install a new MySQL server or utilize an existing MySQL server on␣
↪the host.
2026-09-25 07:35:14 UTC -- Only RHEL/RockyLinux/AlmaLinux 8|9, Debian␣
↪11|12, Ubuntu 20.04|22.04|24.04 LTS versions are supported
2026-09-25 07:35:14 UTC -- Waiting for locks ...
2026-09-25 07:35:14 UTC -- Update distro repos ...

WARNING: apt does not have a stable CLI interface. Use with caution in␣
↪scripts.

All packages are up to date.

WARNING: apt does not have a stable CLI interface. Use with caution in␣
↪scripts.

Reading package lists...
Building dependency tree...
Reading state information...
lsb-release is already the newest version (12.0-2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
2026-09-25 07:35:14 UTC -- Extracting /tmp/cc_repo.tar.gz to /
↪usr/local/repo/cc_repo...
2026-09-25 07:35:16 UTC -- Configuring Debian repository...

WARNING: apt does not have a stable CLI interface. Use with caution in␣
↪scripts.

Get:1 file:/usr/local/repo/cc_repo stable InRelease
```

```
Ign:1 file:/usr/local/repo/cc_repo stable InRelease
Get:2 file:/usr/local/repo/cc_repo stable Release [626 B]
Get:2 file:/usr/local/repo/cc_repo stable Release [626 B]
Get:3 file:/usr/local/repo/cc_repo stable Release.gpg
Ign:3 file:/usr/local/repo/cc_repo stable Release.gpg
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages [38.7 kB]
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Get:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Ign:5 file:/usr/local/repo/cc_repo stable/main Translation-en
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Ign:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Ign:6 file:/usr/local/repo/cc_repo stable/main amd64 c-n-f Metadata
Get:4 file:/usr/local/repo/cc_repo stable/main amd64 Packages [135 kB]
Reading package lists...
Building dependency tree...
Reading state information...
All packages are up to date.
```

```
W: No Hash entry in Release file /
 ↪var/lib/apt/lists/partial/_usr_local_repo_cc%5frepo_dists_stable_Release␣
 ↪which is considered strong enough for security purposes
2026-09-25 07:35:35 UTC -- Debian repository successfully configured.
2026-09-25 07:35:35 UTC -- Repository imported!
Running ClusterControl installation...

This script will configure the Severalnines repository server for both␣
 ↪deb and rpm packages, and install the ClusterControl Web Application␣
 ↪and Controller.
It will install a new MySQL server or utilize an existing MySQL server on␣
 ↪the host.

Determining network interfaces ...
2026-09-25 07:35:37 UTC -- Web application IP => https://157.180.11.42

2026-09-25 07:35:37 UTC -- Using hostname 10.10.12.20

2026-09-25 07:35:37 UTC -- Using web document root /var/www
2026-09-25 07:35:37 UTC -- Install the default distro MySQL Server ...
2026-09-25 07:35:37 UTC -- Installing the ClusterControl Web Application .
 ↪..
2026-09-25 07:35:37 UTC -- Installing ClusterControl frontend packages ...

WARNING: apt does not have a stable CLI interface. Use with caution in␣
 ↪scripts.

Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  clustercontrol-clud clustercontrol-kuber-proxy clustercontrol-proxy
The following NEW packages will be installed:
  clustercontrol-cloud clustercontrol-clud clustercontrol-kuber-proxy
  clustercontrol-mcc clustercontrol-notifications clustercontrol-proxy
  clustercontrol-ssh
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/107 MB of archives.
After this operation, 358 MB of additional disk space will be used.
Get:1 file:/usr/local/repo/cc_repo stable/main amd64 clustercontrol-cloud␣
 ↪amd64 2.5.0-487 [24.0 MB]
Get:2 file:/usr/local/repo/cc_repo stable/main amd64 clustercontrol-clud␣
 ↪amd64 2.5.0-487 [19.9 MB]
Get:3 file:/usr/local/repo/cc_repo stable/main amd64 clustercontrol-proxy␣
 ↪amd64 2.5.0-236 [13.0 MB]
Get:4 file:/usr/local/repo/cc_repo stable/main amd64␣
 ↪clustercontrol-kuber-proxy amd64 2.5.0-717 [28.3 MB]
Get:5 file:/usr/local/repo/cc_repo stable/main amd64␣
 ↪clustercontrol-notifications amd64 2.5.0-402 [3980 kB]
Get:6 file:/usr/local/repo/cc_repo stable/main amd64 clustercontrol-ssh␣
```

```
 ↪amd64 2.5.0-262 [12.7 MB]
Get:7 file:/usr/local/repo/cc_repo stable/main amd64 clustercontrol-mcc␣
 ↪amd64 2.5.0-1188 [4711 kB]
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
        LANGUAGE = (unset),
        LC_ALL = "en_US.utf8",
        LANG = "C"
    are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
locale: Cannot set LC_CTYPE to default locale: No such file or directory
locale: Cannot set LC_MESSAGES to default locale: No such file or␣
 ↪directory
locale: Cannot set LC_ALL to default locale: No such file or directory
Selecting previously unselected package clustercontrol-cloud.
(Reading database ... 28335 files and directories currently installed.)
Preparing to unpack .../0-clustercontrol-cloud_2.5.0-487_x86_64.deb ...
Unpacking clustercontrol-cloud (2.5.0-487) ...
Selecting previously unselected package clustercontrol-clud.
Preparing to unpack .../1-clustercontrol-clud_2.5.0-487_x86_64.deb ...
Unpacking clustercontrol-clud (2.5.0-487) ...
Selecting previously unselected package clustercontrol-proxy.
Preparing to unpack .../2-clustercontrol-proxy_2.5.0-236_x86_64.deb ...
Unpacking clustercontrol-proxy (2.5.0-236) ...
Selecting previously unselected package clustercontrol-kuber-proxy.
Preparing to unpack .../3-clustercontrol-kuber-proxy-2.5.0-717-Linux.deb .
 ↪..
Unpacking clustercontrol-kuber-proxy (2.5.0-717) ...
Selecting previously unselected package clustercontrol-notifications.
Preparing to unpack .../
 ↪4-clustercontrol-notifications_2.5.0-402_x86_64.deb ...
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.utf8)
Unpacking clustercontrol-notifications (2.5.0-402) ...
Selecting previously unselected package clustercontrol-ssh.
Preparing to unpack .../5-clustercontrol-ssh_2.5.0-262_x86_64.deb ...
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.utf8)
Unpacking clustercontrol-ssh (2.5.0-262) ...
Selecting previously unselected package clustercontrol-mcc.
Preparing to unpack .../6-clustercontrol-mcc_2.5.0-1188_amd64.deb ...
Unpacking clustercontrol-mcc (2.5.0-1188) ...


No VM guests are running outdated hypervisor (qemu) binaries on this host.
=> Waiting 60s until the CMON Controller completes its startup cycle ...
2026-09-25 07:36:22 UTC -- Create temporary cc setup user ...
User created.
2026-09-25 07:36:22 UTC -- *** Restarting the Controller process to␣
 ↪generate CMON API rpc_tls files.
Synchronizing state of cmon-events.service with SysV service script with /
 ↪usr/lib/systemd/systemd-sysv-install.
```

```
Executing: /usr/lib/systemd/systemd-sysv-install enable cmon-events
Created symlink /
 ↪etc/systemd/system/multi-user.target.wants/cmon-events.service -> /
 ↪etc/systemd/system/cmon-events.service.
Synchronizing state of cmon-ssh.service with SysV service script with /
 ↪usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable cmon-ssh
Created symlink /
 ↪etc/systemd/system/multi-user.target.wants/cmon-ssh.service -> /
 ↪etc/systemd/system/cmon-ssh.service.
Synchronizing state of cmon-cloud.service with SysV service script with /
 ↪usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable cmon-cloud
Created symlink /
 ↪etc/systemd/system/multi-user.target.wants/cmon-cloud.service -> /
 ↪etc/systemd/system/cmon-cloud.service.
2026-09-25 07:36:43 UTC -- Configuring ClusterControl MCC / Ops-C ...
2026-09-25 07:36:43 UTC -- Configuring ccmgr (cmon-proxy) ...
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing port from 19051 to 443
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
2026-09-25 07:36:43 UTC -- ClusterControl MCC / Ops-C frontend installed .
 ↪..
2026-09-25 07:36:43 UTC -- ClusterControl installation completed!
Open your web browser to https://10.10.10.11 to start using␣
 ↪ClusterControl.
If you want to uninstall ClusterControl then please follow the␣
 ↪instructions here,␣
 ↪https://severalnines.com/docs/administration.html#uninstall
```

The ClusterControl offline installer has been tested on the following operating systems:

- Rocky Linux 8.x and 9.x
- Red Hat Enterprise Linux (RHEL) 8.x and 9.x
- CentOS 9.x
- Debian 11 and 12
- Ubuntu 24.04

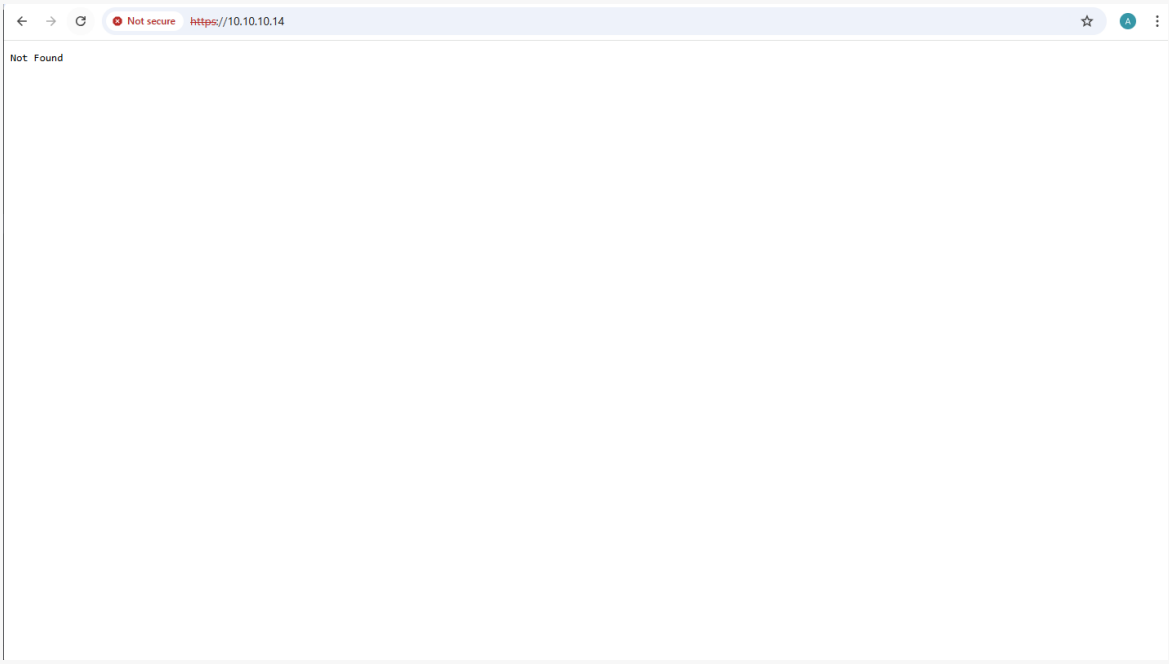
## 1.12 Troubleshooting Issues

### 1.12.1 Error page “Not Found” after installation

In highly restricted environments, after an offline installation of ClusterControl is complete, users may encounter a “Not Found” error when attempting to access the ClusterControl GUI at `https://`

`{ClusterControl_host}/`. This issue typically arises due to incorrect directory permissions within `/var/www/html/clustercontrol-mcc`, mostly originated from a strict default umask.

**Example**




Check the permission for group and others in every directory under `/var/www/html/clustercontrol-mcc`. The directories' permission should be `drwxr-xr-x`. Run the following command on the ClusterControl server:

```
$ chmod 755 /var
$ chmod 755 /var/www
$ chmod 755 /var/www/html
$ chmod 755 /var/www/html/clustercontrol-mcc
```

**Example**

Here is an example output to verify the permission and ownership:

```
$ ls -al /var | grep www
drwxr-xr-x  3 root root   4096 Oct 23 15:26 www
$ ls -al /var/www | grep html
drwxr-xr-x  4 root root 4096 Oct 23 15:29 html
$ ls -al /var/www/html
total 16
drwxr-xr-x 4 root     root     4096 Oct 23 15:29 .
drwxr-xr-x 3 root     root     4096 Oct 23 15:26 ..
drwxr-xr-x 3 root     root     4096 Jan  9 12:44 clustercontrol-mcc
drwxr-x--- 2 www-data www-data 4096 Oct 23 15:29 cmon-repos
```

After the permission for group and others already are correct, restart the `cmon-proxy` service:

```
systemctl restart cmon-proxy
```

Open the ClusterControl GUI in the web browser to verify if the issue is resolved.

## 1.13 Manual Installation

If you want to have more control over the installation process, you may perform a manual installation. ClusterControl requires a number of packages to be installed and configured, as described in the following list:

- `clustercontrol-mcc` – ClusterControl graphical user interface (GUI).
- `clustercontrol-controller` – ClusterControl CMON controller.
- `clustercontrol-notifications` – ClusterControl notification module, to forward alarms and notifications to third-party tools like PagerDuty and Slack.
- `clustercontrol-ssh` – ClusterControl web-based SSH module, to access the host via SSH directly from ClusterControl GUI.
- `clustercontrol-cloud` – ClusterControl cloud module, to integrate with your cloud providers from ClusterControl GUI.
- `clustercontrol-proxy` – ClusterControl controller proxying service for ClusterControl web user interface (GUI).
- `clustercontrol-kuber-proxy` - ClusterControl module for integration to the Kubernetes environment.
- `clustercontrol-clud` – ClusterControl cloud file manager module, to upload and download backups from cloud storage. It requires `clustercontrol-cloud`.
- `s9s-tools` – ClusterControl command-line interface (CLI).

**Note**

Note Installing and uninstalling ClusterControl should not bring any downtime to the managed database cluster.

### 1.13.1 Requirements

Make sure the following is ready prior to this installation:

- The ClusterControl host must be running on the supported operating system. See *Operating System*.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*
- You must have an internet connection on the ClusterControl node during the installation process. Otherwise, see *Offline Installation*.

### 1.13.2 Installation Steps

The steps described in the following sections should be performed on the ClusterControl node unless specified otherwise.

### Red Hat/CentOS/Rocky Linux/AlmaLinux

1. Set up *ClusterControl repository*.
2. Set up *ClusterControl CLI repository*.
3. Disable SElinux and open required ports (or stop firewall):

```
sed -i 's|SELINUX=enforcing|SELINUX=disabled|g' /etc/selinux/config
setenforce 0
systemctl stop firewalld
```

4. Install required packages via package manager:

```
dnf -y install wget dmidecode hostname python3 mariadb mariadb-server
alternatives --set python /usr/bin/python3
```

5. Install EPEL packages:

```
# RHEL/CentOS/Rocky/Alma 9
dnf -y install epel-release

# RHEL/Alma/Rocky 8
dnf config-manager --set-enabled powertools
dnf -y install epel-release epel-next-release
```

6. Install ClusterControl packages:

```
dnf -y install clustercontrol-mcc \
            clustercontrol-controller \
            clustercontrol-ssh \
            clustercontrol-notifications \
            clustercontrol-cloud \
            clustercontrol-clud \
            clustercontrol-proxy \
            clustercontrol-kuber-proxy \
            s9s-tools
```

7. Start the MariaDB server, enable it on boot and set the database root password:

```
systemctl start mariadb
systemctl enable mariadb
mysqladmin -uroot password 'themysqlrootpassword'
```

8. Create a database user called `cmon`, and grant proper privileges:

```
mysql -uroot -p -e 'GRANT ALL PRIVILEGES ON *.* TO "cmon"@"localhost"␣
↪IDENTIFIED BY "{cmonpassword}" WITH GRANT OPTION'
mysql -uroot -p -e 'GRANT ALL PRIVILEGES ON *.* TO "cmon"@"127.0.0.1"␣
↪IDENTIFIED BY "{cmonpassword}" WITH GRANT OPTION'
mysql -uroot -p -e 'GRANT ALL PRIVILEGES ON *.* TO
↪"cmon"@"{controller_ip_address}" IDENTIFIED BY "{cmonpassword}" WITH␣
↪GRANT OPTION'
mysql -uroot -p -e 'FLUSH PRIVILEGES'
```

Replace `{cmonpassword}` with respective value and `{controller_ip_address}` with the valid FQDN or IP address of the ClusterControl node.

9. Generate a ClusterControl key to be used by the `--rpc-token` option further below:

```
$ uuidgen | tr -d '-'
6856d96a19d049aa8a7f4a5ba57a34740b3faf57
```

10. Initialize the ClusterControl Controller service, `cmon` by running the following command:

```
cmon --init \
     --mysql-hostname="127.0.0.1" \
     --mysql-port="3306" \
     --mysql-username="cmon" \
     --mysql-password="{cmonpassword}" \
     --mysql-database="cmon" \
     --hostname="{ClusterControl Primary IP Address}" \
     --rpc-token="{ClusterControl API key as generated above}" \
     --controller-id="clustercontrol"
```

**Example**

```
$ cmon --init \
       --mysql-hostname="127.0.0.1" \
       --mysql-port="3306" \
       --mysql-username="cmon" \
       --mysql-password="xxxx" \
       --mysql-database="cmon" \
       --hostname="10.10.10.13" \
       --rpc-token="dcd17b14e88b47f8ac7f25cd85508fb0" \
       --controller-id="clustercontrol"
   The --init option received, initializing the controller.
   Verifying the Cmon Database...
   Cmon Database connect success, the database is not yet created, ok.
   Checking the Cmon Database schema...
   Cmon Database does not exist, will be created now.
   Applying modifications from
↪'cmon_db.sql,cmon_db_mods_hotfix.sql,cmon_data.sql'.

   Verifying connection...
   Initializing the user manager.
   User manager is creating system users.
   Checking that the system users exist.
   Creating system groups: admins
   Creating system groups: users
   Creating system groups: nobody
   Creating system user.
   Creating nobody user.
  Creating admin user.
```

```
:::{warning} Attention

    The value of the `hostname` must be either a valid FQDN or IP address␣
↪of the ClusterControl node. If the host has multiple IP addresses, pick␣
↪the primary IP address of the host. The `cmon` user password taken from␣
↪previous step when creating the user.
:::
```

11. ClusterControl event and cloud modules require their service definition inside `/etc/default/cmon`. Create the file and add the following lines:

```
EVENTS_CLIENT="http://127.0.0.1:9510"
CLOUD_SERVICE="http://127.0.0.1:9518"
```

12. Initialize the ClusterControl web application to be started on port 443:

```
ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
```

**Example**

```
$ ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
```

```
:::{tip}

If you want to use your own SSL certificate, update the `tls_key` and␣
↪`tls_cert` values inside `/usr/share/ccmgr/ccmgr.yaml` accordingly.
:::
```

13. Enable ClusterControl daemons on boot and start them:

```
systemctl enable cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
systemctl start cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
```

14. Create a user called `ccsetup`, for new registration purposes (if this user exists, ClusterControl GUI first run will default to registration page instead):

```
export S9S_USER_CONFIG=/tmp/ccsetup.conf
s9s user --create --new-password=admin --group=admins --email-address="
↪{your_email_address}" --controller="https://localhost:9501" ccsetup
unset S9S_USER_CONFIG
```

15. Open ClusterControl GUI at `https://<ClusterControl_host>/` and create the default admin user by specifying a username (username "admin" is reserved) and password on the welcome page.

16. Generate an SSH key to be used by ClusterControl when connecting to all managed hosts. In this example, we are using the root user to connect to the managed hosts. To generate an SSH key for the root user, do:

```
$ whoami
  root
$ ssh-keygen -t rsa # Press enter for all prompts
```

> **Note**
>
> Note If you are running as sudoer, the default SSH key will be located under `/home/$USER/.ssh/id_rsa`. See *Operating System User*.

17. Before creating or importing a database server/cluster into ClusterControl, set up passwordless SSH from the ClusterControl host to the database host(s). Use the following command to copy the SSH key to the target hosts:

```
ssh-copy-id -i ~/.ssh/id_rsa <SSH user>@<IP address of the target node>
```

Replace `<SSH user>` and `<IP address of the target node>` with appropriate values. Repeat the command for all target hosts.

The installation is complete and you can start to import existing or deploy a new database cluster. See *User Guide* to start using ClusterControl.

### Debian/Ubuntu

The following steps should be performed on the ClusterControl node unless specified otherwise. Omit sudo if you are installing as the root user.

1. Set up *ClusterControl repository*.
2. Set up *ClusterControl CLI repository*.
3. If you have AppArmor running, disable it and open the required firewall ports (or stop iptables):

```
sudo systemctl stop apparmor
sudo systemctl disable apparmor
sudo systemctl mask apparmor
sudo systemctl stop ufw   # or nftables or iptables
```

4. Install ClusterControl dependencies:

```
sudo apt-get update
sudo apt-get install -y python3 software-properties-common␣
↪mysql-client mysql-server
update-alternatives --install /usr/bin/python python /usr/bin/python3 1
```

5. Install the ClusterControl package:

```
sudo apt-get install -y clustercontrol-controller \
     clustercontrol-mcc \
     clustercontrol-ssh \
     clustercontrol-notifications \
     clustercontrol-cloud \
     clustercontrol-clud \
     clustercontrol-proxy \
     clustercontrol-kuber-proxy \
     s9s-tools
```

6. Start MySQL server, enable it on boot and set a MySQL root password by using the `mysql_secure_installation` script:

```
systemctl start mysql
systemctl enable mysql
mysql_secure_installation
```

7. Create a database user called `cmon` and grant the right database privileges:

```
mysql -uroot -p -e 'GRANT ALL PRIVILEGES ON *.* TO "cmon"@"localhost"␣
↪IDENTIFIED BY "{cmonpassword}" WITH GRANT OPTION'
mysql -uroot -p -e 'GRANT ALL PRIVILEGES ON *.* TO "cmon"@"127.0.0.1"␣
↪IDENTIFIED BY "{cmonpassword}" WITH GRANT OPTION'
mysql -uroot -p -e 'GRANT ALL PRIVILEGES ON *.* TO
↪"cmon"@"{controller_ip_address}" IDENTIFIED BY "{cmonpassword}" WITH␣
↪GRANT OPTION'
mysql -uroot -p -e 'FLUSH PRIVILEGES'
```

8. Generate a ClusterControl key to be used by `--rpc-token` option further below:

```
$ uuidgen | tr -d '-'
6856d96a19d049aa8a7f4a5ba57a34740b3faf57
```

9. Initialize the ClusterControl Controller service called `cmon` by running the following command:

```
cmon --init \
     --mysql-hostname="127.0.0.1" \
     --mysql-port="3306" \
     --mysql-username="cmon" \
     --mysql-password="{cmonpassword}" \
     --mysql-database="cmon" \
     --hostname="{ClusterControl Primary IP Address}" \
     --rpc-token="{ClusterControl API key as generated above}" \
     --controller-id="clustercontrol"
```

**Example**

```
$ cmon --init \
       --mysql-hostname="127.0.0.1" \
       --mysql-port="3306" \
       --mysql-username="cmon" \
```

```
      --mysql-password="xxxx" \
      --mysql-database="cmon" \
      --hostname="10.10.10.13" \
      --rpc-token="dcd17b14e88b47f8ac7f25cd85508fb0" \
      --controller-id="clustercontrol"
   The --init option received, initializing the controller.
   Verifying the Cmon Database...
   Cmon Database connect success, the database is not yet created, ok.
   Checking the Cmon Database schema...
   Cmon Database does not exist, will be created now.
   Applying modifications from
↪'cmon_db.sql,cmon_db_mods_hotfix.sql,cmon_data.sql'.

   Verifying connection...
   Initializing the user manager.
   User manager is creating system users.
   Checking that the system users exist.
   Creating system groups: admins
   Creating system groups: users
   Creating system groups: nobody
   Creating system user.
   Creating nobody user.
  Creating admin user.
```

```
:::{warning} Attention

   The value of the `hostname` must be either a valid FQDN or IP address␣
↪of the ClusterControl node. If the host has multiple IP addresses, pick␣
↪the primary IP address of the host. The `cmon` user password taken from␣
↪previous step when creating the user.
:::
```

10. ClusterControl's event and cloud modules require `/etc/default/cmon` for service definition. Create the file and add the following lines:

```
EVENTS_CLIENT="http://127.0.0.1:9510"
CLOUD_SERVICE="http://127.0.0.1:9518"
```

11. Initialize the ClusterControl web application to be started on port 443:

```
ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
```

**Example**

```
$ ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
```

```
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
```

```
:::{tip}

If you want to use your own SSL certificate, update the `tls_key` and␣
↪`tls_cert` values inside `/usr/share/ccmgr/ccmgr.yaml` accordingly.
:::
```

12. Enable ClusterControl on boot and start them:

```
systemctl daemon-reload
systemctl enable cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
systemctl restart cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
```

13. Create a user called `ccsetup`, for new registration purposes (if this user exists, ClusterControl GUI first run will default to registration page instead):

```
export S9S_USER_CONFIG=/tmp/ccsetup.conf
s9s user --create --new-password=admin --group=admins --email-address="
↪{your_email_address}" --controller="https://localhost:9501" ccsetup
unset S9S_USER_CONFIG
```

14. Open ClusterControl GUI at `https://<ClusterControl_host>/` and create the default admin user by specifying a username (username "admin" is reserved) and password on the welcome page.

15. Generate an SSH key to be used by ClusterControl when connecting to all managed hosts. In this example, we are using the "root" user to connect to the managed hosts. To generate an SSH key for the root user, do:

```
$ whoami
  root
$ ssh-keygen -t rsa # Press enter for all prompts
```

> **Note**
>
> Note If you are running as sudoer, the default SSH key will be located under `/home/$USER/.ssh/id_rsa`. See *Operating System User*.

16. Before importing a database server/cluster into ClusterControl or deploy a new cluster, set up passwordless SSH from ClusterControl host to the database host(s). Use the following command to copy the SSH key to the target hosts:

```
ssh-copy-id -i ~/.ssh/id_rsa <SSH user>@<IP address of the target node>
```

Replace `<SSH user>` and `<IP address of the target node>` with appropriate values. Repeat the command for all target hosts.

The installation is complete and you can start to import existing or deploy a new database cluster. See *User Guide* to start using ClusterControl.

### 1.13.3 Troubleshooting Issues

#### Failed to create `ccsetup` user

In some cases, the `ccsetup` user was failed to be created with the following error:

```
s9s user --create --new-password=admin --group=admins --email-address=
↪"user@example.com" --controller="https://localhost:9501" ccsetup
Connect to localhost:9501 failed(111): Connection refused.
```

Check the ClusterControl Controller log messages in `/var/log/cmon.log`, and see if there is the following error:

```
2025-08-21T07:33:33.296Z : (INFO) Checking if CmonDb access is working␣
↪properly.
2025-08-21T07:33:33.297Z : (WARNING) Cmon DB connection error: Access␣
↪denied for user 'cmon'@'localhost' (using password: YES) (errno: 1045)
2025-08-21T07:33:33.298Z : (INFO) CmonDb connection or query failure. Error␣
↪code: 1045, Message: No connection (Access denied for user 'cmon'@
↪'localhost' (using password: YES))
2025-08-21T07:33:33.298Z : (INFO) Lets block and wait for working CmonDb␣
↪connection.
```

Remove the existing `/etc/cmon.cnf` configuration, and run again `cmon --init` script.

```
$ rm -rf /etc/cmon.cnf
$ cmon --init \
     --mysql-hostname="127.0.0.1" \
     --mysql-port="3306" \
     --mysql-username="cmon" \
     --mysql-password="xxxx" \
     --mysql-database="cmon" \
     --hostname="10.10.10.13" \
     --rpc-token="dcd17b14e88b47f8ac7f25cd85508fb0" \
     --controller-id="clustercontrol"
The --init option received, initializing the controller.
Generating cmon configuration...
------8<------8<------8<------8<------8<------8<------8<------
#
# Configuration file for the Cmon Controller.
#

#
# The name or IP address of the Cmon Controller.
#
hostname=10.10.10.13

#
# Cmon Database credentials. The controller will use
```

(continues on next page)

(continued from previous page)

```
# this database to store its own data structures.
#
mysql_hostname=127.0.0.1
mysql_port=3306
mysql_password='xxxx'
cmon_user=cmon
cmon_db=cmon
rpc_key=dcd17b14e88b47f8ac7f25cd85508fb0

------8<------8<------8<------8<------8<------8<------8<------
Verifying the Cmon Database...
Cmon Database connect is successful, database exists.
Checking the Cmon Database schema...
Applying modifications from
↪'cmon_db_mods-2.2.3-2.3.2.sql,cmon_db_mods_hotfix.sql'.

Verifying connection...
Initializing the user manager.
User manager is creating system users.
Checking that the system users exist.
```

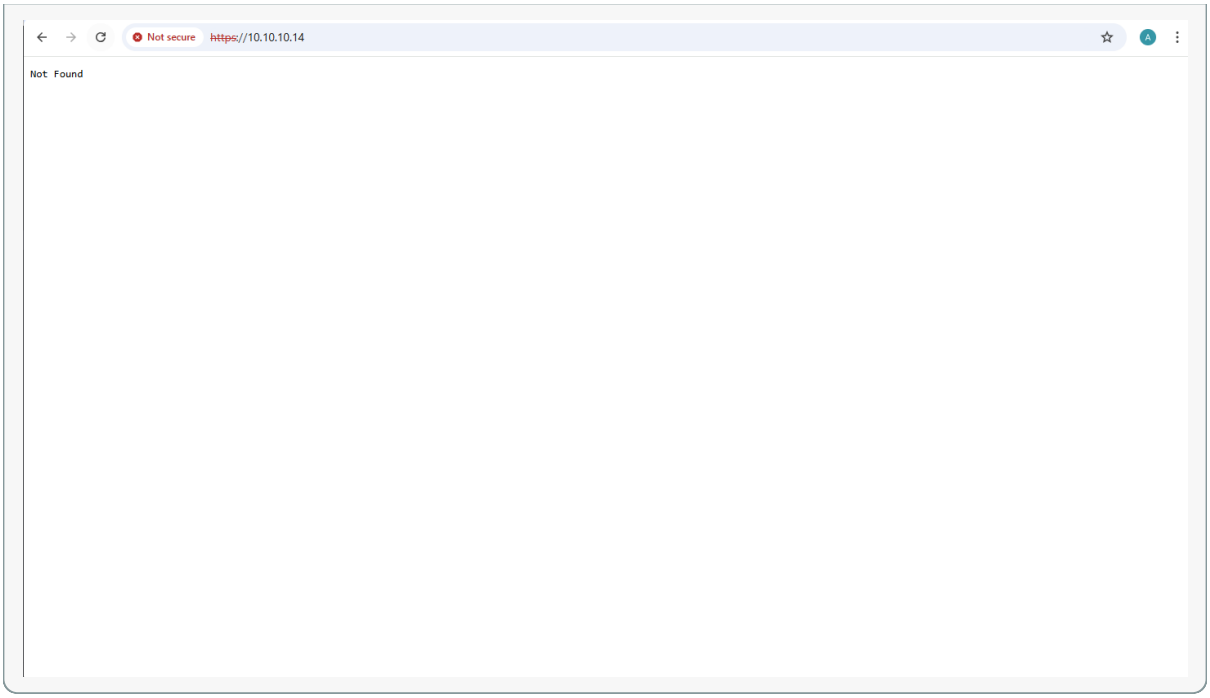
After that, re-create the `ccsetup` user using ClusterControl CLI:

```
s9s user --create --new-password=admin --group=admins --email-address=
↪"user@example.com" --controller="https://localhost:9501" ccsetup
```

### Error page "Not Found" after installation

In highly restricted environments, after a manual installation of ClusterControl is complete, users may encounter a "Not Found" error when attempting to access the ClusterControl GUI at `https://{ClusterControl_host}/`. This issue typically arises due to incorrect directory permissions within `/var/www/html/clustercontrol-mcc`, mostly originated from a strict default umask.

**Example**




Check the permission for group and others in every directory under `/var/www/html/clustercontrol-mcc`. The directories' permission should be `drwxr-xr-x`. Run the following command on the ClusterControl server:

```
$ chmod 755 /var
$ chmod 755 /var/www
$ chmod 755 /var/www/html
$ chmod 755 /var/www/html/clustercontrol-mcc
```

**Example**

Here is an example output to verify the permission and ownership:

```
$ ls -al /var | grep www
drwxr-xr-x  3 root root   4096 Oct 23 15:26 www
$ ls -al /var/www | grep html
drwxr-xr-x  4 root root 4096 Oct 23 15:29 html
$ ls -al /var/www/html
total 16
drwxr-xr-x 4 root     root     4096 Oct 23 15:29 .
drwxr-xr-x 3 root     root     4096 Oct 23 15:26 ..
drwxr-xr-x 3 root     root     4096 Jan  9 12:44 clustercontrol-mcc
drwxr-x--- 2 www-data www-data 4096 Oct 23 15:29 cmon-repos
```

After the permission for group and others already are correct, restart the `cmon-proxy` service:

Severalnines provides RPM-based (yum/dnf) and DEB-based package repositories accessible at https://repo.severalnines.com. The automatic installation procedure (as described in *Online Installation*) automatically configures the ClusterControl node with the respective package repository.

## 1.14 ClusterControl repository

ClusterControl repository offers ClusterControl-related packages that should be installed on the ClusterControl server. The repository provides the following packages:

- ClusterControl Controller
- ClusterControl GUI
- ClusterControl Notifications
- ClusterControl Web SSH
- ClusterControl Cloud
- ClusterControl Cloud File Manager
- ClusterControl Ops-C
- ClusterControl Proxy
- ClusterControl Agent

### yum/dnf

1. Add the Severalnines repository public key into your RPM keyring:

```
rpm --import http://repo.severalnines.com/severalnines-repos.asc
```

2. You can download the repository definition from Severalnines download page:

```
wget http://www.severalnines.com/downloads/cmon/s9s-repo.repo -P /
↪etc/yum.repos.d/
```

Or, you can create the repository definition manually:

```
cat - > /etc/yum.repos.d/s9s-repo.repo << REPOEND
[s9s-repo]
name = Severalnines Release Repository
baseurl = http://repo.severalnines.com/rpm/os/x86_64
enabled = 1
gpgkey = http://repo.severalnines.com/severalnines-repos.asc
gpgcheck = 1
REPOEND
```

3. Look for ClusterControl packages:

```
yum search clustercontrol
```

### apt

1. Add Severalnines repository public key into your APT keyring:

```
apt-get -y install gnupg2
wget http://repo.severalnines.com/severalnines-repos.asc -O- | sudo␣
↪apt-key add -
```

2. You can download the repository definition from Severalnines download page:

```
sudo wget http://www.severalnines.com/downloads/cmon/s9s-repo.list -P /
↪etc/apt/sources.list.d/
```

Or, add the Severalnines APT source list manually:

```
echo 'deb [arch=amd64] http://repo.severalnines.com/deb ubuntu main' |␣
↪sudo tee /etc/apt/sources.list.d/s9s-repo.list
```

3. Update package list:

```
sudo apt-get update
```

4. Look for ClusterControl packages:

```
sudo apt-cache search clustercontrol
```

### Manual download

Severalnines offers a dedicated download page for manual download at Severalnines Download page. Use the filter box to find the right packages and versions quickly.

## 1.15 ClusterControl CLI repository

ClusterControl CLI repository offers ClusterControl client-related packages that should be installed on the client host or workstation. It supports more operating systems than the *ClusterControl repository*. The repository provides the following packages:

- ClusterControl CLI (`s9s`)
- ClusterControl CLI library

### Package repository (yum/dnf/apt)

Use the `install-s9s-tools.sh` script to install ClusterControl CLI and its library. The script will detect the operating system, configure the necessary GPG key, repository definition, download and install the necessary packages and also initialize the CLI user.

1. Download the repository installer script from here:

```
wget https://repo.severalnines.com/s9s-tools/install-s9s-tools.sh
```

2. Run the installer script:

```
bash install-s9s-tools.sh
```

### Manual download

To repository page is accessible publicly at https://repo.severalnines.com/s9s-tools/. You can download the ClusterContol CLI packages directly from there.

## 1.16 Legacy GUI Installation

Starting from ClusterControl 1.9.7 (September 2023), ClusterControl GUI v2 is the default frontend graphical user interface for ClusterControl, accessible via `https://{ClusterControl_host}/`. For new installations, the installer script will only install GUI v2, skipping GUI v1. If you would like to have GUI v1, follow the instructions as written in this article.

### 1.16.1 Caveats

Before continuing with the installation, do mind the following important points:

- ClusterControl GUI v1 requires PHP. It only supports the outdated PHP 7. It does not support the latest version of PHP 8 at the time of this writing. If the operating system comes with PHP 8 installed by default, you have to downgrade to PHP 7 to continue using it. ClusterControl GUI v2 does not have this requirement.
- It requires a MySQL/MariaDB database called "dcps".
- Only the Apache web server is supported due to its dependency on .htaccess and Apache configuration files.
- ClusterControl GUI v1 has reached the end of the development cycle and is considered a feature-freeze product. All new developments will be happening on ClusterControl GUI v2.
- To run ClusterControl GUI v1 on a Red Hat-based distribution, SELinux must be turned off. The installer script only configures the SELinux policy for ClusterControl GUI v2.

### 1.16.2 Installation steps

The following installation steps are written based on Ubuntu 22.04 LTS and Rocky Linux 9.2. It is assumed that you have already performed a ClusterControl installation using the installer script, `install-cc` which by default would give you ClusterControl GUI v2. See the *Quickstart* section for details.

#### Rocky Linux 9.2

1. If you have created an admin user during the first login of ClusterControl GUI v2, proceed to step 2. Otherwise, create a ClusterControl admin user if you haven't done so by accessing ClusterControl GUI v2.
2. Disable SELinux by configuring the following line inside `/etc/selinux/config`:

   ```
   SELINUX=disabled
   ```

   Deactivate SELinux during runtime:

   ```
   setenforce 0
   ```

3. Install ClusterControl GUI v1 package:

   ```
   dnf install -y clustercontrol
   ```

4. Copy the provided Apache configuration files to the Apache configuration directory:

   ```
   cp -f /var/www/html/clustercontrol/app/tools/apache2/s9s*.conf /
   ↪etc/httpd/conf.d/
   ```

5. Add `Listen {HTTPS port}` at the top of the `s9s-ssl.conf` file (note that port 443 is already taken by GUI v2, so we have to pick another port):

```
vim /etc/httpd/conf.d/s9s-ssl.conf
```

Add the following in the first line:

```
Listen 9443
```

In the same file, configure the same HTTPS port as defined above on line 3:

```
<VirtualHost _default_:9443>
```

Also in the same file, configure the SSL certificate and key with the provided cert and key (lines 55 and 56, after you have added the first line above):

```
            SSLCertificateFile      /etc/ssl/certs/s9server.crt
            SSLCertificateKeyFile /etc/ssl/private/s9server.key
```

Restart the Apache web server to apply the changes:

```
systemctl restart httpd
```

6. Check the provided PHP module of the operating system. For Rocky Linux 9.2, the PHP module provided by the default AppStream repository is 8.1, which is not supported. We can check using the following command:

```
dnf module list php
```

7. If PHP7 is unavailable as in this case, we will configure the Remi repository:

```
dnf install dnf-utils http://rpms.remirepo.net/enterprise/remi-release-
↪9.rpm -y
```

8. Verify that the PHP 7 module is listed (in this example, it is `remi-7.4`):

```
$ dnf module list php -y

Rocky Linux 9 - AppStream
Name            Stream              Profiles                              ␣
↪Summary
php             8.1                 common [d], devel, minimal            ␣
↪PHP scripting language

Remi's Modular repository for Enterprise Linux 9 - x86_64
Name            Stream              Profiles                              ␣
↪Summary
php             remi-7.4            common [d], devel, minimal            ␣
↪PHP scripting language
php             remi-8.0            common [d], devel, minimal            ␣
↪PHP scripting language
php             remi-8.1            common [d], devel, minimal            ␣
```

(continues on next page)

(continued from previous page)

```
↪PHP scripting language
php             remi-8.2            common [d], devel, minimal          ␣
↪PHP scripting language
php             remi-8.3            common [d], devel, minimal          ␣
↪PHP scripting language
```

9. Switch PHP to use `remi-7.4` module:

```
dnf module -y switch-to php:remi-7.4
```

10. Install the required PHP7 packages:

```
dnf install -y php php-mysql php-ldap php-gd php-curl php-xml php-fpm
```

11. Start the php-fpm service and restart Apache web server:

```
systemctl start php-fpm
systemctl enable php-fpm
systemctl restart httpd
```

12. Create `bootstrap.php` from the provided template:

```
cp /var/www/html/clustercontrol/bootstrap.php.default /
↪var/www/html/clustercontrol/bootstrap.php
```

13. Configure `/var/www/html/clustercontrol/bootstrap.php` and set the password, `DBPASS`, port, `DBPORT` and `RPCTOKEN` accordingly:

```
define('DB_PASS', 'DBPASS');
define('DB_PORT', 'DBPORT');
define('RPC_TOKEN','RPCTOKEN');
```

**Note**

Note

Set the password and port used when installing CCv2 for the cmon MySQL user. You can also retrieve the information from `/etc/cmon.cnf` (including the RPC token). :::

**Example output**

```
...
define('DB_HOST', '127.0.0.1');
define('DB_LOGIN', 'cmon');
define('DB_PASS', 'PjsL23&H7dNsN$b');
define('DB_NAME', 'dcps');
define('DB_PORT', '3306');
...
define('RPC_TOKEN','80e0a3c8894e931f8b26d6f196c52a9818e513ba');
```

14. Login to MySQL and create the 'dcps' database:

```
mysql -ucmon -p -e 'CREATE DATABASE dcps'
```

> **Note**
>
> Note

We skip configuring MySQL user privilege since the "cmon" MySQL user has a global admin privilege. :::
15. Import the `dcps` database schema structure from the provided SQL file:

```
```bash
mysql -ucmon -p dcps < /var/www/html/clustercontrol/sql/dc-schema.sql
```
```

16. Finally, add the following database flag to skip the registration and admin user creation page since we have done it via ClusterControl GUI v2 when creating the first admin user:

```
$ mysql -ucmon -p
mysql> INSERT INTO dcps.settings (name, type) VALUES (
↪'registration_complete', 'settings');
```

17. The installation is now complete. ClusterControl GUI v1 should be accessible at `https://{ClusterControl_host}:9443/` . The final thing to do is to import the existing clusters (if you have any) into ClusterControl GUI v1, by executing this script:

```
#!/bin/bash

function import() {
for i in /etc/cmon.d/cmon_*.cnf; do
    NAME=$(grep ^name= $i | cut -d "=" -f2 | tr -d "'")
    TYPE=$(grep ^cluster_type= $i | cut -d "=" -f2)
    TOKEN=$(grep ^rpc_key= $i | cut -d "=" -f2 | tr -d "'")
    CREATED_BY=$(grep ^owner= $i | cut -d "=" -f2)
    CID=$(grep ^cluster_id= $i | cut -d "=" -f2)
    DBUSER=cmon
    DBPASS=$(grep ^mysql_password= $i | cut -d "=" -f2 | tr -d "'")


    echo "Importing cluster ID = $CID, from $i"
    MYSQL_PWD="$DBPASS" mysql -u$DBUSER -f -A -Bse \
        "INSERT IGNORE INTO dcps.clusters SET company_id = 1, \
            id = ${CID}, \
            cluster_id = ${CID}, \
            name = '${NAME}', \
            token = '${TOKEN}', \
            cmon_db = 'cmon', \
            type = '${TYPE}', \
            status = 100, \
            created_by = $CREATED_BY"
```

(continues on next page)

(continued from previous page)

```
    retval=$?

    if [ $retval -eq 0 ]; then
        echo 'OK'
    else
        echo 'ERROR. Fix the script and try again.'
        exit 1
    fi
done

echo 'Import complete.'
exit 0
}

echo 'Use this script to import clusters into ClusterControl GUI v1'
echo 'after GUI v1 is installed and running (v1.9.7 only).'
echo 'This script can be run multiple times if any error occurs.'
echo
read -p "Do you wish to continue? [Y/n] " yn
case $yn in
        [Yy]* ) import; break;;
        * ) echo 'Exiting..'; exit 1;;
esac
```

Copy the above code inside the ClusterControl server and save it as `import_cluster_guiv1.sh`. Example output as below:

```
$ vim import_cluster_guiv1.sh # paste the code & save
$ chmod 755 import_cluster_guiv1.sh

$ ./import_cluster_guiv1.sh
Use this script to import clusters into ClusterControl GUI v1
after GUI v1 is installed and running (v1.9.7 only).
This script can be run multiple times if any error occurs.

Do you wish to continue? [Y/n] Y
Importing cluster ID = 2, from /etc/cmon.d/cmon_2.cnf
OK
Importing cluster ID = 3, from /etc/cmon.d/cmon_3.cnf
OK
Importing cluster ID = 4, from /etc/cmon.d/cmon_4.cnf
OK
Importing cluster ID = 7, from /etc/cmon.d/cmon_7.cnf
OK
```

### Ubuntu 22.04

1. If you have created an admin user during the first login of ClusterControl GUI v2, proceed to step 2. Otherwise, create a ClusterControl admin user if you haven't done so by accessing ClusterControl GUI v2.

2. Install ClusterControl GUI v1 package:

```
apt update -y
apt install -y clustercontrol
```

3. Copy the provided Apache configuration files to the Apache configuration directory:

```
cp -f /var/www/html/clustercontrol/app/tools/apache2/s9s*.conf /
↪etc/apache2/sites-enabled/
```

4. Add `Listen {HTTPS port}` at the top of the `s9s-ssl.conf` file (note that port 443 is already taken by GUI v2, so we have to pick another port):

```
vim /etc/apache2/sites-enabled/s9s-ssl.conf
```

Add the following in the first line:

```
Listen 9443
```

In the same file, configure the same HTTPS port as defined above on line 3:

```
<VirtualHost _default_:9443>
```

Also in the same file, configure the SSL certificate and key with the provided cert and key (lines 55 and 56, after you have added the first line above):

```
            SSLCertificateFile      /etc/ssl/certs/s9server.crt
            SSLCertificateKeyFile /etc/ssl/private/s9server.key
```

Restart the Apache web server to apply the changes:

```
systemctl restart apache2
```

5. By default, the installer script configures the PHP7 repository called "ondrej". You may verify by using the following command:

```
apt-cache policy | grep ondrej
```

6. If the repository is not listed in the above output, you may install it by using the following commands:

```
apt install -y software-properties-common apt-transport-https
add-apt-repository -y ppa:ondrej/php
apt update -y
```

7. Install the required PHP7 packages:

```
apt install -y php7.4-mysql php7.4-gd libapache2-mod-php7.4␣
↪php7.4-curl php7.4-ldap php7.4-xml
```

**Note**

Note

By default, the installer script will install the above packages. In that case, the above command will return those packages as installed. ::: 8. Create `bootstrap.php` from the provided template:

```
```bash
cp /var/www/html/clustercontrol/bootstrap.php.default /
↪var/www/html/clustercontrol/bootstrap.php
```
```

9. Configure `/var/www/html/clustercontrol/bootstrap.php` and set the password, `DBPASS`, port, `DBPORT` and `RPCTOKEN` accordingly:

```
define('DB_PASS', 'DBPASS');
define('DB_PORT', 'DBPORT');
define('RPC_TOKEN','RPCTOKEN');
```

**Note**

Note

Set the password and port used when installing CCv2 for the cmon MySQL user. You can also retrieve the information from `/etc/cmon.cnf` (including the RPC token). :::

**Example output**

```
...
define('DB_HOST', '127.0.0.1');
define('DB_LOGIN', 'cmon');
define('DB_PASS', 'PjsL23&H7dNsN$b');
define('DB_NAME', 'dcps');
define('DB_PORT', '3306');
...
define('RPC_TOKEN','80e0a3c8894e931f8b26d6f196c52a9818e513ba');
```

10. Login to MySQL and create the 'dcps' database:

```
mysql -ucmon -p -e 'CREATE DATABASE dcps'
```

**Note**

Note

We skip configuring MySQL user privilege since the "cmon" MySQL user has a global admin privilege. ::: 11. Import the `dcps` database schema structure from the provided SQL file:

```
```bash
mysql -ucmon -p dcps < /var/www/html/clustercontrol/sql/dc-schema.sql
```
```

12. Finally, add the following database flag to skip the registration and admin user creation page since we have done it via ClusterControl GUI v2 when creating the first admin user:

```
$ mysql -ucmon -p
mysql> INSERT INTO dcps.settings (name, type) VALUES (
↪'registration_complete', 'settings');
```

13. The installation is now complete. ClusterControl GUI v1 should be accessible at `https://{ClusterControl_host}:9443/` . The final thing to do is to import the existing clusters (if you have any) into ClusterControl GUI v1, by executing this script:

```
#!/bin/bash

function import() {
for i in /etc/cmon.d/cmon_*.cnf; do
    NAME=$(grep ^name= $i | cut -d "=" -f2 | tr -d "'")
    TYPE=$(grep ^cluster_type= $i | cut -d "=" -f2)
    TOKEN=$(grep ^rpc_key= $i | cut -d "=" -f2 | tr -d "'")
    CREATED_BY=$(grep ^owner= $i | cut -d "=" -f2)
    CID=$(grep ^cluster_id= $i | cut -d "=" -f2)
    DBUSER=cmon
    DBPASS=$(grep ^mysql_password= $i | cut -d "=" -f2 | tr -d "'")


    echo "Importing cluster ID = $CID, from $i"
    MYSQL_PWD="$DBPASS" mysql -u$DBUSER -f -A -Bse \
        "INSERT IGNORE INTO dcps.clusters SET company_id = 1, \
            id = ${CID}, \
            cluster_id = ${CID}, \
            name = '${NAME}', \
            token = '${TOKEN}', \
            cmon_db = 'cmon', \
            type = '${TYPE}', \
            status = 100, \
            created_by = $CREATED_BY"
    retval=$?

    if [ $retval -eq 0 ]; then
        echo 'OK'
    else
        echo 'ERROR. Fix the script and try again.'
        exit 1
    fi
done

echo 'Import complete.'
```

(continues on next page)

(continued from previous page)

```
exit 0
}

echo 'Use this script to import clusters into ClusterControl GUI v1'
echo 'after GUI v1 is installed and running (v1.9.7 only).'
echo 'This script can be run multiple times if any error occurs.'
echo
read -p "Do you wish to continue? [Y/n] " yn
case $yn in
        [Yy]* ) import; break;;
        * ) echo 'Exiting..'; exit 1;;
esac
```

Copy the above code inside the ClusterControl server and save it as `import_cluster_guiv1.sh`. Example output as below:

```
$ vim import_cluster_guiv1.sh # paste the code & save
$ chmod 755 import_cluster_guiv1.sh

$ ./import_cluster_guiv1.sh
Use this script to import clusters into ClusterControl GUI v1
after GUI v1 is installed and running (v1.9.7 only).
This script can be run multiple times if any error occurs.

Do you wish to continue? [Y/n] Y
Importing cluster ID = 2, from /etc/cmon.d/cmon_2.cnf
OK
Importing cluster ID = 3, from /etc/cmon.d/cmon_3.cnf
OK
Importing cluster ID = 4, from /etc/cmon.d/cmon_4.cnf
OK
Importing cluster ID = 7, from /etc/cmon.d/cmon_7.cnf
OK
```

## 1.17 ClusterControl on Marketplaces

An online/cloud marketplace provides users with access to applications and services that are built on, integrated with, or complement the service provider's offering. Most offer pre-configured one-click provisioning of applications and services.

ClusterControl will be offered on a number of marketplaces, allowing you to easily deploy a ClusterControl instance that you can use to manage your database nodes on that service provider's platform. The marketplace requirements on how to create an application differ and in some cases this will require that a few extra steps are taken by you to get started.

### 1.17.1 Current Marketplaces

- DigitalOcean
- gridscale.io
- Vultr
- Linode
- Google Cloud Platform
- Stay tuned … more to come!

### 1.17.2 Passwordless SSH connections

ClusterControl requires password-less SSH access to the hosts where the database nodes will be managed. The ClusterControl application can in some marketplaces automatically generate a private and public SSH key for the root user while in others you will need to perform this step manually.

The public SSH key is needed when you launch your hosts to be used for the database nodes. It must be included/selected as one of the SSH keys that will be used when the cloud instances are created otherwise ClusterControl will not be able to connect and manage the database nodes.

### 1.17.3 Launching the ClusterControl Application

On all marketplaces, there is a 1-click button to launch ClusterControl. You are then presented with a launch form where you might need to provide the following:

- OS image to use – Only Ubuntu 18.04 and Ubuntu 20.04 is supported for the ClusterControl node.
- MySQL root user password – If there is no input field for it then a random password will be generated.
- CMON (DB) user password – If there is no input field for it then a random password will be generated.
- Public SSH key – The SSH key to use when launching the image. This is not the same SSH key that ClusterControl will need later.
- Enable private IP – Enable the use of private IP if that option is shown.

### 1.17.4 Getting the Public SSH Key

On some marketplaces, the ClusterControl image is allowed to generate a password-less SSH key automatically at boot up. You can then copy the SSH public key by SSH into the ClusterControl instance where the public SSH key is displayed on login.

If that is not the case you will need to generate the password-less SSH key for the default SSH user by SSH into the ClusterControl instance and perform the following (ex for a root user):

```
$ ssh-keygen -b 2048 -t rsa -f /root/.ssh/id_rsa -q -N ""
$ ssh-keygen -y -f /root/.ssh/id_rsa > /root/.ssh/id_rsa.pub
$ cat /root/.ssh/id_rsa.pub
```

Copy the public SSH key and use it later when the hosts/cloud instances for the database nodes are provisioned.

### 1.17.5 Launching the Hosts for the Database Nodes

Depending on the cloud provider that you are using, you can add the previously created public SSH key to your 'security settings' or add it when the cloud instances are launched. The selected public SSH key(s) will be inserted into the cloud instance's `~/.ssh/authorized_keys` file so that the ClusterControl instance is able to SSH into the host.

You will also need to make sure that the cloud instances are on the same private network as the ClusterControl instance and enable 'private IPs' if available when launching the cloud instances.

### 1.17.6 Deploying a Database Cluster

After you have launched your cloud instances for the database nodes then proceed to open your web browser to `https://<ClusterControl_instance_IP_address>` and enter the **User registration** information for the initial admin user.

Start to deploy a database cluster onto the previously launched hosts by following the *deployment guide*.

## 1.18 Tutorials

This section provides hands-on tutorials to help you master ClusterControl. Each tutorial is designed to give you practical experience and a deeper understanding of ClusterControl's key features and workflows.

### 1.18.1 Day-1 Operations

**"Your First Cluster" Tutorial Series**

**MySQL/MariaDB**

- *Standalone →*
- *Replication →*
- *Group Replication →*
- *Percona XtraDB Cluster →*
- *MariaDB Cluster →*

**PostgreSQL/TimescaleDB**

- *PostgreSQL Streaming Replication →*
- *TimescaleDB Streaming Replication →*
- *PostgreSQL Logical Replication →*

**MongoDB**

- *MongoDB Replica Set →*
- *MongoDB Enterprise Replica Set →*
- *MongoDB Sharded Cluster →*

**Redis/Valkey**

- *Redis Sentinel →*
- *Redis Cluster →*
- *Valkey Sentinel →*
- *Valkey Cluster →*

**SQL Server**

- *Standalone →*
- *Always-On Availability Group →*

**Elasticsearch**

- *Single-node Cluster →*
- *High Availability Cluster →*

**See also**

- *Deploy Database Cluster in Offline Environment →*

### 1.18.2 Day-2 Operations

**See also**

- *Deploy and Failover to Secondary Cluster →*
- *Perform Major Upgrade for PostgreSQL →*
- *Backup to and Restore from Cloud →*
- *Database Point-in-time Recovery →*
- *Connect to ClusterControl RPC API as Read-only User →*
- *Migrating Existing Clusters from ClusterControl v1 to ClusterControl v2 →*
- *Managing Clusters using ClusterControl MCP →*

## 1.19 Standalone MySQL (MariaDB, Percona, Oracle)

This guide will take you through the process of setting up your first single-node MySQL/MariaDB/Percona server using ClusterControl. By following this tutorial, you'll have a fully functional (but non-replicated) database node that you can monitor, manage, and later scale if your requirements change.

A standalone MySQL/MariaDB server is the simplest deployment which perfect for development, testing, or lightweight production workloads. If you decide to expand to replication or a high availability (HA) cluster, you can do so seamlessly from this single-node foundation.

This tutorial applies to the following MySQL variants:

- MariaDB Server
- Percona Server for MySQL
- Percona Pro Server for MySQL
- Oracle MySQL Community Server

**See also**

- MySQL (Oracle) documentation
- MariaDB documentation
- Percona Server for MySQL documentation
- Percona Distribution for MySQL documentation

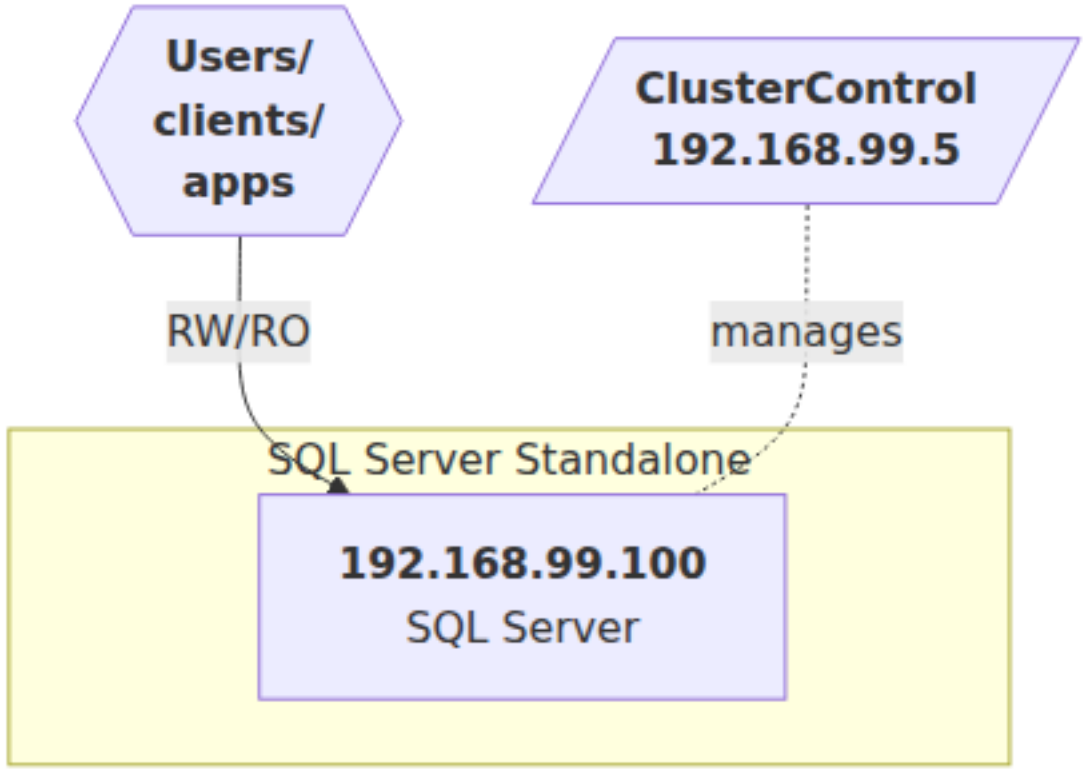
### 1.19.1 Prerequisites

Before proceeding, ensure you have:

- ClusterControl installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least two hosts (bare-metal or virtual machines):
  - One for the ClusterControl server.
  - One for the MySQL/MariaDB server.
- SSH access to all servers.
- Internet access on the database host to install required packages.
- Network Time Protocol (NTP) is configured and running on both hosts to keep their clocks synchronized.

### 1.19.2 Architecture

Below is a simplified diagram of the final architecture for a standalone deployment:




No load balancers or replicas are present here. You have a single MySQL/MariaDB server that directly serves both read and write requests from your applications.

---

### 1.19.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

Copy the public key to your standalone database node (replace 192.168.99.101 with your node's IP/hostname):

```
```bash
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101
```
```

If the target node uses a custom SSH key or port, you can add options:

```
```bash
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.101
```
```

2. Test passwordless SSH from the ClusterControl server:

```
ssh root@192.168.99.101 "ls /root"
```

Ensure there is no password prompt. If the command returns directory contents, you're set.

### 1.19.4 Step 2: Deploy a new cluster (single node)

1. Open a web browser and go to the ClusterControl server's IP or hostname.
2. On the ClusterControl dashboard, click ***Deploy a cluster (top-right) → Create a database cluster***. This opens the **Deploy a cluster** panel.
3. Choose "MySQL Replication" from the **Database** dropdown, choose "MariaDB" or your preferred vendor from the **Vendor** dropdown and also the database major version from the **Version** dropdown. Click **Continue**.
4. In the **Deploy MySQL Replication** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: For example, `Standalone MySQL`.
- **Tags**: (Optional) e.g., `standalone`, `production`, `dc1`.

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: `22`
- **SSH sudo password**: (leave blank if you rely on key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **Server port**: `3306` (default)
- **Server data directory**: `/var/lib/mysql` (default)
- **Admin/Root user**: `root`
- **Admin/Root password**: Password to be assigned to the database `root` user
- **Repository**: Use vendor repositories (default)
- **Version**: Choose the minor version for your selected vendor
- **Configuration template**: A default `my.cnf` or `my.cnf.replication`
- **Enable SSL encryption**: Optional (default off if not required)

#### (4) Add node

- **Standalone node**: Enter `192.168.99.101` and press ++enter++.
- Wait for the node to turn **green**. If an error (red) appears, check the logs in Activity center for details.

#### (5) Preview

- Review your configuration. You can go back and adjust if needed.

5. Click **Finish** to start deployment.
6. ClusterControl will install and configure MySQL on 192.168.99.101. You can track progress in the **Activity center**. After a few minutes, your new single-node cluster will appear on the **Home** page.

### 1.19.5 Step 3: Monitor your cluster

Once deployed, you'll see:

- Cluster health: The **Home** page provides the cluster state, even though it's a single node, it's treated as a "cluster" of size one.
- Node health: Hover on the honeycomb diagram or check the Nodes tab. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → {cluster} → Dashboards***.
- Recent alarms: Any triggered alarms will appear if there are configuration or resource issues.
- Automatic recovery status: If enabled, ClusterControl can attempt to restart a crashed MySQL server automatically.
- Topology viewer: You'll see a very simple topology with only one node.

### 1.19.6 Step 4: Import data

1. You can import data into the standalone server in a variety of ways:

#### Using MySQL dump

```
mysql -u root -p -h database_name < dumpfile.sql
```

#### Using `LOAD DATA INFILE`

```
LOAD DATA INFILE '/path/to/file.csv'
INTO TABLE table_name
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. If you have a large dataset (hundreds of GB), ensure you have enough disk space and memory.

### 1.19.7 Step 5: (Optional) Deploy a new load balancer

A load balancer (e.g., ProxySQL, HAProxy) is not necessary for a single-node environment. Typically, it is used for multi-node or high availability setups to split traffic or handle failover. You can skip this entirely if you only have one node.

**Tip**

If you plan to add replicas or switch to a high-availability cluster later, a load balancer can be introduced at that point. See *Create Load Balancers* for more info.

### 1.19.8 Step 6: Connect to the database server

Your application or client will connect directly to the standalone server:

- Host: 192.168.99.101
- Port: 3306
- User/Password: The credentials specified during the deployment

### 1.19.9 Step 7: Enable automatic backups

Even for a single node, regular backups are critical. You can create scheduled backups and optionally enable point-in-time recovery (PITR) by keeping binary logs.

1. Go to ***ClusterControl GUI → {cluster} → Backups***.
2. Click ***Create Backup → Schedule a Backup***. The Create a backup schedule wizard will open and proceed to configure your backup as below:

#### (1) Configuration

- **Schedule name**: `Daily backup - mysqldump`
- **Backup method**: `mysqldump` (or `xtrabackup`/`mariabackup` for physical)
- **Dump type**: `Complete PITR-compatible`
- **Upload backup to cloud**: Off (default)

#### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: `6`
- **Enable encryption**: (optional)
- **Verify backup**: Off (default)
- **Retention**: On (default)
- **Failover backup if node is down**: On (default)

#### (3) Storage

- **Store on controller**: (default)
- **Storage directory**: `/root/backups` (default)
- **Backup subdirectory**: `BACKUP-%I` (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at `02:00`
- **Timezone**: your local timezone

#### (5) Preview

- Verify all settings. You can go back to adjust if needed.

3. Click **Create** to schedule it. The backup schedule will appear under the **Schedules** tab.

ClusterControl will now automatically perform your backups. You can also enable backup verification, encryption, or point-in-time recovery under the same Backups section.

### 1.19.10 Step 8: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. ***ClusterControl GUI → {cluster} → Settings → Email Notifications***.
2. Select a User group (your group) from the **User group** dropdown.
3. Select your email address to from the **Users in the selected group** dropdown.
4. Click ***Enable***.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures timely notifications when something goes wrong.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.19.11 Step 9: Manage your node

ClusterControl offers a range of management features even for a standalone setup:

- Database upgrades and patching: Perform minor version upgrade to keep your cluster up to date. This feature is available at ***ClusterControl GUI → {cluster} → Actions → Upgrades***.
- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → {cluster} → Nodes → Actions***.
- ProxySQL management: Perform ProxySQL management related to query rules, query cache, proxy user management, export configuration, synchronize instances (in cluster mode), restart service and reboot host. These features are available at ***ClusterControl GUI → {cluster} → Nodes → Actions*** and ***ClusterControl GUI → {cluster} → Nodes → ProxySQL***.

- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → {cluster} → Manage → Configuration***.
- Database user management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → {cluster} → Manage → DB Users***.
- Backup management: Create, schedule, restore, perform point-in-time recovery, upload backup to cloud, verify an existing backup, and set retention period. These features are available at **ClusterControl GUI → {cluster} → Backups → Actions_** and **ClusterControl GUI → {cluster} → Backups → More_**.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → Actions → SSH Console***.

### 1.19.12 Step 10: Scale your cluster

Scaling your standalone database server into a multi-node setup is straightforward with ClusterControl. You can add new replica nodes (for read scaling or backup offloading), configure a load balancer for improved performance and failover, or even switch to a synchronous replication topology (like Galera) for high availability, all from within the ClusterControl GUI. This means you don't have to redeploy your database from scratch. Instead, you can gradually expand as your workloads and traffic demands increase.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → {cluster} → Actions → Add new → Replica node → Create a replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 3306
- **Data directory**: /var/lib/mysql
- **Configuration template**: my.cnf.repl80
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Advanced settings

- **Rebuild from a backup**: Off (default)
- **Include in load balancer set (if exists)**: On
- **Delay the replica node**: Off (default)

#### (3) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press "Enter".

#### (4) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

---

### 1.19.13 Conclusion

Congratulations! You've deployed, monitored, and managed a standalone MySQL/MariaDB server using ClusterControl. Although you now have only one node, you can effortlessly evolve this environment to replication or high availability in the future. Remember to:

- Keep your backups current and tested.
- Monitor performance and resource usage.
- Secure your node by restricting access and using strong passwords.
- Expand to multi-node topologies if your needs grow.

## 1.20 MySQL Replication (MariaDB, Percona, Oracle)

This guide will lead you through the process of setting up your first MySQL replication topology (primary-replica), complete with ProxySQL load balancers and Keepalived for high availability, using ClusterControl. By following this tutorial, you'll have a fully functional database cluster that you can monitor, manage, and scale.

MySQL Replication is a well-established feature that enables data from a primary (also known as a source) MySQL server to be copied automatically to one or more replicas. This allows you to scale out read workloads, offload backups and reporting, and provide basic high availability with manual or automatic failover strategies.

This tutorial applies to the following MySQL distributions:

- MariaDB Server
- Percona Server for MySQL
- Percona Pro Server for MySQL
- Oracle MySQL Community Server

**See also**

- MySQL Replication (Oracle) documentation
- MariaDB Replication documentation
- Percona Server for MySQL documentation
- Percona Distribution for MySQL documentation

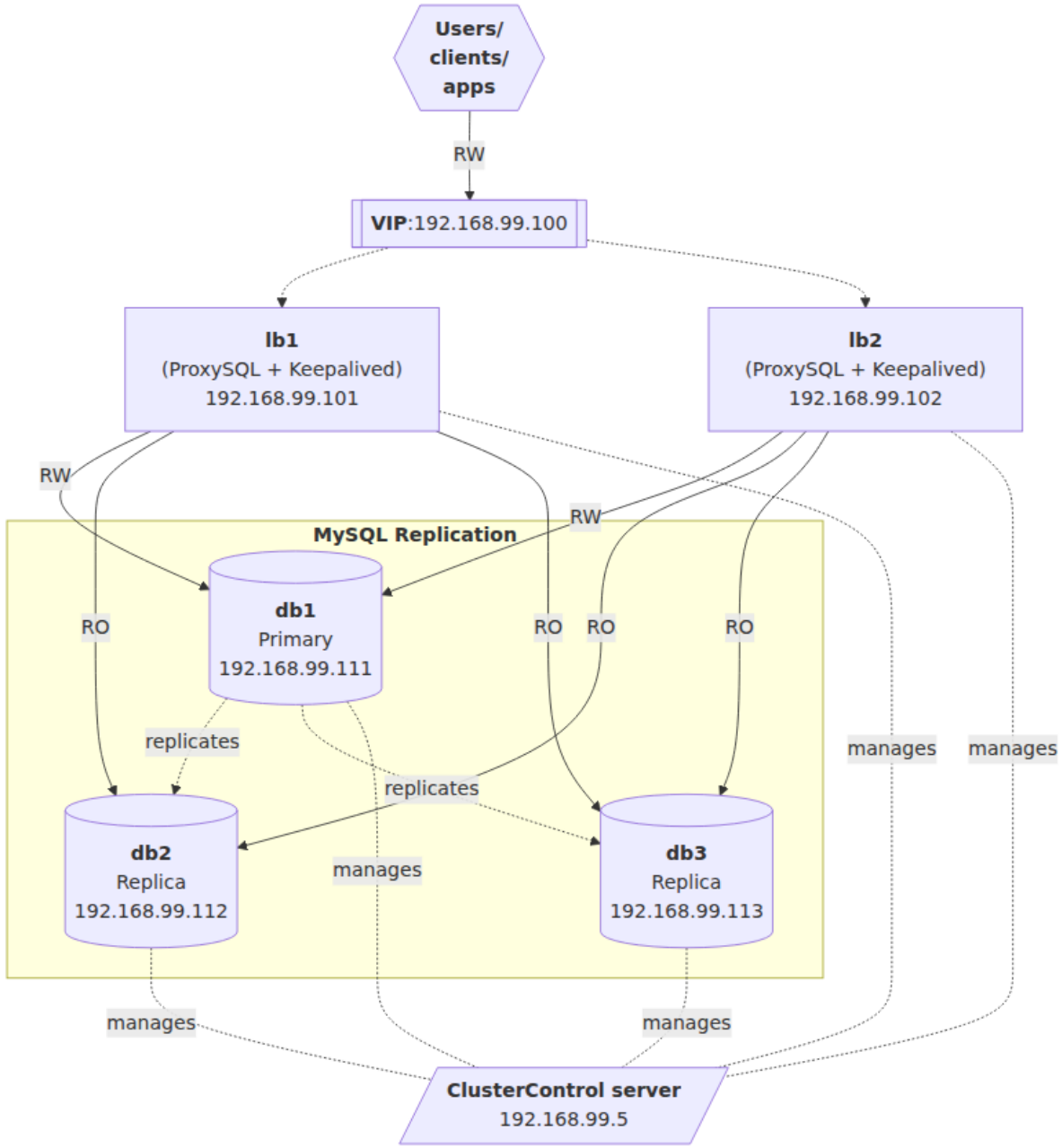
### 1.20.1 Prerequisites

Before proceeding, ensure the following requirements are met:

- ClusterControl is installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least four hosts (bare-metal or virtual machines) with a supported OS:
  - A host for the primary MySQL server.
  - Two hosts for the replica MySQL servers.
  - Two hosts for the load balancer (ProxySQL).
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- Network Time Protocol (NTP) is configured and running on both hosts to keep their clocks synchronized.

### 1.20.2 Architecture

Below is a simplified diagram of the final architecture:




### 1.20.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes (specify the password if prompted):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101  # lb1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # lb2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.111  # db1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.112  # db2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.113  # db3
```

If the target node only requires a key-based authentication, you can use the `-o` flag to customize the SSH option:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.101  # lb1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.102  # lb2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.111  # db1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.112  # db2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.113  # db3
```

3. Perform some tests and make sure you can execute the following commands from the ClusterControl server without any password prompt:

```
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.111 "ls /root"
ssh root@192.168.99.112 "ls /root"
ssh root@192.168.99.113 "ls /root"
```

### 1.20.4 Step 2: Deploy a new cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on the **Deploy a cluster** button in the top-right corner and choose **Create a database cluster**. This will open the **Deploy a cluster** panel.
3. Choose "MySQL Replication" from the **Database** dropdown, choose "Oracle" or your preferred vendor from the **Vendor** dropdown and also the database major version from the **Version** dropdown. Click **Continue**.
4. In the **Deploy MySQL Replication** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g, My First Replication Cluster).
- **Tags**: Enter one or more tags separate by pressing "Enter" (e.g, production, replication, dc1).

#### (2) SSH configuration

- **SSH user**: root
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave it blank)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install sofware**: On (default)
- **Disable firewall**: Checked (default)

- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **Server port**: 3306 (default)
- **Server data directory**: `/var/lib/mysql` (default)
- **Admin/Root user**: root
- **Admin/Root password**: Enter the password for the database admin/root
- **Repository**: Use vendor repositories (default)
- **Version**: Choose a database minor version.
- **Configurion template**: my.cnf.replication (default)
- **Enable SSL encryption**: On (default)

#### (4) Add nodes

- **Galera node**:
  - Specify "192.168.99.111" and press ++enter++, followed by "192.168.99.112" and press ++enter++, followed by "192.168.99.113" and press ++enter++.
  - Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
  - You can only proceed to the next step if all of the specified nodes are reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. Click **Finish** to start the deployment.
6. ClusterControl will provision your MySQL Replication cluster. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the cluster will appear in the **Home** page.

### 1.20.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → {cluster} → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.

- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state.

### 1.20.6 Step 4: Import data

When importing data in a primary-replica (asynchronous) replication setup, you must import the data on the primary server, and it will replicate to all replicas.

1. Import data on the primary (db1). Depending on your data format, you can use different utilities:

#### Using MySQL dump

```
mysql -u root -p -h database_name < dumpfile.sql
```

#### Using `LOAD DATA INFILE`

```
LOAD DATA INFILE '/path/to/file.csv'
INTO TABLE table_name
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. Check the cluster status in ClusterControl. Go to ***ClusterControl GUI → Clusters → {cluster} → Nodes → Topology***. Ensure db2 and db3 are both green (running) and in sync with db1.
3. Once your data import finishes and replicas catch up, you can proceed with normal read/write operations on the primary, and replicas shall stay up to date.

### 1.20.7 Step 5: Deploy load balancers

Using ProxySQL in front of a MySQL replication allows read-write splitting, enhances performance, scalability, and high availability by providing intelligent query routing, load balancing, connection pooling, failover handling, and security. It allows you to leverage the power of cluster by managing traffic efficiently while ensuring the application remains connected even during node failures or maintenance events.

1. Go to ***ClusterControl GUI → Clusters → {cluster} → Actions → Add new → Load balancer → Create balancer → ProxySQL → Continue***.
2. In the **Create a load balancer** wizard, configure the ProxySQL as below:

#### (1) Where to install

- **Version**: 2.x
- **Server address**: Specify “192.168.99.101” and press ++enter++. Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
- **Admin port**: 6032 (default)
- **Listening port**: 6033 (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)
- **Import configuration**: Unchecked (default)
- **Use native clustering**: Unchecked (default) (1) { .annotate }

1. Check this box if you want to deploy two or more ProxySQL instances. Or, if you plan to add more ProxySQL instance in the future.

### (2) Configuration

- **Administration user**: proxysql-admin (default)
- **Administration password**: Specify the password for ProxySQL admin user. ClusterControl will create this user.
- **Monitor user**: proxysql-monitor (default)
- **Monitor password**: Specify the password for ProxySQL monitor user. ClusterControl will create this user.
- **Create new user**: Selected
- **Database username**: Specify the database username that will be load balanced via ProxySQL. ClusterControl will create this user.
- **Database password**: Specify the database password for the database username above.
- **Database name**: Specify the database and/or table to be granted for this database user. If you want to grant this user to all tables for database example, specify `example.*`.
- **MySQL privilege(s)**: ALL PRIVILEGES

### (3) Server instances

- **192.168.99.111:3306 (Primary)**: On (default)
- **192.168.99.112:3306 (Primary)**: On (default)
- **192.168.99.113:3306 (Primary)**: On (default)
- Leave the **Max replication lag**, **Max connection** and **Weight** as default.
- **Are you using implicit transactions?**: Off (default) (1) { .annotate }

    1. Toggle on if the application relies on `SET autocommit=0` to create a transaction for you.

### (4) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

3. To deploy the load balancer, click **Finish**.
4. ClusterControl will start the ProxySQL deployment job. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the load balancer instance will appear in the **Nodes** page.
5. Repeat the same steps above to add the secondary ProxySQL instance, 192.168.99.102 (lb2).

### 1.20.8 Step 6: Deploy virtual IP address

A Virtual IP address (VIP) allows your application to connect to a single endpoint (IP address), where the IP address floats between two or more load balancer hosts. Only the active node holds the VIP, while the backup node takes over the VIP if the active node becomes unavailable.

Before deploying the virtual IP address, ensure that you have completed *Step 5* and that both ProxySQL instances are listed under the **Nodes** section. Additionally, prepare an unused routable IP address to be used as the VIP.

1. Go to ***ClusterControl GUI → Clusters → {cluster} → Actions → Add new → Load balancer → Create balancer → Keepalived → Continue***.
2. In the **Create Keepalived service** wizard, configure as below:

#### (1) Configuration

- **Load balancer type**: ProxySQL
- **Keepalived**: Choose "192.168.99.101:6032 (ProxySQL)" and "192.168.99.102:6032 (ProxySQL)" from the dropdown. Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Host configuration

- **Virtual IP**: 192.168.99.100
- **Network interface**: Specify which network interface that the virtual IP address will be associated with. You can use the `ifconfig` or `ip` command to get the correct interface name. The default value is "eth0".

#### (3) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

3. To deploy the Keepalived, click **Finish**.
4. ClusterControl will start the deployment job. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the Keepalived instances will appear in the **Nodes** page.

### 1.20.9 Step 7: Connect to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- Connect to the virtual IP address 192.168.99.100 on port 6033. This is the recommended method for high availability. It provides a load-balanced MySQL connection via ProxySQL through the virtual IP address, which exists on only one of the load balancers at any given time. Use the database user credentials created in *Step 5*.
- Connect to any of the load balancer instances (192.168.99.101 or 192.168.99.102) on port 6033. This also provides a load-balanced MySQL connection via ProxySQL. Use the database user credentials created in *Step 5*.

- In a replication setup, only the primary node can serve read and write workloads, while replicas operate in read-only mode. Connect directly to the primary node (192.168.99.111) on port 3306 for read-write workloads, or connect to the replica nodes (192.168.99.112 or 192.168.99.113) on port 3306 for read-only workloads.

**Note**

Note With ProxySQL in the topology, it is recommended to create a new database user using ClusterControl via ***ClusterControl GUI → {cluster} → Manage → DB Users → Create DB User***. ClusterControl ensures that the database user is created across all database nodes as well as ProxySQL nodes. If you create the database user manually, you must also create the same user on both ProxySQL hosts.

### 1.20.10 Step 8: Enable automatic backups

To protect your data, set up automatic backups. ClusterControl supports both logical and physical backups for MySQL replication. For a small dataset, a daily logical backup should be sufficient since by default, ClusterControl configures every database server with binary logs enabled that can be used for point-in-time recovery.

1. Go to ***ClusterControl GUI → {cluster} → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily backup - mysqldump
- **Cluster**: (leave as it is)
- **Backup host**: db3
- **Backup method**: mysqldump
- **Dump type**: Complete PITR-compatible
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Enable encryption**: On
- **Verify backup**: Off (default)
- **Retention**: On (default)
- **Use extended insert**: On (default)
- **Failover backup if node is down**: On

#### (3) Storage

- **Storage location**: Store on controller (default)
- **Storage directory**: /root/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. ClusterControl also support advanced backup features like partial backup, backup encryption, backup verification, backup throttling, back up to cloud and point-in-time recovery.

### 1.20.11 Step 9: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → {cluster} → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.20.12 Step 10: Manage your cluster and load balancers

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform minor version rolling upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → {cluster} → Actions → Upgrades***.
- Database node management: Perform actions such as start, stop, and restart nodes; reboot hosts; bootstrap the cluster; activate or deactivate read-only mode; resync nodes; change the primary source; and rebuild replicas. These features are available at ***ClusterControl GUI → {cluster} → Nodes → {node} → Actions***.
- ProxySQL management: Manage ProxySQL components such as query rules, query cache, proxy users, configuration export, instance synchronization (in cluster mode), service restarts, and host reboots. These features are available at ***ClusterControl GUI → {cluster} → Nodes → {node} → Actions*** and ***ClusterControl GUI → {cluster} → Nodes → ProxySQL***.
- Configuration management: Apply database configuration changes globally or to individual database nodes. This feature is available at ***ClusterControl GUI → {cluster} → Manage → Configuration***.
- Database user management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → {cluster} → Manage → DB Users***.
- Backup management: Create, schedule, and restore backups, perform point-in-time recovery, upload backups to the cloud, verify backups, and set retention policies. These features are available at **ClusterControl GUI → {cluster} → Backups → {backup} → Actions_** and **ClusterControl GUI → {cluster} → Backups → More_**.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from the ClusterControl GUI via the web-based SSH console. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → {node} → Actions → SSH Console***.

### 1.20.13 Step 11: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster and scaling your MySQL replication is straightforward. You can scale out the cluster with another database node or a read-only replica (delayed or real-time replication and asynchronous or semi-synchronous replication). Common use cases are to use the replica node for a dedicated read-only workloads, backups, reporting or analytics purposes.

Adding a new database node or replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node if you stream directly from the primary. You may opt to stream from an existing backup to minimize this risk.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → {cluster} → Actions → Add new → Replica node → Create a replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 3306
- **Data directory**: /var/lib/mysql
- **Configuration template**: my.cnf.repl80
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Advanced settings

- **Rebuild from a backup**: Off (default)
- **Include in load balancer set (if exists)**: On
- **Delay the replica node**: Off (default)

#### (3) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press ++enter++.

#### (4) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

---

### 1.20.14 Conclusion

Congratulations! You've successfully deployed, monitored, managed, and scaled your first MySQL replication cluster using ClusterControl. Explore additional features like failover management, advanced monitoring, backup management, and more to optimize your database infrastructure for high availability and performance.

## 1.21 MySQL Group Replication

In this guide, we will walk you through setting up your first database cluster running on MySQL Group Replication with a HAProxy load balancer using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

MySQL Group Replication, an open-source plugin by Oracle, offers built-in high availability and fault tolerance for MySQL (version 5.7.17 and later). This replication solution enables multiple MySQL servers to form a clustered environment, ensuring data redundancy and automatic failover.

Utilizing a Paxos-based consensus protocol, MySQL Group Replication guarantees that transactions are replicated and confirmed by a majority of servers before commitment. Key features include automatic failover, distributed recovery and seamless addition of new nodes. This makes it a robust choice for mission-critical MySQL deployments demanding high uptime and fault tolerance.

**See also**

- MySQL Group Replication documentation

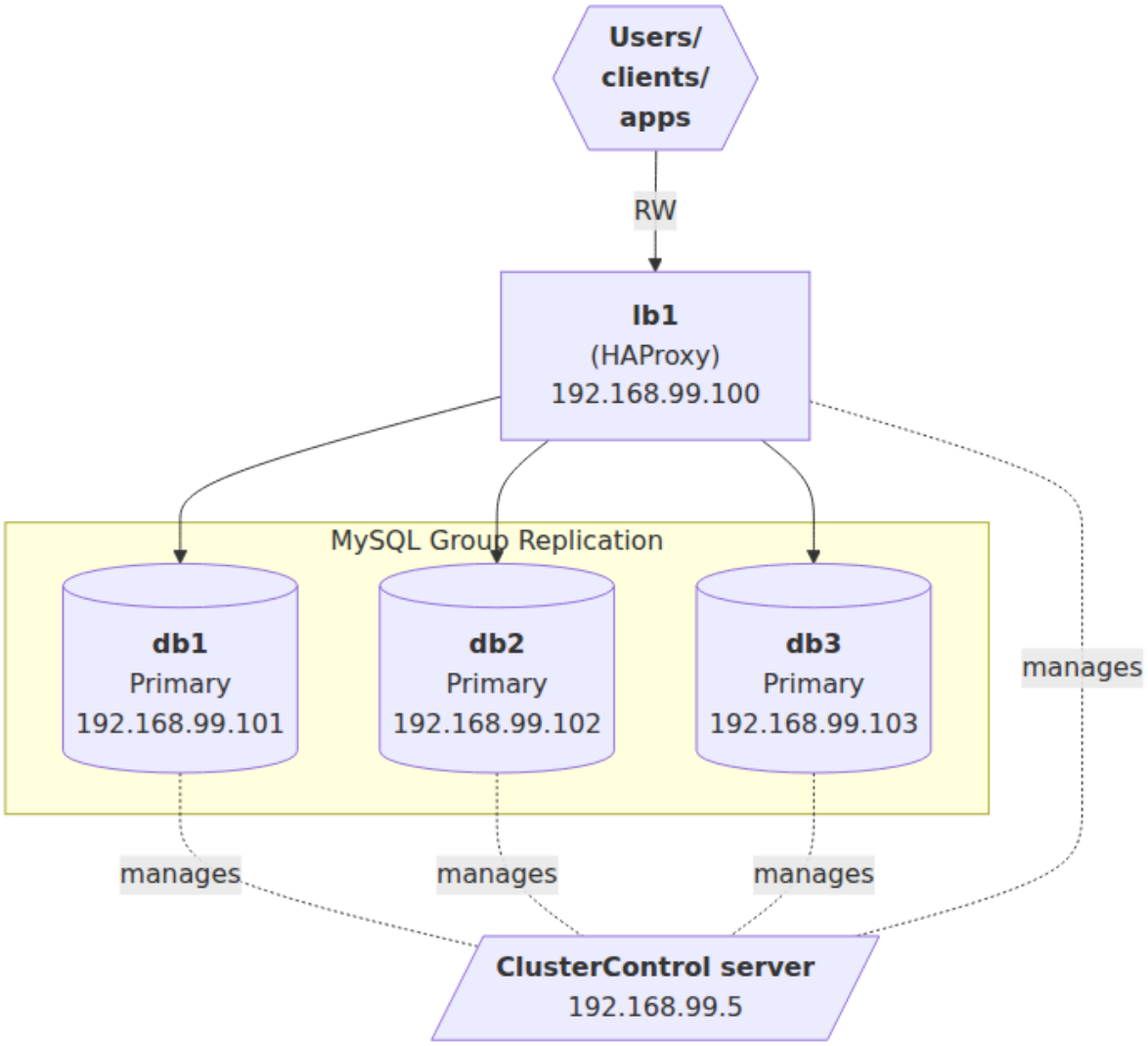
### 1.21.1 Prerequisites

Before you start, ensure that you have the following:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quick-start* or use the *Installer Script*.
- You need at least four hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*. One host is for database load balancer and another three hosts are for MySQL Group Replication nodes.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.21.2 Architecture

Our final system architecture will look like this:




### 1.21.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes (specify the password if prompted):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.100  # lb1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db3
```

If the target node only requires a key-based authentication, you can use the `-o` flag to customize the SSH option:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.100  # lb1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.101  # db1
```

(continues on next page)

(continued from previous page)

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.103  # db3
```

3. Perform some tests and make sure you can execute the following commands from the ClusterControl server without any password prompt:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.103 "ls /root"
```

### 1.21.4 Step 2: Deploy a new cluster

1. Open your browser and go to the IP address or domain name where your ClusterControl is installed.
2. From the ClusterControl dashboard, click on the **Deploy a cluster** button in the top-right corner and choose **Create a database cluster**. This will open the **Deploy cluster** wizard.
3. Choose "MySQL Group Replication" from the **Database** dropdown, choose "Oracle" from the **Vendor** dropdown and also the database major version from the **Version** dropdown. Click **Continue**.
4. In the **Deploy MySQL Group Replication** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g, My First Group Replication).
- **Tags**: Enter one or more tags separate by pressing "Enter" (e.g, production, group replication, dc1).

#### (2) SSH configuration

- **SSH user**: root
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave it blank)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install sofware**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **Server port**: 3306 (default)
- **Server data directory**: `/var/lib/mysql` (default)
- **Admin/Root user**: root
- **Admin/Root password**: Enter the password for the database admin/root

- **Repository**: Use vendor repositories (default)
- **Version**: Choose a database minor version.
- **Configurion template**: my.cnf.grouprepl (default)
- **Enable SSL encryption**: On (default)

#### (4) Add nodes

- **MySQL node**:
  - Specify "192.168.99.101" and press "Enter", followed by "192.168.99.102" and press "Enter", followed by ""192.168.99.103" and press ++enter++.
  - Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
  - You can only proceed to the next step if all of the specified nodes are reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. To deploy the cluster, click **Finish**.
6. ClusterControl will now start provisioning the MySQL Group Replication. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the cluster will appear in the **Home** page.

### 1.21.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.

### 1.21.6 Step 4: Import data

To import data into a MySQL Group Replication, you need to take into account that MySQL Group Replication uses synchronous replication and all changes are replicated across all nodes. Performing restore using `mysqldump` or `LOAD DATA INFILE` will have a significant impact to the cluster and it is a slow operation since all inserts will be cerfified by the group replication. To avoid this, you should perform the restore on one of the database nodes.

Here are the steps to safely import data into a MySQL Group Replication using ClusterControl:

1. Stop two out of three nodes. Firstly, stop db2. Go to ***ClusterControl GUI → Nodes → locate db2 (1) → Actions → Stop node → Stop***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.102`, `Port: 3306`, `Status: Operational`, `Type: MySQL Group Replication` and `Role: Primary` from the nodes list.
2. Next, stop db3. Go to ***ClusterControl GUI → Nodes → locate db3 (1) → Actions → Stop node → Stop***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.103`, `Port: 3306`, `Status: Operational`, `Type: MySQL Group Replication` and `Role: Primary` from the nodes list.
3. At this point, the database should be running as a single-node cluster with only db1 is running. Depending on the format of your data, you can use different MySQL utilities to import data:

#### Using MySQL dump

If you are importing from a dump file, transfer the dump file to db1 and use the following command to restore:

```
mysql -u root -p -h database_name < dumpfile.sql
```

#### Using `LOAD DATA INFILE`

If you are importing from a CSV or similar file format, transfer the file to db1 and run the following SQL statement from the MySQL console:

```
LOAD DATA INFILE '/path/to/file.csv'
INTO TABLE table_name
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

4. Once imported, start db2 using **Start Node**. This will automatically sync the transaction write-set through Group Communication System (GCS) in MySQL Group Replication. Go to ***ClusterControl GUI → Nodes → locate db2 (1) → Actions → Start Node***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.102`, `Port: 3306`, `Status: Shut Down`, `Type: MySQL Group Replication` and `Role: Primary` from the nodes list.
5. Start db3 using **Resync Node**. This will automatically sync the transaction write-set through Group Communication System (GCS) in MySQL Group Replication. Go to ***ClusterControl GUI → Nodes → locate db3 (1) → Actions → Start Node***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.103`, `Port: 3306`, `Status: Shut Down`, `Type: MySQL Group Replication` and `Role: Primary` from the nodes list.

6. Once all nodes up and running, verify that all nodes should be green from the **Home** page. The import exercise is complete.

### 1.21.7 Step 5: Deploy a new load balancer

Using HAProxy in front of a MySQL Group Replication enhances performance, scalability, and high availability by providing intelligent query routing, load balancing, connection pooling, failover handling, and security. It allows you to fully leverage the power of cluster by managing traffic efficiently while ensuring the application remains connected even during node failures or maintenance events.

1. Go to ***ClusterControl GUI → Clusters → locate the cluster → Actions → Add new → Load balancer → Create balancer → HAProxy → Continue***.
2. In the **Create a load balancer** wizard, configure HAProxy as below:

#### (1) Where to install

- **Server address**: `192.168.99.100`. Press "Enter" and wait until it turns green.
- **Policy**: leastconn (default) (1) { .annotate }
    1. You can choose the policy that suits your requirement.
- **Listen port (read/write)**: 5433 (default)
- **Listen port (read-only)**: 5434 (default)
- **Install for read/write splitting (master-slave replication)**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Installation settings

- **Overwrite existing /usr/local/sbin/mysqlchk_rw_split on targets**: On (default)

#### (3) Advanced settings

- **Stats socket**: /var/run/haproxy.socket (default)
- **Admin port**: 9600 (default)
- **Admin user**: admin
- **Admin password**: Enter the password for the Admin user
- **Backend name (RW)**: haproxy_5433_rw (default)
- **Backend name (RO)**: haproxy_5434_ro (default)
- **Timeout server (in seconds)**: 10800 (default)
- **Timeout client (in seconds)**: 10800 (default)
- **Max connections frontend**: 8192 (default)
- **Max connections backend (per instance)**: 1024
- **xinted allow connections from**: 0.0.0.0/0 (default)

#### (4) Server instances in the load balancer

- **192.168.99.101:3306 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.101]
- **192.168.99.102:3306 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.102]
- **192.168.99.103:3306 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.103]

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

3. To deploy the load balancer, click **Finish**.
4. ClusterControl will now start provisioning the HAProxy instance. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the load balancer instance will appear in the **Nodes** page.

!!! Tips

```
For production environment, it is recommended to have at least two load␣
↪balancers to eliminate single-point of failure. You can deploy another␣
↪HAProxy instance and tie them together with a virtual IP address. See␣
↪[Create Load␣
↪Balancers](../../../../../user-guide/deployment/create-load-balancer.md).
```

### 1.21.8 Step 6: Connecting to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- Load balancer instance, 192.168.99.100 on port 3307. This is a load-balanced MySQL connection via HAProxy. Use the database user credentials that you created at *Step 5*.
- If there are two or more load balancers, tie them together with a virtual IP address provided by Keepalived. Use the virtual IP address in your connection string instead of the load balancers' primary IP address.

### 1.21.9 Step 7: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For MySQL Group Replication, ClusterControl supports both logical and physical backups. For a small dataset, a daily logical backup should be sufficient since by default, ClusterControl configures every database server with binary logs enabled that can be used for point-in-time recovery.

1. Go to ***ClusterControl GUI → choose the cluster → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily backup - mysqldump
- **Cluster**: (leave as it is)
- **Backup host**: db3
- **Backup method**: mysqldump
- **Dump type**: Complete PITR-compatible
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Enable encryption**: On
- **Verify backup**: Off (default)
- **Retention**: On (default)
- **Use extended insert**: On (default)
- **Failover backup if node is down**: On

#### (3) Storage

- **Storage location**: Store on controller (default)
- **Storage directory**: /root/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. ClusterControl also support advanced backup features like partial backup, backup encryption, backup verification, backup throttling, back up to cloud and point-in-time recovery. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size and perform restoration if necessary.

### 1.21.10 Step 8: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.21.11 Step 9: Manage your cluster and load balancer

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform rolling upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → choose the cluster → Actions → Upgrades***.

- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- HAProxy management: Perform HAProxy management related disable nodes, see the detailed status, restart service and reboot host. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions*** and ***ClusterControl GUI → choose the cluster → Nodes → HAProxy***.
- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Database user management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → DB Users***.
- Backup management: Create, schedule, restore, perform point-in-time recovery, upload backup to cloud, verify an existing backup, and set retention period. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions**_ and **ClusterControl GUI → choose the cluster → Backups → More**_.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.21.12 Step 10: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. For MySQL Group Replication, you can scale out the cluster with another database node.

Adding a new database node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node if you stream directly from the primary. You may opt to stream from an existing backup to minimize this risk.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new node, go to ***ClusterControl GUI → choose the cluster → Actions → Add new → Node → Create a database node***.
2. In the **Create a database node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 3306
- **Data directory**: /var/lib/mysql
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Advanced settings

- **Rebuild from a backup**: Off (default)
- **Include in load balancer set (if exists)**: On

#### (3) Add node

- **Node**: Specify the IP address or hostname of the node that you want to add and press "Enter".

#### (4) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

### 1.21.13 Conclusion

Congratulations! You've successfully deployed, monitored, managed and scaled your first MySQL Group Replication using ClusterControl. From here, you can explore additional features such as failover management, advanced monitoring, advanced backup management, database restore, cluster cloning and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant.

## 1.22 Percona XtraDB Cluster

In this guide, we will walk you through setting up your first database cluster running on Percona XtraDB Cluster with a ProxySQL load balancer using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

Percona XtraDB Cluster (PXC) is an open-source, high-availability solution for MySQL, based on the Galera replication technology. It provides a synchronous multi-master replication system where all nodes in the cluster hold the same data, ensuring strong consistency and zero data loss. PXC is designed to deliver automatic failover, seamless scalability, and load balancing across multiple nodes, making it highly resilient to node failures. It includes features like automatic node provisioning, parallel replication, and integrated backup support, making it a robust solution for mission-critical MySQL environments that require high uptime and fault tolerance.

**See also**

- Percona XtraDB Cluster product page
- Percona XtraDB Cluster documentation

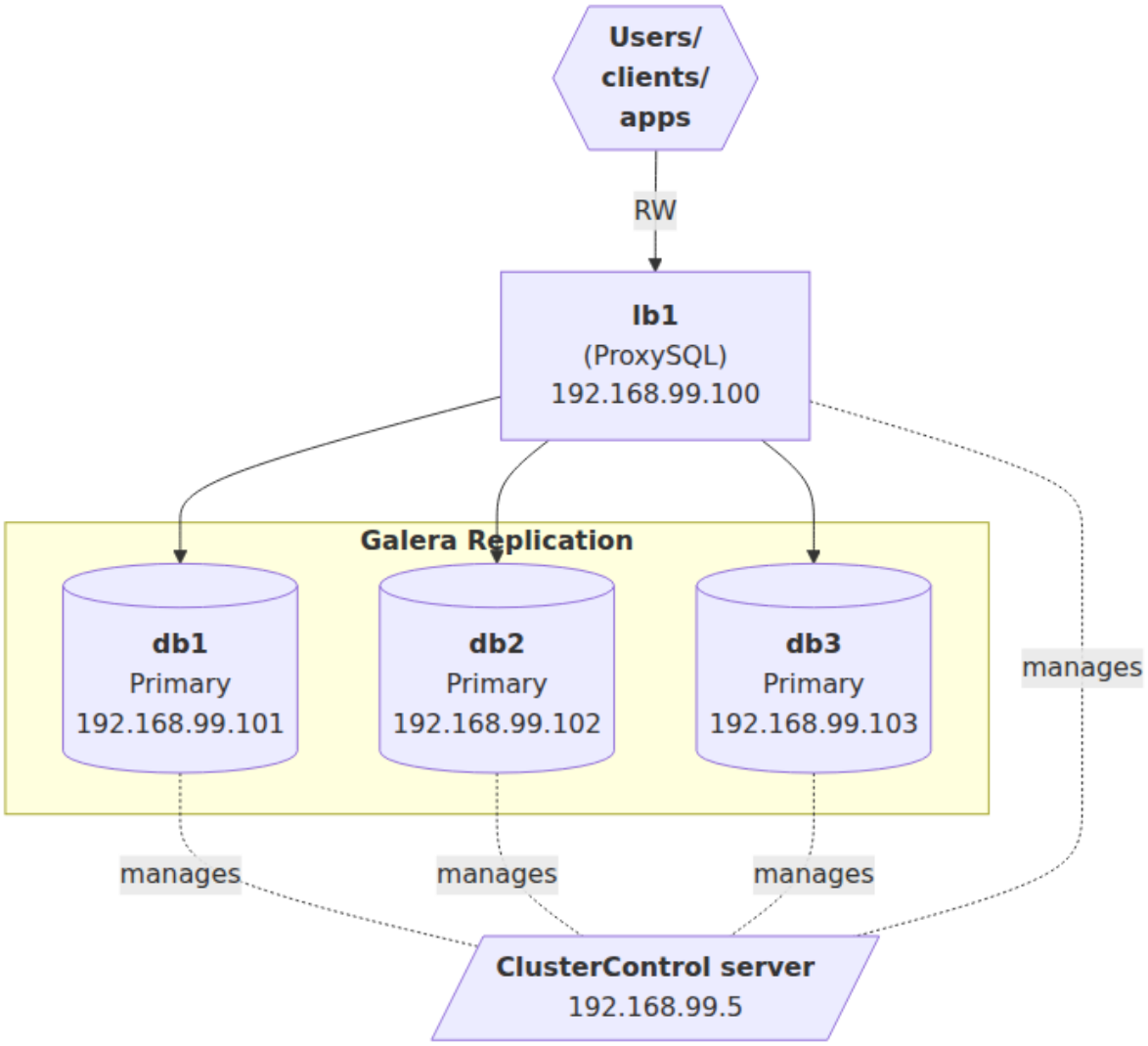
### 1.22.1 Prerequisites

Before you start, ensure that you have the following:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quickstart* or use the *Installer Script*.
- You need at least four hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*. One host is for database load balancer and another three hosts are for Percona XtraDB Cluster.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.22.2 Architecture

Our final system architecture will look like this:




### 1.22.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes (specify the password if prompted):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.100  # lb1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db3
```

If the target node only requires a key-based authentication, you can use the `-o` flag to customize the SSH option:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.100  # lb1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
```

(continues on next page)

(continued from previous page)

```
↪root/myprivatekey.pem' root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.103  # db3
```

3. Perform some tests and make sure you can execute the following commands from the ClusterControl server without any password prompt:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.103 "ls /root"
```

### 1.22.4 Step 2: Deploy a new cluster

1. Open your browser and go to the IP address or domain name where your ClusterControl is installed.
2. From the ClusterControl dashboard, click on the **Deploy a cluster** button in the top-right corner and choose **Create a database cluster**. This will open the **Deploy cluster** wizard.
3. Choose "MySQL Galera" from the **Database** dropdown, choose "Percona XtraDB" from the **Vendor** dropdown and also the database major version from the **Version** dropdown. Click **Continue**.
4. In the **Deploy MySQL Galera Cluster** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g, My First Galera Cluster).
- **Tags**: Enter one or more tags separate by pressing "Enter" (e.g, production, galera, dc1).

#### (2) SSH configuration

- **SSH user**: root
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave it blank)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install sofware**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **Server port**: 3306 (default)
- **Server data directory**: `/var/lib/mysql` (default)
- **Admin/Root user**: root

- **Admin/Root password**: Enter the password for the database admin/root
- **Repository**: Use vendor repositories (default)
- **Version**: Choose a database minor version.
- **Configurion template**: my.cnf.80-pxc (default)
- **Enable SSL encryption**: On (default)

#### (4) Add nodes

- **Galera node**:
  - Specify “192.168.99.101” and press “Enter”, followed by “192.168.99.102” and press “Enter”, followed by ““192.168.99.103” and press ++enter++.
  - Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
  - You can only proceed to the next step if all of the specified nodes are reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. To deploy the cluster, click **Finish**.
6. ClusterControl will now start provisioning the Percona XtraDB Cluster. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the cluster will appear in the **Home** page.

### 1.22.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster’s status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → {cluster} → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → {cluster} → Nodes → Topology***.

### 1.22.6 Step 4: Import data

To import data into a Percona XtraDB Cluster (PXC), you need to take into account that PXC uses synchronous replication and all changes are replicated across all nodes. Performing restore using `mysqldump` or `LOAD DATA INFILE` will have a significant impact to the cluster and it is a slow operation since all inserts will be cerfified by Galera write-set replication. To avoid this, you should perform the restore on one of the database nodes.

Here are the steps to safely import data into a PXC cluster using ClusterControl:

1. Stop two out of three nodes. Firstly, stop db2. Go to ***ClusterControl GUI → Nodes → locate db2 (1) → Actions → Stop node → Stop***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.102`, `Port: 3306`, `Status: Operational`, `Type: XtraDB` and `Role: Primary` from the nodes list.
2. Next, stop db3. Go to ***ClusterControl GUI → Nodes → locate db3 (1) → Actions → Stop node → Stop***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.103`, `Port: 3306`, `Status: Operational`, `Type: XtraDB` and `Role: Primary` from the nodes list.
3. At this point, the database should be running as a single-node cluster with only db1 is running. Depending on the format of your data, you can use different MySQL utilities to import data:

#### Using MySQL dump

If you are importing from a dump file, transfer the dump file to db1 and use the following command to restore:

```
mysql -u root -p -h database_name < dumpfile.sql
```

#### Using `LOAD DATA INFILE`

If you are importing from a CSV or similar file format, transfer the file to db1 and run the following SQL statement from the MySQL console:

```
LOAD DATA INFILE '/path/to/file.csv'
INTO TABLE table_name
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

4. Once imported, trigger **Resync Node** for db2. This will initiate a full resynchronization using State Snapshot Transfer (SST), and the node will automatically start once the resync completes. Go to ***ClusterControl GUI → Nodes → db2 (1) → Actions → Resync Node → Resync***.
{ .annotate }
    1. Find the node with `Hostname: 192.168.99.102`, `Port: 3306`, `Status: Shut Down`, `Type: XtraDB` and `Role: Primary` from the nodes list.
5. Repeat the same step for db3. Trigger **Resync Node**, which will perform a full SST, and the node will automatically start after the process completes. Go to ***ClusterControl GUI → Nodes → db3 (1) → Actions → Resync Node → Resync***.
{ .annotate }
    1. Find the node with `Hostname: 192.168.99.103`, `Port: 3306`, `Status: Shut Down`, `Type: XtraDB` and `Role: Primary` from the nodes list.

6. After all nodes are resynced, verify from the **Home** page that all nodes are in green (**Operational**) status. The import process is now complete.

### 1.22.7 Step 5: Deploy a new load balancer

Using ProxySQL in front of a Percona XtraDB Cluster enhances performance, scalability, and high availability by providing intelligent query routing, load balancing, connection pooling, failover handling, and security. It allows you to fully leverage the power of cluster by managing traffic efficiently while ensuring the application remains connected even during node failures or maintenance events.

1. Go to ***ClusterControl GUI → Clusters → locate the cluster → Actions → Add new → Load balancer → Create balancer → ProxySQL → Continue***.
2. In the **Create a load balancer** wizard, configure the ProxySQL as below:

#### (1) Where to install

- **Version**: 2.x
- **Server address**: Specify "192.168.99.100" and press "Enter". Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
- **Admin port**: 6032 (default)
- **Listening port**: 6033 (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)
- **Import configuration**: Unchecked (default)
- **Use native clustering**: Unchecked (default) (1) { .annotate }

    1. Check this box if you want to deploy two or more ProxySQL instances. Or, if you plan to add more ProxySQL instance in the future.

#### (2) Configuration

- **Administration user**: proxysql-admin (default)
- **Administration password**: Specify the password for ProxySQL admin user. ClusterControl will create this user.
- **Monitor user**: proxysql-monitor (default)
- **Monitor password**: Specify the password for ProxySQL monitor user. ClusterControl will create this user.
- **Create new user**: Selected
- **Database username**: Specify the database username that will be load balanced via ProxySQL. ClusterControl will create this user.
- **Database password**: Specify the database password for the database username above.
- **Database name**: Specify the database and/or table to be granted for this database user. If you want to grant this user to all tables for database example, specify `example.*`.
- **MySQL privilege(s)**: ALL PRIVILEGES

#### (3) Server instances

- **192.168.99.101:3306 (Primary)**: On (default)
- **192.168.99.102:3306 (Primary)**: On (default)
- **192.168.99.103:3306 (Primary)**: On (default)
- Leave the **Max replication lag**, **Max connection** and **Weight** as default.
- **Are you using implicit transactions?**: Off (default) (1) { .annotate }

    1. Toggle on if the application relies on `SET autocommit=0` to create a transaction for you.

#### (4) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

3. To deploy the load balancer, click **Finish**.
4. ClusterControl will now start provisioning the ProxySQL instance. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the load balancer instance will appear in the **Nodes** page.

!!! Tips

```
For production environment, it is recommended to have at least two load␣
↪balancers to eliminate single-point of failure. You can deploy another␣
↪ProxySQL instance and tie them together with a virtual IP address. See␣
↪[Load␣
↪Balancers](../../../../../user-guide/deployment/create-load-balancer.md).
```

### 1.22.8 Step 6: Connect to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- Load balancer instance, 192.168.99.100 on port 6033. This is a load-balanced MySQL connection via ProxySQL. Use the database user credentials that you created at *Step 5*.
- If there are two or more load balancers, tie them together with a virtual IP address provided by Keepalived. Use the virtual IP address in your connection string instead of the load balancers' primary IP address.
- One of the database nodes on port 3306. However, you need to create a new database user using ClusterControl. A user created during the ProxySQL deployment stage is only allowed from the ProxySQL host.

### 1.22.9 Step 7: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For Percona XtraDB Cluster, ClusterControl supports both logical and physical backups. For a small dataset, a daily logical backup should be sufficient since by default, ClusterControl configures every database server with binary logs enabled that can be used for point-in-time recovery.

1. Go to ***ClusterControl GUI → {cluster} → Backups*** to open the backup overview page of the cluster.

2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

### (1) Configuration

- **Schedule name**: Daily backup - mysqldump
- **Cluster**: (leave as it is)
- **Backup host**: db3
- **Backup method**: mysqldump
- **Dump type**: Complete PITR-compatible
- **Upload backup to cloud**: Off

### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Enable encryption**: On
- **Verify backup**: Off (default)
- **Retention**: On (default)
- **Use extended insert**: On (default)
- **Failover backup if node is down**: On

### (3) Storage

- **Storage location**: Store on controller (default)
- **Storage directory**: /root/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. ClusterControl also support advanced backup features like partial backup, backup encryption, backup verification, backup throttling, back up to cloud and point-in-time recovery. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size and perform restoration if necessary.

### 1.22.10 Step 8: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → {cluster} → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.22.11 Step 9: Manage your cluster and load balancer

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform rolling upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → {cluster} → Actions → Upgrades***.
- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → {cluster} → Nodes → Actions***.
- ProxySQL management: Perform ProxySQL management related to query rules, query cache, proxy user management, export configuration, synchronize instances (in cluster mode), restart service and reboot host. These features are available at ***ClusterControl GUI → {cluster} → Nodes → Actions*** and ***ClusterControl GUI → {cluster} → Nodes → ProxySQL***.

- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → {cluster} → Manage → Configuration***.
- Database user management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → {cluster} → Manage → DB Users***.
- Backup management: Create, schedule, restore, perform point-in-time recovery, upload backup to cloud, verify an existing backup, and set retention period. These features are available at **ClusterControl GUI → {cluster} → Backups → Actions_** and **ClusterControl GUI → {cluster} → Backups → More_**.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → Actions → SSH Console***.

### 1.22.12 Step 10: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. For Percona XtraDB Cluster, you can scale out the cluster with another database node or a read-only replica (delayed or real-time replication and asynchronous or semi-synchronous replication). Common use cases are to use the replica node for a dedicated read-only workloads, backups, reporting or analytics purposes.

Adding a new database node or replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node if you stream directly from the primary. You may opt to stream from an existing backup to minimize this risk.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → {cluster} → Actions → Add new → Replica node → Create a replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 3306
- **Data directory**: /var/lib/mysql
- **Configuration template**: my.cnf.repl80
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Advanced settings

- **Rebuild from a backup**: Off (default)
- **Include in load balancer set (if exists)**: On
- **Delay the replica node**: Off (default)

#### (3) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press "Enter".

#### (4) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

---

### 1.22.13 Conclusion

Congratulations! You've successfully deployed, monitored, managed and scaled your first Percona XtraDB Cluster using ClusterControl. From here, you can explore additional features such as failover management, advanced monitoring, advanced backup management, database restore, cluster cloning and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant.

## 1.23 MariaDB Galera Cluster

This guide will lead you through the process of setting up your first MariaDB Galera Cluster, complete with HAProxy load balancers and Keepalived for high availability, using ClusterControl. By following this tutorial, you'll have a fully functional database cluster that you can effortlessly monitor, manage, and scale.

MariaDB Galera Cluster is an open-source solution designed for high availability in MariaDB environments. It leverages Galera replication technology to offer synchronous multi-master replication, ensuring data consistency across all nodes and preventing data loss. The cluster supports automatic failover, seamless scaling, and effective load balancing, providing resilience against node failures. With features such as automatic node provisioning, parallel replication, and built-in backup options, it's an excellent choice for critical MariaDB deployments that demand maximum uptime.

**See also**

- MariaDB Galera Cluster product page
- MariaDB Galera Cluster documentation
- HAProxy product page
- HAProxy documentation page

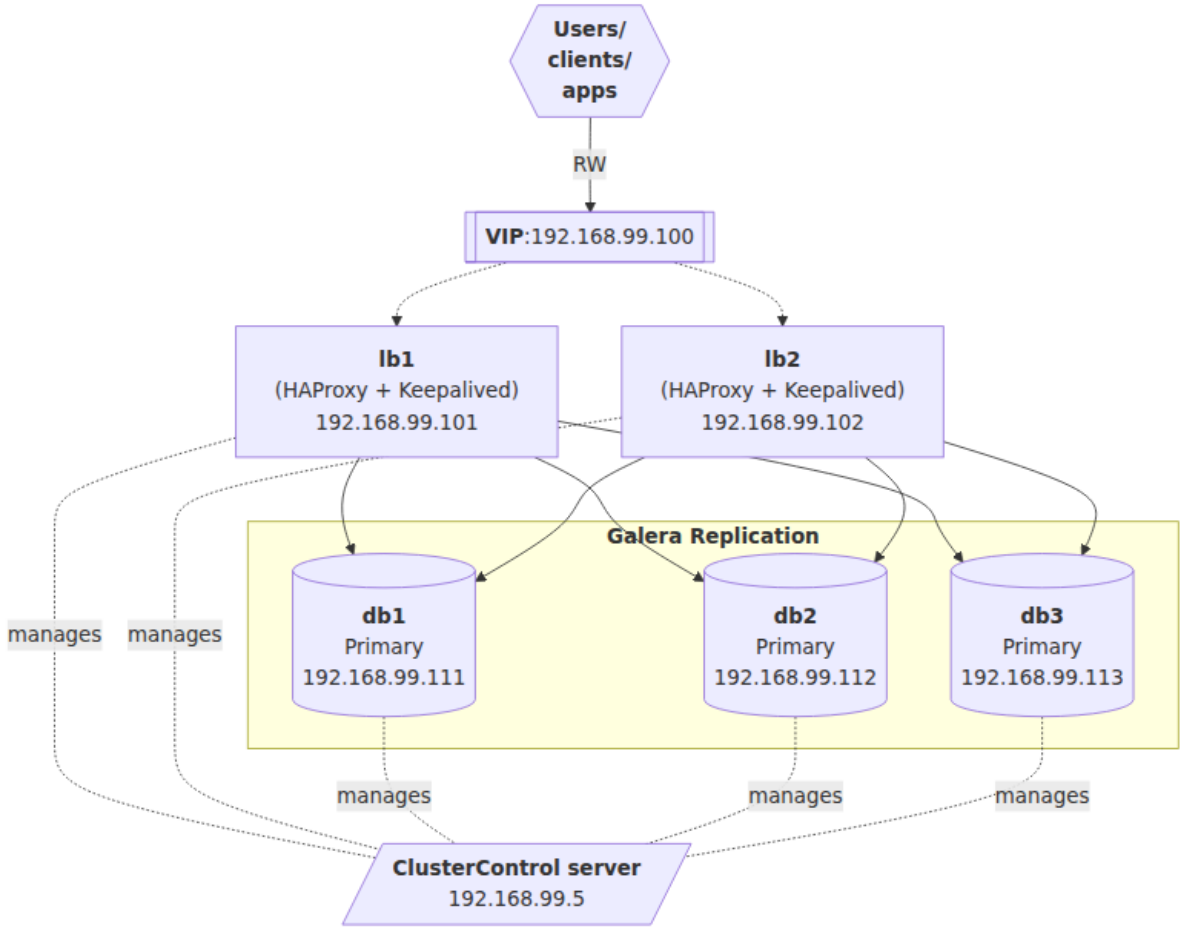
### 1.23.1 Prerequisites

Before proceeding, ensure the following requirements are met:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quick-start* or use the *Installer Script*.
- You need at least five hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*. Two hosts are for database load balancer and another three hosts are for MariaDB Galera Cluster.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.23.2 Architecture

Our final system architecture will look like this:



### 1.23.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes (specify the password if prompted):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101  # lb1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # lb2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.111  # db1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.112  # db2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.113  # db3
```

If the target node only requires a key-based authentication, you can use the `-o` flag to customize the SSH option:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.101  # lb1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.102  # lb2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.111  # db1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.112  # db2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.113  # db3
```

3. Perform some tests and make sure you can execute the following commands from the ClusterControl server without any password prompt:

```
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.111 "ls /root"
ssh root@192.168.99.112 "ls /root"
ssh root@192.168.99.113 "ls /root"
```

### 1.23.4 Step 2: Deploy a new cluster

1. Open your browser and go to the IP address or domain name where your ClusterControl is installed.
2. From the ClusterControl dashboard, click on the **Deploy a cluster** button in the top-right corner and choose **Create a database cluster**. This will open the **Deploy cluster** wizard.
3. Choose "MySQL Galera" from the **Database** dropdown, choose "MariaDB" from the **Vendor** dropdown and also the database major version from the **Version** dropdown. Click **Continue**.
4. In the **Deploy MySQL Galera Cluster** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g, My First Galera Cluster).
- **Tags**: Enter one or more tags separate by pressing "Enter" (e.g, production, galera, dc1).

#### (2) SSH configuration

- **SSH user**: root
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave it blank)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install sofware**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **Server port**: 3306 (default)
- **Server data directory**: `/var/lib/mysql` (default)
- **Admin/Root user**: root
- **Admin/Root password**: Enter the password for the database admin/root
- **Repository**: Use vendor repositories (default)
- **Version**: Choose a database minor version.
- **Configurion template**: my.cnf.mdb106+-galera (default)
- **Enable SSL encryption**: On (default)

#### (4) Add nodes

- **Galera node**:
  - Specify "192.168.99.111" and press ++enter++, followed by "192.168.99.112" and press ++enter++, followed by ""192.168.99.113" and press ++enter++.
  - Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
  - You can only proceed to the next step if all of the specified nodes are reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. To deploy the cluster, click **Finish**.

6. ClusterControl will now start provisioning the MariaDB Galera Cluster. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the cluster will appear in the **Home** page.

### 1.23.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → {cluster} → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state.

### 1.23.6 Step 4: Import data

When importing data into a MariaDB Galera Cluster, consider the impact of synchronous replication. For large datasets, it's recommended to import data on a single node and then resynchronize the cluster.

Here are the steps to safely import data into a MariaDB Galera cluster using ClusterControl:

1. Stop two out of three nodes. Firstly, stop db2. Go to ***ClusterControl GUI → Nodes → locate db2 (1) → Actions → Stop node → Stop***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.112`, `Port: 3306`, `Status: Operational`, `Type: MariaDB` and `Role: Primary` from the nodes list.
2. Next, stop db3. Go to ***ClusterControl GUI → Nodes → locate db3 (1) → Actions → Stop node → Stop***. { .annotate }
    1. Find the node with `Hostname: 192.168.99.113`, `Port: 3306`, `Status: Operational`, `Type: MariaDB` and `Role: Primary` from the nodes list.
3. At this point, the database should be running as a single-node cluster with only db1 is running. Depending on the format of your data, you can use different MySQL utilities to import data:

#### Using MySQL dump

If you are importing from a dump file, transfer the dump file to db1 and use the following command to restore:

```
mysql -u root -p -h database_name < dumpfile.sql
```

#### Using `LOAD DATA INFILE`

If you are importing from a CSV or similar file format, transfer the file to db1 and run the following SQL statement from the MySQL console:

```
LOAD DATA INFILE '/path/to/file.csv'
INTO TABLE table_name
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

4. Once imported, start db2 using **Resync Node**. This will trigger a full resyncing job using State Snapshot Tranfer (SST). Go to ***ClusterControl GUI → Nodes → locate db2 (1) → Actions → Resync node → Resync***. { .annotate }

    1. Find the node with `Hostname: 192.168.99.112`, `Port: 3306`, `Status: Shut Down`, `Type: MariaDB` and `Role: Primary` from the nodes list.

5. Start db3 using **Resync Node**. This will trigger a full resyncing job using State Snapshot Tranfer (SST). Go to ***ClusterControl GUI → Nodes → locate db3 (1) → Actions → Resync node → Resync***. { .annotate }

    1. Find the node with `Hostname: 192.168.99.113`, `Port: 3306`, `Status: Shut Down`, `Type: MariaDB` and `Role: Primary` from the nodes list.

6. Once resynced, verify that all nodes should be green from the **Home** page. The import exercise is complete.

### 1.23.7 Step 5: Deploy load balancers and virtual IP

Integrating HAProxy and Keepalived with MariaDB Galera Cluster enhances performance, load balancing, and high availability.

1. Go to ***ClusterControl GUI → Clusters → locate the cluster → Actions → Add new → Load balancer → Create balancer → HAProxy → Continue***.

2. In the **Create a load balancer** wizard, configure the HAProxy as below:

#### (1) Where to install

- **Server address**: Specify "192.168.99.101" and press ++enter++. Wait until everything is green. If red notice appears, inspect the error seen by ClusterControl and fix it.
- **Policy**: leastconn (default) (1) { .annotate }

    1. You can choose the policy that suits your requirement.

- **Listen port (read/write)**: 3307 (default)
- **Install for read/write splitting (master-slave replication)**: Off (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)
- **Import configuration**: Unchecked (default)

### (2) Installation settings

- **Overwrite existing /usr/local/sbin/mysqlchk on target**: On (default)

### (3) Advanced settings

- **Stats socket**: /var/run/haproxy.socket (default)
- **Admin port**: 9600 (default)
- **Admin user**: admin
- **Admin password**: Enter the password for the Admin user
- **Backend name (RW)**: haproxy_3307_rw (default)
- **Backend name (RO)**: haproxy_3308_ro (default)
- **Timeout server (in seconds)**: 10800 (default)
- **Timeout client (in seconds)**: 10800 (default)
- **Max connections frontend**: 8192 (default)
- **Max connections backend (per instance)**: 64 (default)
- **xinted allow connections from**: 0.0.0.0/0 (default)

### (4) Server instances in the load balancer

- **192.168.99.111:3306 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.111]
- **192.168.99.112:3306 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.112]
- **192.168.99.113:3306 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.113]

### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

3. To deploy the load balancer, click **Finish**.

4. ClusterControl will now start provisioning the HAProxy instance. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, proceed to the next step to add Keepalived.
5. Repeat the same steps above to add the second HAProxy instance, 192.168.99.102 (lb2).
6. Go to ***ClusterControl GUI → Clusters → locate the cluster → Actions → Add new → Load balancer → Create balancer → Keepalived → Continue***.
7. In the **Create a load balancer** wizard, configure the Keepalived as below:

#### (1) Configuration

- **Load balancer type**: Select HAProxy
- **Keepalived**: Select **192.168.99.101:9600** and **192.168.99.102:9600**. Ensure everything is green.(1) { .annotate }

    1. Two HAProxy instances are required to configure the Keepalived.

- **Security configuration**
    - **Disable firewall**: Checked (default)
    - **Disable SELinux/AppArmor**: Checked (default)

#### (2) Host configuration

- **Virtual IP**: Enter the Virtual IP, as in this case is 192.168.99.100.
- **Network interface**: Specify the network interface for the Virtual IP. In this case is eth0.

#### (3) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

8. ClusterControl will now start provisioning the Keepalived instance. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the load balancer instance will appear in the **Nodes** page.
9. To check the HAProxy status, in the **Nodes** page click on the HAProxy under **Type** column and it will open the **Node details** page for the selected HAProxy. Verify everything is green here.

### 1.23.8 Step 6: Connect to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- Virtual IP (VIP) managed by Keepalived, 192.168.99.100 on port 3307. Use the virtual IP address in your connection string instead of the load balancers' primary IP address. This VIP handles all read and write traffic to the database nodes, ensuring seamless failover and high availability.
- Direct connection to a database node on port 3306.

With Keepalived managing read and write traffic through the VIP, applications can seamlessly interact with the database without needing to handle individual HAProxy instances. This setup significantly enhances redundancy and ensures that the database remains accessible at all times.

### 1.23.9 Step 7: Enable automatic backups

To protect your data, set up automatic backups. ClusterControl supports both logical and physical backups for MariaDB Galera Cluster. For a small dataset, a daily logical backup should be sufficient since by default, ClusterControl configures every database server with binary logs enabled that can be used for point-in-time recovery.

1. Go to ***ClusterControl GUI → {cluster} → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily backup - mysqldump
- **Cluster**: (leave as it is)
- **Backup host**: db3
- **Backup method**: mysqldump
- **Dump type**: Complete PITR-compatible
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Enable encryption**: On
- **Verify backup**: Off (default)
- **Retention**: On (default)
- **Use extended insert**: On (default)
- **Failover backup if node is down**: On

#### (3) Storage

- **Storage location**: Store on controller (default)
- **Storage directory**: /root/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. ClusterControl also support advanced backup features like partial backup, backup encryption, backup verification, backup throttling, back up to cloud and point-in-time recovery.

### 1.23.10 Step 8: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → {cluster} → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.23.11 Step 9: Manage your cluster and load balancer

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform rolling minor version upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → {cluster} → Actions → Upgrades***.

- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → {cluster} → Nodes → Actions***.
- HAProxy management: Perform HAProxy management related disable nodes, see the detailed status, restart service and reboot host. These features are available at ***ClusterControl GUI → {cluster} → Nodes → Actions*** and ***ClusterControl GUI → {cluster} → Nodes → HAProxy***.
- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → {cluster} → Manage → Configuration***.
- Database user management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → {cluster} → Manage → DB Users***.
- Backup management: Create, schedule, restore, perform point-in-time recovery, upload backup to cloud, verify an existing backup, and set retention period. These features are available at **ClusterControl GUI → {cluster} → Backups → Actions_** and **ClusterControl GUI → {cluster} → Backups → More_**.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → {cluster} → Nodes → Actions → SSH Console***.

### 1.23.12 Step 10: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster and scaling your MariaDB Galera Cluster is straightforward. You can scale out the cluster with another database node or a read-only replica (delayed or real-time replication and asynchronous or semi-synchronous replication). Common use cases are to use the replica node for a dedicated read-only workloads, backups, reporting or analytics purposes.

Adding a new database node or replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node if you stream directly from the primary. You may opt to stream from an existing backup to minimize this risk.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → {cluster} → Actions → Add new → Replica node → Create a replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 3306
- **Data directory**: /var/lib/mysql
- **Configuration template**: my.cnf.repl80
- **Install software**: On (default)
- **Disable firewall**: Checked (default)

- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Advanced settings

- **Rebuild from a backup**: Off (default)
- **Include in load balancer set (if exists)**: On
- **Delay the replica node**: Off (default)

#### (3) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press ++enter++.

#### (4) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

---

### 1.23.13 Conclusion

Congratulations! You've successfully deployed, monitored, managed, and scaled your first MariaDB Galera Cluster using ClusterControl. Explore additional features like failover management, advanced monitoring, backup management, and more to optimize your database infrastructure for high availability and performance.

## 1.24 PostgreSQL Streaming Replication

This guide will lead you through the process of setting up your first PostgreSQL streaming replication cluster (primary-replica), complete with an HAProxy load balancer, using ClusterControl. By following this tutorial, you'll have a fully functional database cluster that you can effortlessly monitor, manage, and scale.

PostgreSQL streaming replication is a built-in feature that allows data from a primary PostgreSQL server to be streamed in real-time to one or more replica servers. This architecture improves read scalability, offers data redundancy, and provides a basis for high availability when combined with a failover mechanism. This tutorial applies to major versions of PostgreSQL starting from 10 and above.

**See also**

- PostgreSQL Documentation on Streaming Replication

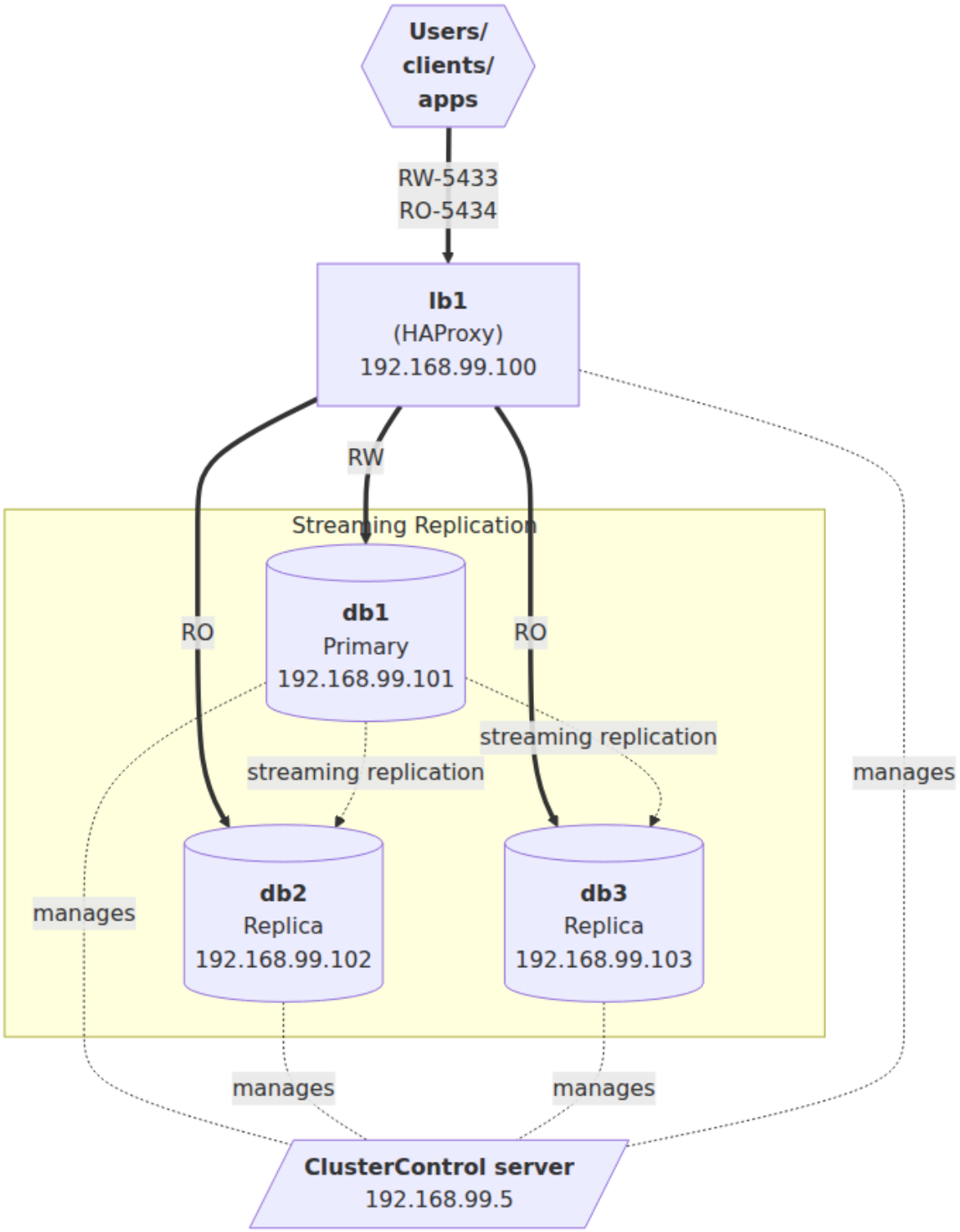
### 1.24.1 Prerequisites

Before proceeding, ensure the following requirements are met:

- ClusterControl is installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least four hosts (bare-metal or virtual machines) with a supported OS:
  - One host for the PostgreSQL primary server.
  - Two hosts for the PostgreSQL replica servers.
  - One host for the load balancer (HAProxy).
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to keep their clocks synchronized.

### 1.24.2 Architecture

Below is a high-level architecture diagram of the final architecture:




### 1.24.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes (specify the password if prompted):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.100  # lb1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db3
```

If the target node only requires a key-based authentication, you can use the `-o` flag to customize the SSH option:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.100  # lb1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.103  # db3
```

3. Perform some tests and make sure you can execute the following commands from the ClusterControl server without any password prompt:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.103 "ls /root"
```

### 1.24.4 Step 2: Deploy a new cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose "PostgreSQL Streaming" from the **Database** dropdown and the appropriate **Vendor** and **Version** (e.g., "PostgreSQL" and "16") from the wizard. Click **Continue**.
4. In the **Deploy PostgreSQL wizard**, configure the cluster as follows:

#### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g., My First PostgreSQL Cluster).
- **Tags**: Enter one or more tags, separated by pressing "Enter" (e.g., production, replication, dc1).

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave blank if using key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)

- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **Server data directory**: `/var/lib/pgsql` (default varies by distro)
- **Server port**: 5432 (default)
- **User**: `postgres`
- **Password**: Enter a secure password for the PostgreSQL `postgres` superuser
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: On (default)

#### (4) Add nodes

- **Primary node**:
  - Specify "192.168.99.101" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.99.102" as replica and press ++enter++.
  - Specify "192.168.99.103" as another replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Turn "On" **Synchronous Replication** if required.
- Proceed only if all nodes are reachable (shown in green).

#### (5) Extensions

- Leave the toggle button "Off" if you don't require the extensions, else turn "On" "pgvector" or "PostGIS" if you want to enable them.

#### (6) Preview

- Review the summary. You can go back to modify any configuration. The settings persist until you exit the wizard.

5. Click **Finish** to start the deployment.
6. ClusterControl will now start provisioning the PostgreSQL streaming replication cluster. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the cluster will appear in the **Home** page.

### 1.24.5 Step 3: Monitor your cluster

After deployment, you are redirected to the **Home** page. This page provides an at-a-glance view of the cluster's status:

- Cluster health: The **Home** page provides the cluster state.

- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.
- Verify the cluster status from the **ClusterControl GUI** by checking that the **Primary node** is active, **Replica nodes** are connected and streaming correctly, replication lag remains low, and no alarms or failed jobs are present. This helps confirm that **PostgreSQL streaming replication** is healthy and ready for normal operations.

### 1.24.6 Step 4: Import data

In a primary-replica (asynchronous) streaming replication setup, you must import the data on the primary, and changes will replicate automatically to the replicas.

To import data from a dump file on the primary (db1: 192.168.99.101), you can use `psql` command as below:

```
psql -U postgres -h 192.168.99.101 -f dumpfile.sql
```

Once the import completes and the replicas have caught up, you can proceed with normal read/write operations on the primary, while replicas remain in sync.

### 1.24.7 Step 5: Deploy a new load balancer

Integrating HAProxy with PostgreSQL improves high availability and load balancing. ClusterControl allows you to automate the deployment of HAProxy.

1. Go to ***ClusterControl GUI → Clusters → locate the cluster → Actions → Add new → Load balancer → Create balancer → HAProxy → Continue***.
2. In the **Create a load balancer** wizard, configure HAProxy as below:

#### (1) Where to install

- **Server address**: `192.168.99.100`. Press "Enter" and wait until it turns green.
- **Policy**: leastconn (default) (1) { .annotate }
    1. You can choose the policy that suits your requirement.
- **Listen port (read/write)**: 5433 (default)
- **Listen port (read-only)**: 5434 (default)
- **Install for read/write splitting (master-slave replication)**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

### (2) Installation settings

- **Overwrite existing /usr/local/sbin/postgreschk_rw_split on targets**: On (default)

### (3) Advanced settings

- **Stats socket**: /var/run/haproxy.socket (default)
- **Admin port**: 9600 (default)
- **Admin user**: admin
- **Admin password**: Enter the password for the Admin user
- **Backend name (RW)**: haproxy_5433_rw (default)
- **Backend name (RO)**: haproxy_5434_ro (default)
- **Timeout server (in seconds)**: 10800 (default)
- **Timeout client (in seconds)**: 10800 (default)
- **Max connections frontend**: 8192 (default)
- **Max connections backend (per instance)**: 1024
- **xinted allow connections from**: 0.0.0.0/0 (default)

### (4) Server instances in the load balancer

- **192.168.99.101:5432 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.101]
- **192.168.99.102:5432 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.102]
- **192.168.99.103:5432 (Primary)**: On (default)
  - **Advance options**
    - **Role**: Active (default)
    - **Connection address**: External Address [192.168.99.103]

### (4) Preview

- Review the summary. You can go back to any previous section to modify configurations if needed.

3. Click **Finish** to deploy HAProxy.
4. ClusterControl will now start provisioning the HAProxy instance. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the load balancer instance will appear in the **Nodes** page.

!!! Tips

```
For production environment, it is recommended to have at least two load␣
↪balancers to eliminate single-point of failure. You can deploy another␣
↪HAProxy instance and tie them together with a virtual IP address. See␣
↪[Load␣
↪Balancers](../../../../../user-guide/deployment/create-load-balancer.md).
```

### 1.24.8 Step 6: Connecting to the database cluster

Start by creating a new database user by going to ***ClusterControl GUI → Clusters → choose the cluster → Manage → DB Users → Create DB User***. Ensure the **Hostname** is the IP address of the application/client/user. Under the **Privileges** section, set a grant on the database and tables and add more privileges for the user by clicking **Add privileges** if necessary. Click **Create** and the database user will be created and granted on all of the database nodes.

**Tip**

You can create a new database by going to ***ClusterControl GUI → Clusters → choose the cluster → Actions → Create database***.

Next, the application/client/user can access the database cluster via:

- Load balancer instance, 192.168.99.100 on port 5433 for read-write workloads. For read-only workloads, send the database connections to port 5434. These are load-balanced PostgreSQL connections via HAProxy.
- If there are two or more load balancers, tie them together with a virtual IP address provided by Keepalived. Use the virtual IP address in your connection string instead of the load balancers' primary IP address.
- For read-write workloads, send the database connections to the PostgreSQL primary server, 192.168.99.101 on port 5432. For read-only workloads, send the database connections to either of the PostgreSQL replica servers, 192.168.99.102 or 192.168.99.103 on port 5432.

**Note**

Note If a failover happens, 192.168.99.101 might not be running as primary, and you have to find the new primary from ClusterControl GUI (or by querying directly to the database server) and send the read-write workloads there. That is why having a load balancer is a better approach for your application.

### 1.24.9 Step 7: Enable automatic backups

To protect your data, set up automatic backups. ClusterControl supports both logical (`pg_dumpall`) and physical (`pg_basebackup`, `pgBackRest`) backups for PostgreSQL.

1. Go to ***ClusterControl GUI → Clusters → choose the cluster → Backups*** to open the backup overview.
2. Go to ***Create Backup → Schedule a Backup***. The backup wizard appears.
3. In the **Create a backup schedule** wizard:

#### (1) Configuration

- **Schedule name**: e.g., `Daily Backup`
- **Cluster**: (auto-selected)
- **Backup host**: `db3` (select one of the replicas to offload backup load)
- **Backup method**: `pg_dump` or `pg_basebackup`
- **PITR enabled**: On (default for other than `pg_basebackup`)
- **Upload backup to cloud**: Off (default, can be enabled if needed)

#### (2) Advanced settings

- **Compression**: On
- **Compression level**: 6
- **Enable encryption**: Off or On (optional)
- **Verify backup**: Off (default)
- **Retention**: On, configure how many backups to keep
- **Failover backup if node is down**: Off (default)

#### (3) Storage

- **Storage location**: `Store on controller` (default)
- **Storage directory**: `/root/backups` (default)
- **Backup subdirectory**: `BACKUP-%I` (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Timezone**: Your preferred timezone

#### (5) Preview

- Review the summary. You can always go back to any previous section to modify settings.

4. Click Create to set the backup schedule.

ClusterControl will handle the scheduled backups automatically. Advanced backup features like partial backups, encryption, verification, and point-in-time recovery can also be configured.

### 1.24.10 Step 8: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server.

You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.24.11 Step 9: Manage your cluster and load balancer

ClusterControl offers many features for ongoing management:

- **PostgreSQL upgrades**: Perform rolling major or minor version upgrade to keep the cluster up-to-date. This feature is available at ***ClusterControl GUI → choose cluster → Actions → Upgrades***.
- **Node management**: Start/stop/restart a node, promote a replica to primary, reboot host, etc. These features are available at ***ClusterControl GUI → choose cluster → Nodes → Actions***.
- **Load balancer management**: Manage and configure HAProxy backends, frontend ports, perform service restarts, and advanced routing. These features are available at ***ClusterControl GUI → choose cluster → Nodes → Actions*** (for HAProxy nodes).
- **Configuration management**: Adjust PostgreSQL settings globally or on individual nodes. This feature is available at ***ClusterControl GUI → choose cluster → Manage → Configuration***.
- **User management**: Manage PostgreSQL roles and privileges. These features are available at ***ClusterControl GUI → choose cluster → Manage → DB Users***.
- **Backup management**: Create, schedule, restore, perform point-in-time recovery, upload backups to cloud, verify backups, and set retention. These features are available at ***ClusterControl GUI → choose cluster → Backups***.
- **Maintenance mode**: Activate or schedule maintenance for nodes. These features are available at ***ClusterControl GUI → choose cluster → Nodes → Actions → Schedule maintenance***.
- **SSH console**: Directly access nodes via a web-based SSH console. This feature is available at ***ClusterControl GUI → choose cluster → Nodes → Actions → SSH Console***.

### 1.24.12 Step 10: Scale your cluster

A key advantage of ClusterControl is easy scalability. You can add more replica nodes to handle greater read workloads or to distribute backups, analytics, or failover targets.

1. To add a new replica, go to ***ClusterControl GUI → choose cluster → Actions → Add new → Replica node***.
2. In the **Create a replica node** wizard, configure:

#### (1) Node configuration

- **Port**: 5432 (or your chosen port)
- **Data directory**: `/var/lib/pgsql`
- **Install software**: On
- **Disable firewall**: Checked
- **Disable SELinux/AppArmor**: Checked

#### (2) Advanced settings

- **Include in load balancer set**: On (if you want it in HAProxy)
- **Synchronous replication**: 0ff (default)
- **Instance name**: Enter instance name
- **Version**: Pre-selected by default

#### (3) Add node

- **Primary node**: Pick the primary node from the dropdown `192.168.99.101`.
- **Node**: Type the new replica node IP, then `Enter` and wait until green:

#### (4) Preview

- Confirm your deployment summary.

3. Click **Finish** to trigger the job.
4. Monitor the deployment from **Activity center**. When complete, the new replica node appears in **Nodes** page.

### 1.24.13 Conclusion

Congratulations! You've successfully deployed, monitored, managed, and scaled your first PostgreSQL Streaming Replication cluster with ClusterControl. Explore additional features like failover management, advanced monitoring, backup management, and more to further optimize your PostgreSQL infrastructure for high availability and performance.

This guide will lead you through setting up your first TimescaleDB streaming replication cluster (primary-replica) with PgBouncer as a load balancer using ClusterControl. By following this tutorial, you'll have a fully functional database cluster that you can effortlessly monitor, manage, and scale.

TimescaleDB Streaming Replication is a built-in feature that allows data from a primary TimescaleDB server to be streamed in real-time to one or more replica servers. This architecture improves read scalability, provides data redundancy, and enhances high availability when combined with a failover mechanism.

**See also**

- TimescaleDB Documentation
- PgBouncer Documentation

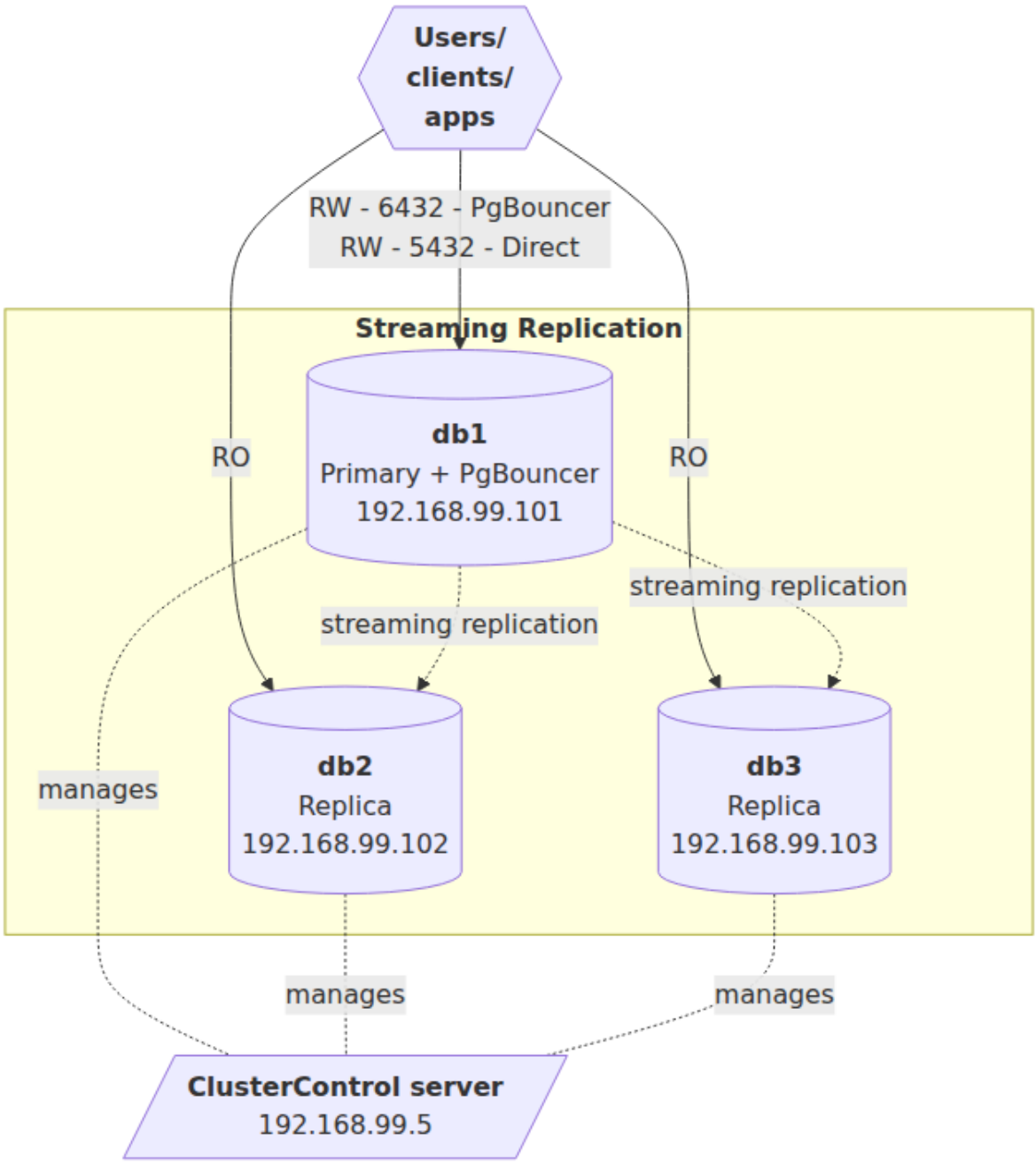
## 1.25 Prerequisites

Ensure the following requirements are met before proceeding:

- ClusterControl is installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least four hosts (bare-metal or virtual machines) with a supported OS:
  - One host for the TimescaleDB primary server
  - Two hosts for the TimescaleDB replica servers
  - One host for the PgBouncer load balancer
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to keep their clocks synchronized.

## 1.26 Architecture

Below is a simplified diagram of the final architecture:




## 1.27 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes:

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.100  # lb1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.103  # db3
```

3. Test connectivity from the ClusterControl server to each host:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.103 "ls /root"
```

## 1.28 Step 2: Deploy a new cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose "TimescaleDB" from the **Database** dropdown and the appropriate **Vendor** and **Version** (e.g., "TimescaleDB" and "16") from the wizard. Click **Continue**.
4. In the **Deploy TimescaleDB wizard**, configure the cluster as follows:

### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g., My First TimescaleDB Cluster).
- **Tags**: Enter one or more tags, separated by pressing "Enter" (e.g., production, replication, dc1).

### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave blank if using key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

### (3) Node configuration

- **Server data directory**: `/var/lib/pgsql` (default varies by distro)
- **Server port**: 5432 (default)
- **User**: `postgres`
- **Password**: Enter a secure password for the TimescaleDB `postgres` superuser
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: On (default)

#### (4) Add nodes

- **Primary node**:
  - Specify "192.168.99.101" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.99.102" as replica and press ++enter++.
  - Specify "192.168.99.103" as another replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Turn "On" **Synchronous Replication** if required.
- Proceed only if all nodes are reachable (shown in green).

#### (5) Extensions

- Leave the toggle button "Off" if you don't require the extensions, else turn "On" "pgvector" or "PostGIS" if you want to enable them.

#### (6) Preview

- Review the summary. You can go back to modify any configuration. The settings persist until you exit the wizard.

5. Click Finish to start the deployment.
6. ClusterControl will provision your TimescaleDB streaming replication cluster. Monitor progress from **Activity center**. This process can take a few minutes. Once complete, the cluster appears on the Home page.

## 1.29 Step 3: Monitor your cluster

After deployment, the **Home** page provides an overview of your cluster's health, node status, alarms, automatic recovery status, and topology visualization.

## 1.30 Step 4: Deploy PgBouncer

Integrating PgBouncer with TimescaleDB improves connection pooling and scalability. Follow these steps to deploy PgBouncer with ClusterControl:

1. Navigate to ***ClusterControl → Clusters → locate the cluster → Actions → Add new → Load balancer → Create balancer → PgBouncer → Continue***.
2. In the **Create a load balancer** wizard, configure PgBouncer:

#### (1) Authentication

- **PgBouncer admin user**: Enter a PgBouncer admin user. For example: `pgbouncer`.
- **PgBouncer admin password**: : Enter the password.

#### (2) Add nodes

- **PgBouncer node**: Select the TimescaleDB node from the dropdown menu and wait until the status turned green.
- **Listen port**: 6432 (default).

#### (4) Preview

- Review the summary. You can go back to any previous section to modify configurations if needed.

3. Click **Finish** to deploy PgBouncer.
4. ClusterControl provisions the PgBouncer instance. Monitor the progress from Activity center. Once done, the load balancer appears under **Nodes**.

## 1.31 Step 5: Creating connection pool for PgBouncer

Now that the PgBouncer is deployed, the next step is to create the connection pool. To create a connection pool:

1. Navigate to **ClusterControl → Clusters → Nodes**, click on the node that is showing PgBouncer under **Type** category.
2. In the **Nodes details** page, click on **Create new pool**:
3. In the **Create new pool** wizard, configure connection pool:

#### Create new pool

- **PgBouncer host name**: Pick the PgBouncer node.
- **Pool Name**: : Enter connection pool name.
- **Username**: select the database user from the dropdown menu.
- **Pool Mode**: Select one of the following modes:
  - `session`: Releases server after client disconnects.
  - `transaction`: Releases server after transaction completes.
  - `statement`: Releases server after each query. Multi-statement transactions are not allowed in this mode.
- **Pool Size**: Leave empty to use the default.
- **Max Connections**: Enter the maximum number of connections.

4. Click **Add** to create the connection pool.

Once created, PgBouncer will begin managing connections based on your configured pool parameters.

## 1.32 Step 6: Connecting to the database cluster

For read-write workloads, connect to the primary TimescaleDB serrver, for example:

```
psql -U postgres -h 192.168.99.101 -p 5432
```

## 1.33 Step 7: Connecting to the database cluster via PgBouncer

After creating a connection pool in ClusterControl, you can connect to that specific pool using the format:

```
psql -U <username> -h <pgbouncer_host> -p <pgbouncer_port> -d <pool_name>
```

Example: If you created a pool named `app_pool` for user `appuser`, and PgBouncer runs on 192.168.99.101:

```
psql -U appuser -h 192.168.99.101 -p 6432 -d app_pool
```

!!! important The `-d` parameter must match the pool name you defined during pool creation (not necessarily the actual database name).

Parameters:

- `U`: The database user assigned to the pool
- `h`: PgBouncer host/IP
- `p`: PgBouncer listening port (default: 5432)
- `d`: The pool name (acts like the database name to PgBouncer)

## 1.34 Step 8: Enable automatic backups

To protect your data, set up automatic backups. ClusterControl supports both logical (pg_dump) and physical (pg_basebackup, pgBackRest) backups for TimescaleDB.

1. Go to ***ClusterControl → Clusters → choose the cluster → Backups*** to open the backup overview.
2. Go to ***Create Backup → Schedule a Backup***. The backup wizard appears.
3. In the **Create a backup schedule** wizard:

### (1) Configuration

- **Schedule name**: e.g., `Daily Backup`
- **Cluster**: (auto-selected)
- **Backup host**: `db3` (select one of the replicas to offload backup load)
- **Backup method**: `pg_dump` or `pg_basebackup`
- **PITR enabled**: On (default for other than pgbasebackup)
- **Upload backup to cloud**: Off (default, can be enabled if needed)

### (2) Advanced settings

- **Compression**: On
- **Compression level**: 6
- **Enable encryption**: Off or On (optional)
- **Verify backup**: Off (default)
- **Retention**: On, configure how many backups to keep

- **Failover backup if node is down**: Off (default)

#### (3) Storage

- **Storage location**: `Store on controller` (default)
- **Storage directory**: `/root/backups` (default)
- **Backup subdirectory**: `BACKUP-%I` (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Timezone**: Your preferred timezone

#### (5) Preview

- Review the summary. You can always go back to any previous section to modify settings.

4. Click Create to set the backup schedule.

ClusterControl will handle the scheduled backups automatically. Advanced backup features like partial backups, encryption, verification, throttling, and point-in-time recovery can also be configured.

## 1.35 Step 9: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

## 1.36 Step 10: Manage your cluster

ClusterControl offers many features for ongoing management:

- **PostgreSQL upgrades**: Perform rolling version upgrade to keep the cluster up-to-date. This feature is available at ***ClusterControl GUI → choose cluster → Actions → Upgrades***.
- **Node management**: Start/stop/restart a node, promote a replica to primary, reboot host, etc. These features are available at ***ClusterControl GUI → choose cluster → Nodes → Actions***.
- **Connection pool management**: Manage and configure PgBouncer connection pools and databases. These features are available at ***ClusterControl GUI → choose cluster → Nodes → Actions*** (for PgBouncer nodes).
- **Configuration management**: Adjust TimescaleDB settings globally or on individual nodes. This feature is available at ***ClusterControl GUI → choose cluster → Manage → Configuration***.
- **User management**: Manage TimescaleDB roles and privileges. These features are available at ***ClusterControl GUI → choose cluster → Manage → DB Users***.
- **Backup management**: Create, schedule, restore, perform point-in-time recovery, upload backups to cloud, verify backups, and set retention. These features are available at ***ClusterControl GUI → choose cluster → Backups***.
- **Maintenance mode**: Activate or schedule maintenance for nodes. These features are available at ***ClusterControl GUI → choose cluster → Nodes → Actions → Schedule maintenance***.
- **SSH console**: Directly access nodes via a web-based SSH console. This feature is available at ***ClusterControl GUI → choose cluster → Nodes → Actions → SSH Console***.

## 1.37 Conclusion

Congratulations! You have successfully deployed a TimescaleDB streaming replication with PgBouncer as a load balancer using ClusterControl. Explore additional features such as automatic failover, monitoring, and backup strategies to enhance your database's reliability and performance.

## 1.38 PostgreSQL Logical Replication

This tutorial introduces how to implement PostgreSQL Logical Replication in a bi-directional architecture using ClusterControl. It is particularly suited for environments that require multi-region write availability, zero-downtime upgrades, or active-active application architectures.

### 1.38.1 Introduction

PostgreSQL supports two replication paradigms:

- **Streaming Replication** — built-in, synchronous or asynchronous binary-level replication from primary to read-only standbys.

- **Logical Replication** — table-level, row-based, asynchronous replication using a publish/subscribe model.

Logical replication is ideal when:

- You want granular control (replicating specific tables or schemas).
- You need to replicate between different PostgreSQL versions.
- You require multi-primary setups, like bi-directional replication.

**Warning**

Warning!

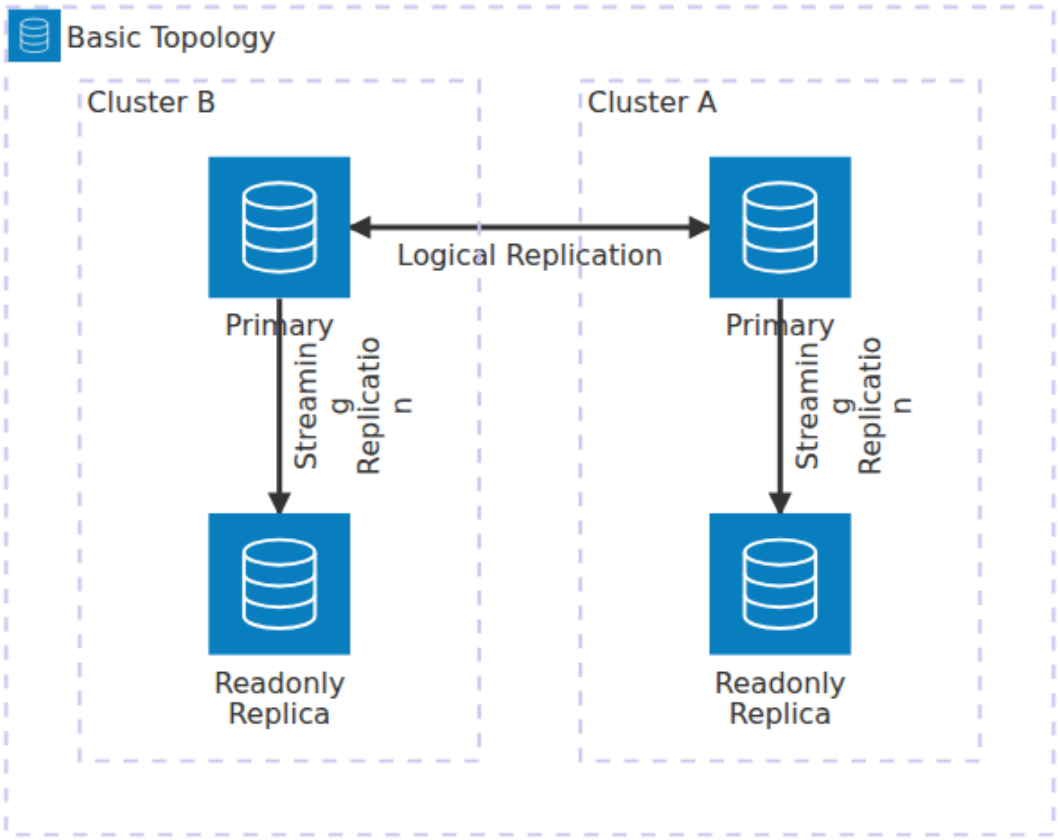
Logical replication does not replicate DDL changes or automatically resolve conflicts.

### 1.38.2 Architecture

The bi-directional setup supported by ClusterControl consists of two PostgreSQL streaming replication clusters, each configured with:

- One primary node
- One or more read-only replicas
- Logical replication enabled between the two primary nodes

Below is a high-level architecture diagram of the final architecture:




### 1.38.3 Prerequisites

Ensure you have:

- ClusterControl version 2.3.1 or higher. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- The number of nodes that you want to include in the clusters with a supported OS.
- Network connectivity between all the database nodes and ClusterControl.
- SSH access to all the nodes.

### 1.38.4 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes (specify the password if prompted):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.103  # db3
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.99.104  # db4
```

If the target node only requires a key-based authentication, you can use the `-o` flag to customize the SSH option:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.101  # db1
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.102  # db2
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.103  # db3
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.99.104  # db4
```

3. Perform some tests and make sure you can execute the following commands from the ClusterControl server without any password prompt:

```
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.103 "ls /root"
ssh root@192.168.99.104 "ls /root"
```

### 1.38.5 Step 2: Deploy a new cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose "PostgreSQL Logical" from the **Database** dropdown and the appropriate **Vendor** and **Version** (e.g., "PostgreSQL" and "17") from the wizard. Click **Continue**.
4. In the **Deploy PostgreSQL Logical cluster wizard**, configure the cluster as follows:

#### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g., My First PostgreSQL Cluster).
- **Tags**: Enter one or more tags, separated by pressing "Enter" (e.g., production, replication, dc1).

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave blank if using key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

### (3) Node configuration

- **Server data directory**: `/var/lib/pgsql` (default varies by distro)
- **Server port**: 5432 (default)
- **User**: `postgres`
- **Password**: Enter a secure password for the PostgreSQL `postgres` superuser
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: On (default)

### (4) Add nodes

- **Cluster 1**: It means the first PostgreSQL Streaming Replication cluster.
- **Primary node**:
  - Specify "192.168.99.101" as the primary node for Cluster 1 and press ++enter++.
- **Replica node**:
  - Specify "192.168.99.102" as replica and press ++enter++.
  - You can add more replica nodes here.
  - Turn "On" **Synchronous Replication** if required.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- **Cluster 2**: It means the second PostgreSQL Streaming Replication cluster.
- **Primary node**:
  - Specify "192.168.99.103" as the primary node for Cluster 2 and press ++enter++.
- **Replica node**:
  - Specify "192.168.99.104" as replica and press ++enter++.
  - You can add more replica nodes here.
  - Turn "On" **Synchronous Replication** if required.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

### (5) Extensions

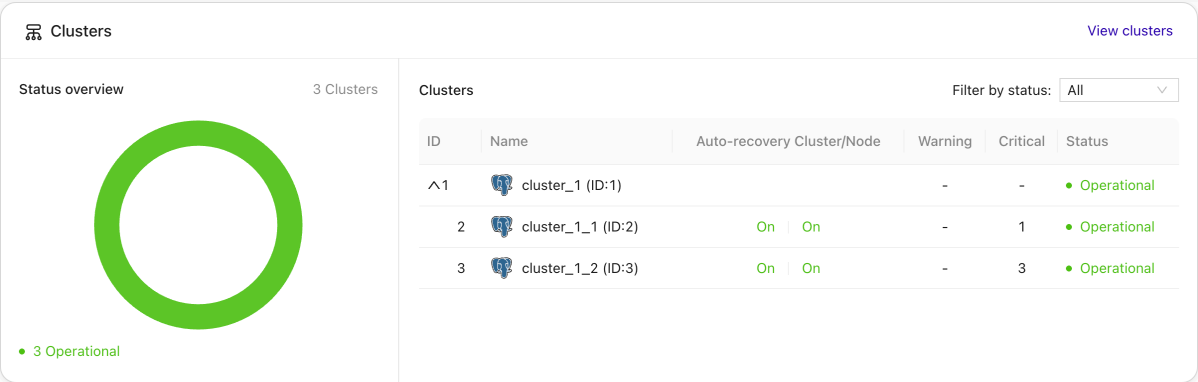
- Leave the toggle button "Off" if you don't require the extensions, else turn "On" "pgvector" or "PostGIS" if you want to enable them.

### (6) Preview

- Review the summary. You can go back to modify any configuration. The settings persist until you exit the wizard.

5. Click **Finish** to start the deployment.

6. ClusterControl will now start provisioning both PostgreSQL streaming replication clusters. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the clusters will appear in the **Home** page.




### 1.38.6 Step 3: Publication and Subscription

Now, for PostgreSQL Logical Replication, you have to create a Publication (a set of tables to be replicated), and a Subscription to access this Publication and stream changes.

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. Choose your PostgreSQL Logical cluster from the list.
3. Go to **Create publications / subscriptions**.
4. Create a new **publication** to publish changes to a set of tables, which can then be consumed by logical replication subscribers when creating a **subscription**:

#### (1) Create publication

- **Logical cluster**: The selected PostgreSQL Logical cluster.
- **Publisher cluster**: Choose one of the PostgreSQL Streaming Replication clusters.
- **Database**: Choose the databases to be replicated.
- **Publication name**: Create a name for the new **publication**.
- **Include all tables**: On (default).
- Then, press on **Create publication**.

#### (2) Create subscription

- **Logical cluster**: The selected PostgreSQL Logical cluster.
- **Subscriber cluster**: Choose one of the PostgreSQL Streaming Replication clusters.
- **Subscription name**: Create a name for the new **subscription**.
- **Publication**: Choose an existing **publication**.
- **Database**: Choose the database where to replicate data.
- **Copy data**: On (default). This flag should be false when creating second subscription on a bi-directional link. Specifies whether to copy pre-existing data in the publications that are being subscribed to when the replication starts.

- **Origin**: **None** (default). Set in **None** means the subscription will only replicate changes that do not have an associated origin (i.e., changes made locally). If origin is set in **Any**, the subscription will replicate all changes regardless of their origin, including those already replicated from other nodes.
- Then, press on **Create subscription**.
- **WARNING: Setting origin=any can lead to a potential infinite loop in bi-directional replication setups, as changes replicated from one node can be sent back to the originating node.**

For more information about each PostgreSQL Streaming Replication setup, refer to *PostgreSQL Streaming Replication*

This guide will lead you through deploying your first MongoDB ReplicaSet using ClusterControl. You'll end up with a fully functional, production-ready MongoDB cluster that's equipped for monitoring, automatic failover, backups, scaling, and alerting.

MongoDB ReplicaSet is a built-in high availability feature that replicates data from a primary node to one or more secondary nodes. ClusterControl simplifies the setup process, offering automation, visibility, and lifecycle management.

**See also**

- MongoDB Replica Set Docs

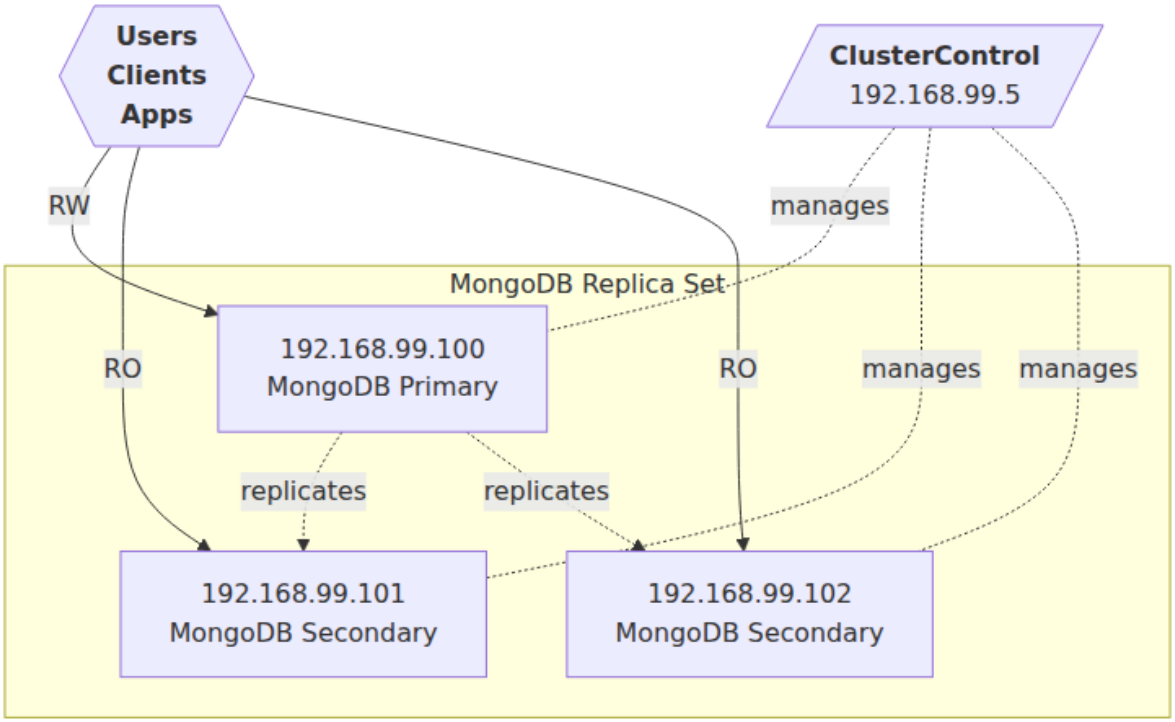
## 1.39 Prerequisites

Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quickstart* or use the *Installer Script*.
- You need at least three hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.
  - One host for the MongoDB primary server.
  - Two hosts for the MongoDB replica servers.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to keep their clocks synchronized.

## 1.40 Architecture

Below is the architecture diagram of the replica set:




## 1.41 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.100  # Primary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.101  # Secondary 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.102  # Secondary 2
```

3. Test connectivity from the ClusterControl server to each host:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
```

## 1.42 Step 2: Deploy MongoDB replica set

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose "MongoDB" from the **Database** dropdown and the appropriate **Vendor** and **Version** (e.g., "MongoDB" and "7.0") from the wizard. Click **Continue**.
4. In the **Deploy MongoDB wizard**, configure the cluster as follows:

### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g., My First MongoDB Cluster).
- **Tags**: Enter one or more tags, separated by pressing "Enter" (e.g., production, replication, dc1).

### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave blank if using key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

### (3) Node configuration

- **Server port**: 27017 (default)
- **Server data directory**: `/var/lib/mongodb`
- **User**: Enter MongoDB admin user eg: `mongoadm`
- **Password**: Enter a secure password for the MongoDB admin
- **ReplicaSet name**: Enter your ReplicaSet name
- **Configuration template**: mongodb.conf.7.0.percona (default)
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: On (default)

### (4) Add nodes

- **Primary node**:
  - Specify "192.168.99.100" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.99.101" as replica and press ++enter++.
  - Specify "192.168.99.102" as another replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

#### (6) Preview

- Review the summary. You can go back to modify any configuration. The settings persist until you exit the wizard.

5. Click Finish to start the deployment.
6. ClusterControl will provision your MongoDB ReplicaSet cluster. Monitor progress from **Activity center**. This process can take a few minutes. Once complete, the cluster appears on the Home page.

## 1.43 Step 3: Monitor your cluster

After deployment, the **Home** page provides an overview of your cluster's health, node status, alarms, automatic recovery status, and topology visualization.

## 1.44 Step 4: Schedule automatic backups

To protect your data, set up automatic backups:

1. Go to ***ClusterControl → Clusters → choose the cluster → Backups*** to open the backup overview.
2. Go to ***Create Backup → Schedule a Backup***. The backup wizard appears.
3. In the **Create a backup schedule** wizard:

#### (1) Configuration

- **Schedule name**: Enter the backup name, eg: `DailyBackup`
- **Cluster**: Auto selected by default.
- **Backup method**: mongodump.
- **Upload to cloud**: Off (enable if configured)

#### (2) Advanced settings

- **Retention**: 31 Days (default)
- **Enable encryption**: `Off` (default)

#### (3) Configure storage

- **Storage host**: Select your storage host
- **Storage directory**: Set accordingly
- **Backup subdirectory**: Set accordingly

#### (5) Schedule

- **Set backup schedule**: Simple
- **Every**: Your preferred time
- **Timezone**: Your preferred timezone

#### (4) Preview

- Review and click **Create**

4. Click **Create** to set the backup schedule.

## 1.45 Step 5: Configure alerts and notifications

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

## 1.46 Step 6 Connect to the cluster

Use the Mongo shell or your application driver:

```
mongosh -u <mongo_admin> -p <password> --authenticationDatabase admin --
↪host 192.168.99.100
```

Check replica set status:

```
rs.status()
```

## 1.47 Step 7: Cluster management operations

ClusterControl supports:

- Adding/removing replica nodes
- Performing rolling upgrades
- Resyncing lagging nodes
- Manual or automatic failover
- Customizing MongoDB configuration
- Upgrading MongoDB versions
- Deploying sharded clusters (optional)
- Managing backup schedules and restores
- User and role management

All operations can be executed safely from the UI without logging into individual nodes.

## 1.48 Conclusion

You have successfully deployed a MongoDB ReplicaSet using ClusterControl. Your cluster is now fault-tolerant, secure, and manageable through a unified interface.

This guide will lead you through deploying your first MongoDB Enterprise Replica Set using ClusterControl. You'll end up with a fully functional, production-ready MongoDB cluster that's equipped for monitoring, automatic failover, backups, scaling, and alerting.

MongoDB Community and MongoDB Enterprise use the same core database engine and offer identical functionality for data modeling, querying, replication, sharding, and transactions, but they differ significantly in security, compliance, and operational features. MongoDB Enterprise adds capabilities required in regulated and corporate environments, such as encryption at rest, auditing, LDAP and Kerberos authentication, FIPS compliance, enterprise key management, and commercial support, none of which are available in the Community edition.

MongoDB Enterprise Replica Set is a built-in high availability feature that replicates data from a primary node to one or more secondary nodes. ClusterControl simplifies the setup process, offering automation, visibility, and lifecycle management.

**See also**

- MongoDB Enterprise Replica Set Docs

## 1.49 Prerequisites

Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quickstart* or use the *Installer Script*.
- You need at least three hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.

- One host for the MongoDB primary server.
- Two hosts for the MongoDB replica servers.

- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to keep their clocks synchronized.

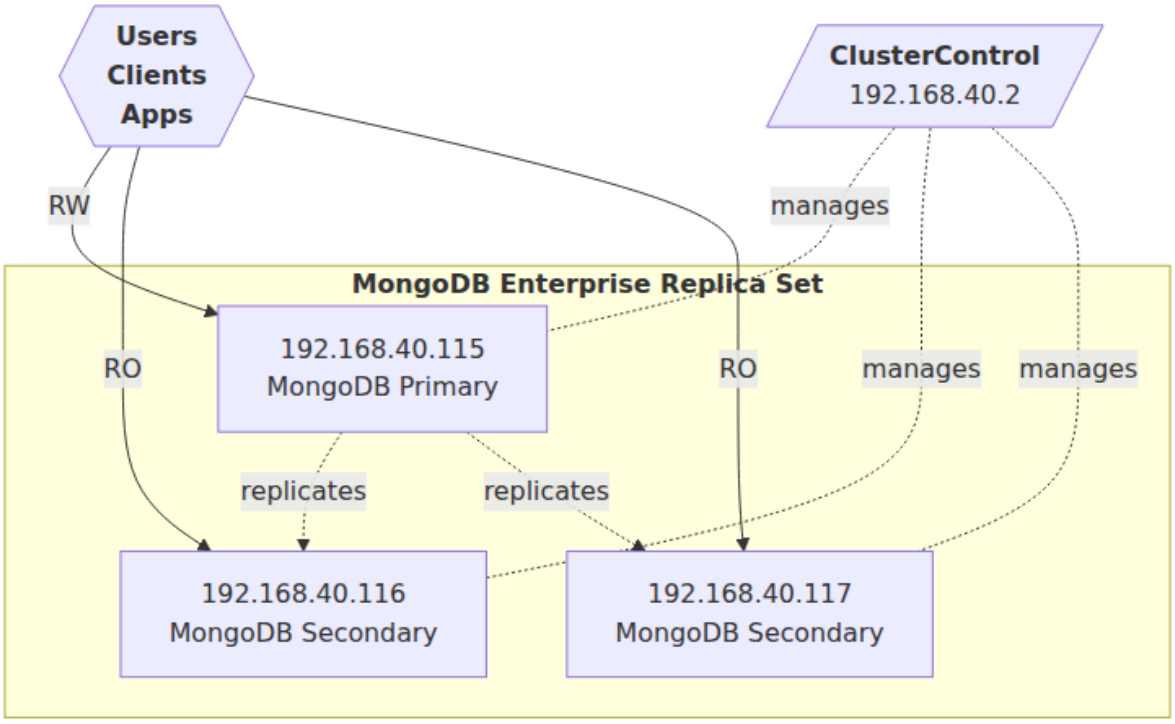
**Warning**

Attention

For production usage, ensure you have a valid MongoDB Enterprise commercial subscription license for your cluster. MongoDB Enterprise can be installed without a license but only permitted for evaluation purposes.

## 1.50 Architecture

Below is the architecture diagram of the replica set:




## 1.51 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.115  # Primary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.116  # Secondary 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.117  # Secondary 2
```

3. Test connectivity from the ClusterControl server to each host:

```
ssh root@192.168.40.115 "ls /root"
ssh root@192.168.40.116 "ls /root"
ssh root@192.168.40.117 "ls /root"
```

## 1.52 Step 2: Deploy MongoDB Enterprise Replica Set

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose "MongoDB" from the **Database** dropdown and the appropriate **Vendor** and **Version** (e.g., "MongoDB" and "7.0") from the wizard. Click **Continue**.
4. In the **Deploy MongoDB wizard**, configure the cluster as follows:

### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g., My First MongoDB Cluster).
- **Tags**: Enter one or more tags, separated by pressing "Enter" (e.g., production, replication, dc1).

### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave blank if using key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

### (3) Node configuration

- **Server port**: 27017 (default)
- **Server data directory**: `/var/lib/mongodb`
- **User**: Enter MongoDB admin user eg: `mongoadm`
- **Password**: Enter a secure password for the MongoDB admin
- **Replica Set name**: Enter your Replica Set name
- **Configuration template**: mongodb.conf.7.0.percona (default)

- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: On (default)

#### (4) Add nodes

- **Primary node**:
  - Specify "192.168.40.115" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.40.116" as replica and press ++enter++.
  - Specify "192.168.40.117" as another replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

#### (6) Preview

- Review the summary. You can go back to modify any configuration. The settings persist until you exit the wizard.

5. Click Finish to start the deployment.
6. ClusterControl will provision your MongoDB Enterprise Replica Set cluster. Monitor progress from **Activity center**. This process can take a few minutes. Once complete, the cluster appears on the Home page.

## 1.53 Step 3: Monitor your cluster

After deployment, the **Home** page provides an overview of your cluster's health, node status, alarms, automatic recovery status, and topology visualization.

## 1.54 Step 4: Schedule automatic backups

To protect your data, set up automatic backups:

1. Go to ***ClusterControl → Clusters → choose the cluster → Backups*** to open the backup overview.
2. Go to ***Create Backup → Schedule a Backup***. The backup wizard appears.
3. In the **Create a backup schedule** wizard:

#### (1) Configuration

- **Schedule name**: Enter the backup name, eg: `DailyBackup`
- **Cluster**: Auto selected by default.
- **Backup method**: mongodump.
- **Upload to cloud**: Off (enable if configured)

#### (2) Advanced settings

- **Retention**: 31 Days (default)
- **Enable encryption**: `Off` (default)

#### (3) Configure storage

- **Storage host**: Select your storage host
- **Storage directory**: Set accordingly
- **Backup subdirectory**: Set accordingly

#### (5) Schedule

- **Set backup schedule**: Simple
- **Every**: Your preferred time
- **Timezone**: Your preferred timezone

#### (4) Preview

- Review and click **Create**

4. Click **Create** to set the backup schedule.

## 1.55 Step 5: Configure alerts and notifications

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

## 1.56 Step 6: Connect to the cluster

Use the Mongo shell or your application driver:

```
mongosh -u <mongo_admin> -p <password> --authenticationDatabase admin --
↪host 192.168.40.115
```

Check replica set status:

```
rs.status()
```

## 1.57 Step 7: Verify Enterprise installation

Based on step 6, as you are still connected through `mongosh`, you can verify the MongoDB node if it is an Enterprise version by running the following command in:

```
db.serverBuildInfo().modules
```

## 1.58 Step 8: Cluster management operations

For MongoDB Enterprise replica set, ClusterControl supports:

- Adding/removing replica nodes
- Performing rolling upgrades
- Resyncing lagging nodes
- Manual or automatic failover
- Customizing MongoDB configuration
- Upgrading MongoDB versions
- Deploying sharded clusters (optional)
- Managing backup schedules and restores
- User and role management

All operations can be executed safely from the GUI without logging into individual nodes.

## 1.59 Conclusion

You have successfully deployed a MongoDB Enterprise Replica Set using ClusterControl. Your cluster is now fault-tolerant, secure, and manageable through a unified interface.

This guide will lead you through deploying your first MongoDB Sharded Cluster using ClusterControl. By the end, you'll have a production-ready sharded setup that's scalable, fault-tolerant, and fully managed through ClusterControl.

A MongoDB sharded cluster is designed to scale horizontally by partitioning data across multiple shard nodes. Each shard can be a replica set to ensure high availability. ClusterControl automates the deployment, configuration, and ongoing management of a sharded cluster.

**See also**

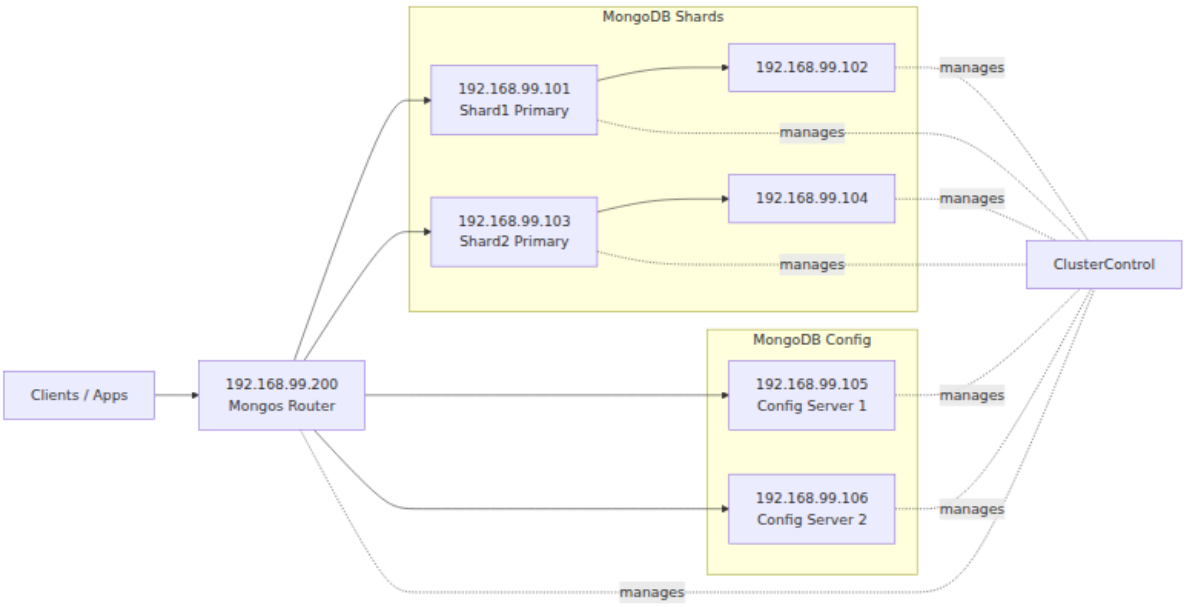
- MongoDB Sharding Docs

## 1.60 Prerequisites

Make sure you have the following before you begin:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quick-start* or use the *Installer Script*.
- At least six hosts (bare-metal or virtual machines):
  - Three hosts for MongoDB shard replica sets (at least 2 nodes per shard)
  - Two config server nodes
  - One mongos router node (can share with ClusterControl)
- All nodes must have internet access during deployment.
- SSH key-based access from ClusterControl to all target hosts.
- NTP service must be enabled and synchronized on all nodes.

## 1.61 Architecture



## 1.62 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

2. Copy the SSH public key to all target nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.200  # Mongos Router
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.101  # Shard 1 Primary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.102  # Shard 1 Secondary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.103  # Shard 2 Primary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.104  # Shard 2 Secondary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.105  # Config Server 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.106  # Config Server 2
```

3. Test connectivity from the ClusterControl server to each host:

```
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
ssh root@192.168.99.103 "ls /root"
ssh root@192.168.99.104 "ls /root"
ssh root@192.168.99.105 "ls /root"
ssh root@192.168.99.106 "ls /root"
```

## 1.63 Step 2: Deploy MongoDB Sharded Cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose "MongoDB Shards" from the **Database** dropdown and the appropriate **Vendor** and **Version** (e.g., "MongoDB" and "7.0") from the wizard. Click **Continue**.
4. In the **Deploy MongoDB wizard**, configure the cluster as follows:

### (1) Cluster details

- **Name**: Enter a name for your cluster (e.g., My First MongoDB Shard Cluster).
- **Tags**: Enter one or more tags, separated by pressing "Enter" (e.g., production, replication, dc1).

### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo password**: (leave blank if using key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)

- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

### (3) Node configuration

- **Server port**: 5432 (default)
- **Server data directory**: `/var/lib/mongodb`
- **User**: Enter MongoDB admin user eg: `mongoadm`
- **Password**: Enter a secure password for the MongoDB admin
- **Router Configuration template**: mongodb.conf.7.0.percona (default)
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: On (default)

### (4) Add Configuration Servers & Routers

- **Configuration Servers**:
  - Specify "192.168.99.105" and "192.168.99.106" as the Configuration Servers and press ++enter++.
- **Router/Mongos node**:
  - Specify "192.168.99.200" as Mongo Router and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

### (5) Add Shard

- **Define shard name**:
  - Specify the shard name for the First Shard.
- **Nodes Shard**:
  - Specify "192.168.99.101" as Primary Shard and press ++enter++.
  - Specify "192.168.99.102" as Secondary/Replica Shard and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).
- **Define another shard name**:
  - Press ***Add another shard*** and specify the shard name for the Second Shard.
- **Nodes Shard**:
  - Specify "192.168.99.103" as Primary Shard and press ++enter++.
  - Specify "192.168.99.104" as Secondary/Replica Shard and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

#### (6) Preview

- Review the summary. You can go back to modify any configuration. The settings persist until you exit the wizard.

5. Click Finish to start the deployment.
6. ClusterControl will provision your MongoDB Sharded Cluster. Monitor progress from **Activity center**. This process can take a few minutes. Once complete, the cluster appears on the Home page.

## 1.64 Step 3: Monitor your cluster

After deployment, the **Home** page provides an overview of your cluster's health, node status, alarms, automatic recovery status, and topology visualization.

## 1.65 Step 4: Schedule automatic backups

ClusterControl uses Percona Backup for MongoDB as default backup method for MongoDB Sharded Cluster. Before installing Percona Backup for MongoDB and configuring the backup, ensure that the servers have mounted the shared storage on all data nodes and config servers. The articles will not cover how to set up the NFS server and client.

To protect your data, set up automatic backups:

1. Go to ***ClusterControl → Clusters → choose the cluster → Backups*** to open the backup overview.
2. Go to ***Create Backup → Schedule a Backup***. The backup wizard appears.
3. In the **Create a backup schedule** wizard:

#### (1) Configuration

- **Schedule name**: Enter the backup name, eg: `DailyBackup`
- **Cluster**: Auto selected by default.
- **Backup method**: percona-backup-mongodb.

We must install the PBM (Percona Backup for MongoDB) for the first setup. just **Continue** and fill the shared storage path directory.

#### (2) Advanced settings

- **Retention**: 31 Days (default)
- **Enable encryption**: `Off` (default)

#### (3) Schedule

- **Set backup schedule**: Simple
- **Every**: Your preferred time
- **Timezone**: Your preferred timezone

#### (4) Preview

- Review and click **Create**

4. Click **Create** to set the backup schedule.

## 1.66 Step 5: Configure alerts and notifications

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

## 1.67 Step 6 Connect to the cluster

Use the Mongo shell or your application driver:

```
mongosh -u <mongo_admin> -p <password> --authenticationDatabase admin --
↪host 192.168.99.200
```

Check shard cluster status:

```
sh.status()
```

## 1.68 Step 7: Cluster management operations

ClusterControl supports:

- Adding/Remove shards
- Adding/remove mongos router & config server
- Add new nodes in shards
- Performing minor rolling upgrades
- Resyncing lagging nodes
- Manual or automatic failover
- Customizing MongoDB configuration
- Upgrading MongoDB versions
- Managing backup schedules and restores
- User and role management

All operations can be executed safely from the UI without logging into individual nodes.

## 1.69 Conclusion

You have successfully deployed a MongoDB Sharded Cluster using ClusterControl. Your cluster is now fault-tolerant, secure, and manageable through a unified interface.

## 1.70 Redis Replication with Sentinel

In this guide, we will walk you through setting up your first database cluster running on Redis replication with Sentinel using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

Redis is an open-source, in-memory data store that combines the speed of a cache with the flexibility of a database and the messaging capabilities of a broker. It holds data entirely in RAM for sub-millisecond read and write performance, while offering optional persistence to disk so your datasets survive restarts. Beyond simple key/value pairs, Redis natively supports rich data types — lists, sets, sorted sets, hashes, bitmaps, hyperloglogs, and streams — making it easy to implement use cases like caching, real-time analytics, leaderboards, session storage, and pub/sub messaging.

With built-in replication, automatic failover via Redis Sentinel, and sharding in Redis Cluster, it can scale seamlessly from a single-server prototype to a highly available, distributed system serving millions of operations per second.

**See also**

- Redis Community Edition - Getting Started
- Redis Sentinel Docs

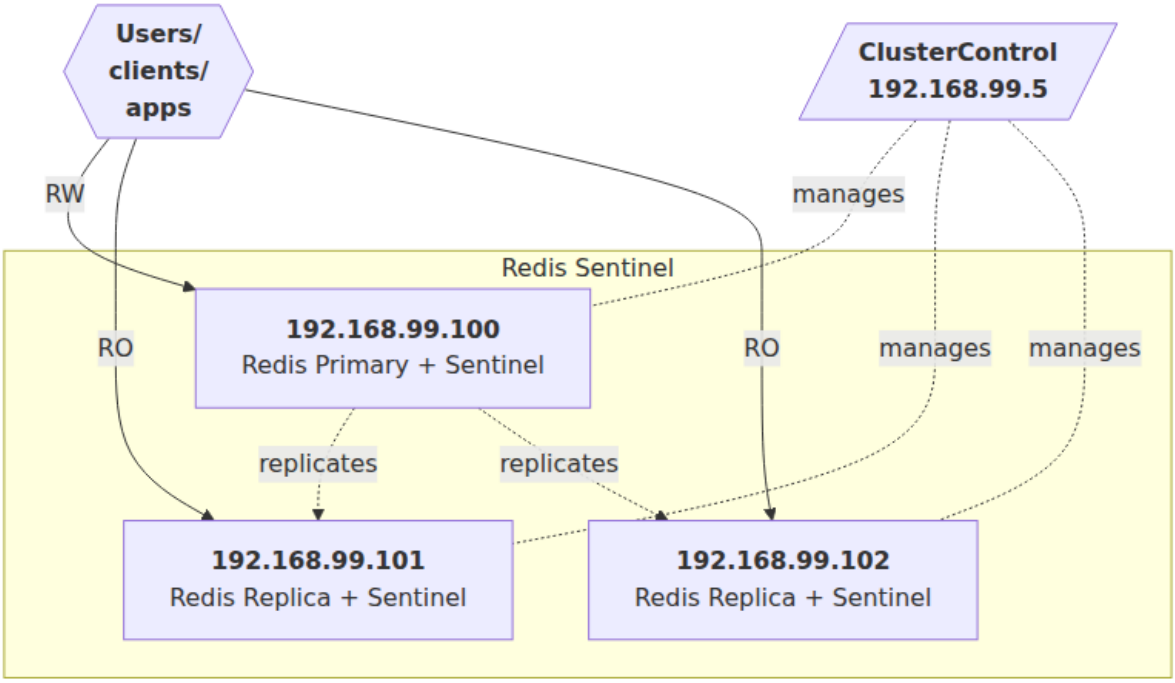
### 1.70.1 Prerequisites

Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quick-start* or use the *Installer Script*.
- You need at least three hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.70.2 Architecture

Below is a simplified diagram of the final architecture:




### 1.70.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key:

```
ssh-keygen -t rsa
```

2. Copy the SSH key to all Redis nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.100  # Primary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.101  # Replica 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.102  # Replica 2
```

3. Test the connection:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
```

### 1.70.4 Step 2: Deploy a Redis Sentinel Cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose **Redis Sentinel** from the **Database** dropdown and the **Vendor** and **Version** (e.g., "Redis" and "7.0") from the wizard. Click **Continue**.
4. In the **Deploy Redis Sentinel cluster wizard**, configure the cluster as follows:

#### (1) Cluster details

- **Name**: My Redis Sentinel Cluster
- **Tags**: redis, sentinel, ha

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: Enabled
- **Disable firewall**: Checked
- **Disable SELinux/AppArmor**: Checked

#### (3) Node configuration

- **Redis Port**: `6379` (default)
- **Redis Sentinel Port**: `26379` (default)
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: `On` (default)
- **Authentication**
  - **Password**: Set or copy the generated password for your reference.

#### (4) Add nodes

- **Primary node**:
  - Specify "192.168.99.100" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.99.101" as replica and press ++enter++.

- Specify “192.168.99.102” as another replica and press ++enter++.

- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. Click **Finish** to begin the deployment.
6. ClusterControl will start to provision the hosts, install Redis, configure replication, and install Sentinel. Monitor the progress from **Activity Center → Jobs**.

### 1.70.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.

### 1.70.6 Step 4: Connecting to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- The primary Redis server 192.168.99.100 on port 6379. Since by default ClusterControl enforces Redis instance with TLS and authentication, you need to get the key and certificate (commonly under `/var/lib/redis` of the database node) and use the password specified during deployment with the default username “admin” to access.
- For read-only workloads, you may send the database connections to the replica nodes, 192.168.99.101 or 192.168.99.102 on port 6379.

**Note**

Note Most of the Redis drivers and application connectors come with a built-in support for Redis replication, clusters and Sentinel. Check your application connector's documentation for details.

**Tip**

You can use popular Redis database management clients like phpRedisAdmin or Redis Insight to perform database administration tasks.

### 1.70.7 Step 5: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For Redis replication with Sentinel, ClusterControl supports both RDB and AOF backups. To schedule a backup:

1. Go to ***ClusterControl GUI → choose the cluster → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily full backup - Redis
- **Cluster**: (leave as it is)
- **Backup method**: (leave as it is)
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Use PIGZ for parallel gzip**: Off
- **Enable encryption**: On
- **Retention**: On (default)

#### (3) Storage

- **Storage location**: Store on controller (default)
- **Storage directory**: /root/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size and perform restoration if necessary.

### 1.70.8 Step 6: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.70.9 Step 7: Manage your cluster

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform rolling upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → choose the cluster → Actions → Upgrades***.

- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Database user and access control management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Access control***.
- Backup management: Create, schedule, restore, upload backup to cloud, and set retention period. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions_** and **ClusterControl GUI → choose the cluster → Backups → More_**.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.70.10 Step 8: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. For Redis replication wiht Sentinel, you can scale out the cluster with another replica. Common use cases are to use the replica node for to scale out the read-only workloads, backups, reporting or analytics purposes.

Adding a new replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node due to initial RDB syncing from the primary.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → choose the cluster → Actions → Add new → Replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 6379
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press ++enter++.

#### (3) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

### 1.70.11 Conclusion

Congratulations! You've successfully deployed, monitored, managed and scaled your first Redis replication cluster with Sentinel using ClusterControl. From here, you can explore additional features such as failover management, access control list (ACL) rules, advanced monitoring, advanced backup management, database restore and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant.

## 1.71 Valkey Replication with Sentinel

In this guide, we will walk you through setting up your first database cluster running on Valkey replication with Sentinel using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

Valkey is an open-source, in-memory key-value datastore designed for ultra-low-latency reads and writes, combining the roles of a cache, message broker, and primary database in a single engine. Originally forked from Valkey in 2024 (from Redis 7.2.4) under the stewardship of the Linux Foundation, Valkey retains a BSD license and is implemented in C for Unix-like systems, ensuring both performance and broad compatibility

With built-in replication, automatic failover via Valkey Sentinel, and sharding in Valkey Cluster, it can scale seamlessly from a single-server prototype to a highly available, distributed system serving millions of operations per second.

**See also**

- Valkey Documentation
- Valkey Documentation - Sentinel

### 1.71.1 Prerequisites

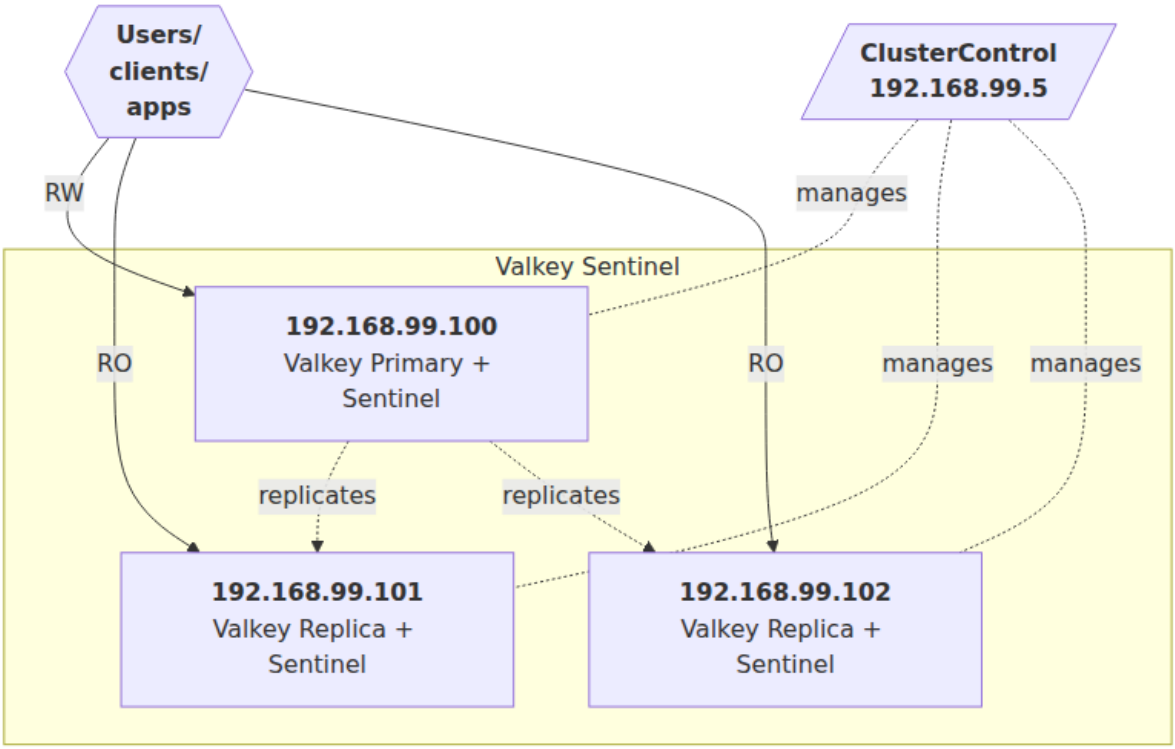
Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quickstart* or use the *Installer Script*.
- You need at least three hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.

- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.71.2 Architecture

Below is a simplified diagram of the final architecture:




### 1.71.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key:

```
ssh-keygen -t rsa
```

2. Copy the SSH key to all Valkey nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.100  # Primary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.101  # Replica 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.102  # Replica 2
```

3. Test the connection:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
```

### 1.71.4 Step 2: Deploy a Valkey Sentinel Cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.

3. Choose **Valkey Sentinel** from the **Database** dropdown and the **Vendor** and **Version** (e.g., "Valkey" and "7.0") from the wizard. Click **Continue**.
4. In the **Deploy Valkey Sentinel cluster wizard**, configure the cluster as follows:

### (1) Cluster details

- **Name**: My Valkey Sentinel Cluster
- **Tags**: valkey, sentinel, ha

### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: Enabled
- **Disable firewall**: Checked
- **Disable SELinux/AppArmor**: Checked

### (3) Node configuration

- **Valkey Port**: 6379 (default)
- **Valkey Sentinel Port**: 26379 (default)
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: On (default)
- **Authentication**
  - **Password**: Set or copy the generated password for your reference.

### (4) Add nodes

- **Primary node**:
  - Specify "192.168.99.100" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.99.101" as replica and press ++enter++.
  - Specify "192.168.99.102" as another replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. Click **Finish** to begin the deployment.
6. ClusterControl will start to provision the hosts, install Valkey, configure replication, and install Sentinel. Monitor the progress from **Activity Center → Jobs**.

### 1.71.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.

### 1.71.6 Step 4: Connecting to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- The primary Valkey server 192.168.99.100 on port 6379. Since by default ClusterControl enforces Valkey instance with TLS and authentication, you need to get the key and certificate (commonly under `/var/lib/valkey` of the database node) and use the password specified during deployment with the default username "admin" to access.
- For read-only workloads, you may send the database connections to the replica nodes, 192.168.99.101 or 192.168.99.102 on port 6379.

**Note**

Note Most of the Valkey drivers and application connectors come with a built-in support for Valkey replication, clusters and Sentinel. Check your application connector's documentation for details.

### 1.71.7 Step 5: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For Valkey replication with Sentinel, ClusterControl supports both RDB and AOF backups. To schedule a backup:

1. Go to ***ClusterControl GUI → choose the cluster → Backups*** to open the backup overview page of the cluster.

2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

### (1) Configuration

- **Schedule name**: Daily full backup - Valkey
- **Cluster**: (leave as it is)
- **Backup method**: (leave as it is)
- **Upload backup to cloud**: Off

### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Use PIGZ for parallel gzip**: Off
- **Enable encryption**: On
- **Retention**: On (default)

### (3) Storage

- **Storage location**: Store on controller (default)
- **Storage directory**: /root/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size and perform restoration if necessary.

### 1.71.8 Step 6: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.71.9 Step 7: Manage your cluster

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform rolling upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → choose the cluster → Actions → Upgrades***.
- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Database user and access control management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Access control***.
- Backup management: Create, schedule, restore, upload backup to cloud, and set retention period. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions_** and **ClusterControl GUI → choose the cluster → Backups → More_**.

- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.71.10 Step 8: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. For Valkey replication wiht Sentinel, you can scale out the cluster with another replica. Common use cases are to use the replica node for to scale out the read-only workloads, backups, reporting or analytics purposes.

Adding a new replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node due to initial RDB syncing from the primary.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → choose the cluster → Actions → Add new → Replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 6379
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press ++enter++.

#### (3) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

### 1.71.11 Conclusion

Congratulations! You've successfully deployed, monitored, managed and scaled your first Valkey replication cluster with Sentinel using ClusterControl. From here, you can explore additional features such as failover management, access control list (ACL) rules, advanced monitoring, advanced backup management, database restore and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant.

## 1.72 Redis Cluster

In this guide, we will walk you through setting up your first database cluster running on Redis Cluster using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

Redis is a high-performance, in-memory data store used for caching, key-value storage, messaging, and real-time data processing. It delivers microsecond-level speed because all data is stored in RAM. Redis supports flexible data structures such as strings, lists, hashes, sets, sorted sets, streams, and JSON (via modules), making it useful for workloads like session storage, API rate limiting, leaderboards, and real-time analytics.

In standalone mode, Redis runs with a single primary node (and optional replicas). This setup is simple but has limitations: write operations cannot scale beyond a single server, all data resides on one machine, and failover requires external tooling like Sentinel.

Redis Cluster addresses these limitations by distributing data across multiple nodes using hash-slot sharding, enabling horizontal scaling and higher throughput. It also provides built-in high availability through automatic failover, where replicas can promote themselves when a primary fails. With its decentralized architecture, Redis Cluster eliminates single points of failure and is suitable for large, production-grade deployments.

**See also**

- Redis Community Edition - Getting Started
- Redis Cluster Specification

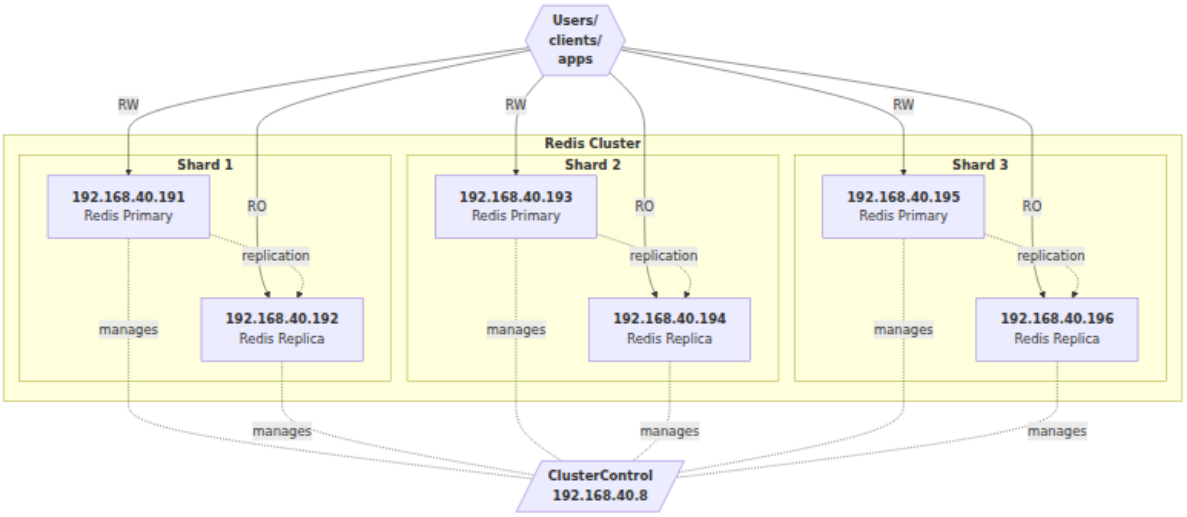
### 1.72.1 Prerequisites

Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quickstart* or use the *Installer Script*.
- You need at least three hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.72.2 Architecture

Below is a simplified diagram of the final architecture:




### 1.72.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key:

```
ssh-keygen -t rsa
```

2. Copy the SSH key to all Redis nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.191  # Primary 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.192  # Replica 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.193  # Primary 2
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.194  # Replica 2
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.195  # Primary 3
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.196  # Replica 3
```

3. Test the connection:

```
ssh root@192.168.40.191 "ls /root"
ssh root@192.168.40.192 "ls /root"
ssh root@192.168.40.193 "ls /root"
ssh root@192.168.40.194 "ls /root"
ssh root@192.168.40.195 "ls /root"
ssh root@192.168.40.196 "ls /root"
```

### 1.72.4 Step 2: Deploy a Redis Cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose **Redis Cluster** from the **Database** dropdown and the **Vendor** and **Version** (e.g., "Redis" and "8") from the wizard. Click **Continue**.
4. In the **Deploy Redis Cluster** cluster wizard, configure the cluster as follows:

### (1) Cluster details

- **Name**: My Redis Cluster
- **Tags**: redis, first-deployment

### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: Enabled
- **Disable firewall**: Checked
- **Disable SELinux/AppArmor**: Checked

### (3) Node configuration

- **Redis Port**: `6379` (default)
- **Cluster bus port **: `16379` (default)
- **Node timeout (ms)**: `15000` (default)
- **Replica validity factor**: `10` (default)
- **Configuration template**: `redis8-tls.conf` (default)
- **Repository**: Use vendor repositories (default)
- **Username**: Specify the Redis admin username. ClusterControl will create this user if not exists.
- **Password**: Specify the password for the Redis admin user.

### (4) Shard 1

- **Primary node**:
  - Specify “192.168.40.191” as the primary node and press ++enter++.
- **Replica node**:
  - Specify “192.168.40.192” as replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Click on **Add another shard**. This will add another step to add the second shard.

### (5) Shard 2

- **Primary node**:
  - Specify “192.168.40.193” as the primary node and press ++enter++.
- **Replica node**:
  - Specify “192.168.40.194” as replica and press ++enter++.

- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Click on **Add another shard**. This will add another step to add the third shard.

#### (6) Shard 3

- **Primary node**:
  - Specify "192.168.40.195" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.40.196" as replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Click **Continue**.

#### (7) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. Click **Finish** to begin the deployment.
6. ClusterControl will start to provision the hosts, install Redis, and configure replication. Monitor the progress from **Activity Center → Jobs**.

### 1.72.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.

### 1.72.6 Step 4: Connecting to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- Any primary Redis server (192.168.40.191, 192.168.40.193 or 192.168.40.195) on port 6379 for read-write workloads. In cluster-mode, the connection will be transparently follow redirects, even though the particular Redis node does not own the cluster slot.
- For read-only workloads, you may send the database connections to the replica nodes (192.168.40.192, 192.168.40.194 or 192.168.40.196) on port 6379.

- Since by default ClusterControl enforces Redis instance with TLS and authentication, you need to get the key and certificate (commonly under `/var/lib/redis` of the database node) and use the password specified during the deployment.

**Note**

Note Most of the Redis drivers and application connectors come with a built-in support for Redis replication, clusters and Sentinel. Check your application connector's documentation for details.

**Tip**

You can use popular Redis database management clients like phpRedisAdmin or Redis Insight to perform database administration tasks.

### 1.72.7 Step 5: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For Redis Cluster, ClusterControl supports both RDB and AOF backups. To schedule a backup:

1. Go to ***ClusterControl GUI → choose the cluster → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily full backup - Redis
- **Cluster**: (leave as it is)
- **Backup method**: (leave as it is)
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Use PIGZ for parallel gzip**: Off
- **Enable encryption**: On
- **Retention**: On (default)

#### (3) Storage

- **Storage location**: Store on controller (default). You can choose the storage host from the drop down list.
- **Storage directory**: /root/backups (default)

- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size and perform restoration if necessary.

### 1.72.8 Step 6: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.72.9 Step 7: Manage your cluster

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform rolling upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → choose the cluster → Actions → Upgrades***.
- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Database user and access control management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Access control***.
- Backup management: Create, schedule, restore, upload backup to cloud, and set retention period. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions_** and **ClusterControl GUI → choose the cluster → Backups → More_**.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.72.10 Step 8: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. For Redis Cluster, you can scale out the cluster with another replica. Common use cases are to use the replica node for to scale out the read-only workloads, backups, reporting or analytics purposes.

Adding a new replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node due to initial RDB syncing from the primary.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → choose the cluster → Actions → Add new → Replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 6379
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press ++enter++.

#### (3) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

### 1.72.11 Conclusion

Congratulations! You've successfully deployed, monitored, managed and scaled your first Redis Cluster using ClusterControl. From here, you can explore additional features such as failover management, access control list (ACL) rules, advanced monitoring, advanced backup management, database restore and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant.

## 1.73 Valkey Cluster

In this guide, we will walk you through setting up your first database cluster running on Valkey Cluster using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

Valkey is an open-source, community-driven, in-memory key-value datastore designed for ultra-low-latency reads and writes, combining the roles of a cache, message broker, and primary database in a single engine. Originally forked from Valkey in 2024 (from Redis 7.2.4) under the stewardship of the Linux Foundation, Valkey retains a BSD license and is implemented in C for Unix-like systems, ensuring both performance and broad compatibility. It is essentially a drop-in replacement for Redis, created after Redis changed its license.

With built-in replication, automatic failover via Valkey Sentinel, and sharding in Valkey Cluster, it can scale seamlessly from a single-server prototype to a highly available, distributed system serving millions of operations per second.

With Valkey Cluster, it provides a way to run a Valkey installation where data is automatically sharded across multiple Valkey nodes. Valkey Cluster also provides some degree of availability during partitions, to continue operations when some nodes fail or are unable to communicate. However, the cluster will become unavailable in the event of larger failures (for example, when the majority of primaries are unavailable).

So, with Valkey Cluster, you get the ability to:

- Automatically split your dataset among multiple nodes.
- Continue operations when a subset of the nodes are experiencing failures or are unable to communicate with the rest of the cluster.

**See also**

- Valkey Documentation
- Valkey Documentation - Cluster

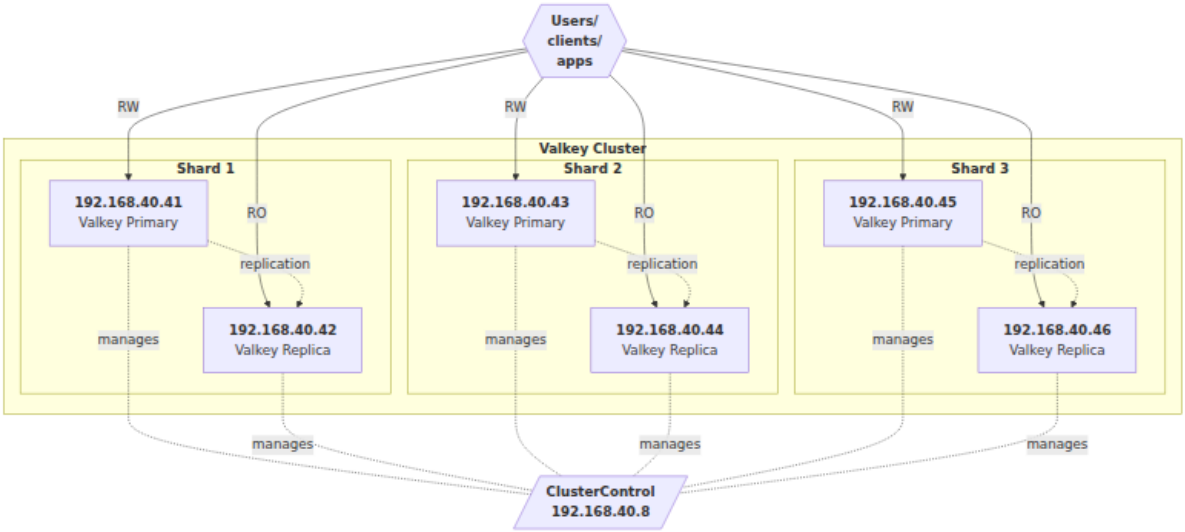
### 1.73.1 Prerequisites

Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quick-start* or use the *Installer Script*.
- You need at least three hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.73.2 Architecture

Below is a simplified diagram of the final architecture:




### 1.73.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key:

```
ssh-keygen -t rsa
```

2. Copy the SSH key to all Valkey nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.41  # Primary 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.42  # Replica 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.43  # Primary 2
```

(continues on next page)

(continued from previous page)

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.44  # Replica 2
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.45  # Primary 3
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.40.46  # Replica 3
```

3. Test the connection:

```
ssh root@192.168.40.41 "ls /root"
ssh root@192.168.40.42 "ls /root"
ssh root@192.168.40.43 "ls /root"
ssh root@192.168.40.44 "ls /root"
ssh root@192.168.40.45 "ls /root"
ssh root@192.168.40.46 "ls /root"
```

### 1.73.4 Step 2: Deploy a Valkey Cluster

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose **Valkey Cluster** from the **Database** dropdown and the **Vendor** and **Version** (e.g., "Valkey" and "8") from the wizard. Click **Continue**.
4. In the **Deploy Valkey Cluster** wizard, configure the cluster as follows:

#### (1) Cluster details

- **Name**: My Valkey Cluster
- **Tags**: Valkey, first-deployment

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: Enabled
- **Disable firewall**: Checked
- **Disable SELinux/AppArmor**: Checked

#### (3) Node configuration

- **Valkey Port**: `6379` (default)
- **Cluster bus port **: `16379` (default)
- **Node timeout (ms)**: `15000` (default)
- **Replica validity factor**: `10` (default)
- **Configuration template**: `valkey8-tls.conf` (default)

- **Repository**: Use vendor repositories (default)
- **Username**: Specify the Valkey admin username. ClusterControl will create this user if not exists.
- **Password**: Specify the password for the Valkey admin user.

### (4) Shard 1

- **Primary node**:
  - Specify "192.168.40.41" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.40.42" as replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Click on **Add another shard**. This will add another step to add the second shard.

### (5) Shard 2

- **Primary node**:
  - Specify "192.168.40.43" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.40.44" as replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Click on **Add another shard**. This will add another step to add the third shard.

### (6) Shard 3

- **Primary node**:
  - Specify "192.168.40.45" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "192.168.40.46" as replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

### (7) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

5. Click **Finish** to begin the deployment.
6. ClusterControl will start to provision the hosts, install Valkey, and configure replication. Monitor the progress from **Activity Center → Jobs**.

### 1.73.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm but the recovery procedures will be skipped.
- Topology viewer: Roll over on each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.

### 1.73.6 Step 4: Connecting to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- Any primary Valkey server (192.168.40.41, 192.168.40.43 or 192.168.40.45) on port 6379 for read-write workloads. In cluster-mode, the connection will be transparently follow redirects, even though the particular Valkey node does not own the cluster slot.
- For read-only workloads, you may send the database connections to the replica nodes (192.168.40.42, 192.168.40.44 or 192.168.40.46) on port 6379.
- Since by default ClusterControl enforces Valkey instance with TLS and authentication, you need to get the key and certificate (commonly under `/var/lib/valkey` of the database node) and use the password specified during the deployment.

**Note**

Note Most of the Valkey drivers and application connectors come with a built-in support for Valkey replication, clusters and Sentinel. Check your application connector's documentation for details.

### 1.73.7 Step 5: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For Valkey Cluster, ClusterControl supports both RDB and AOF backups. To schedule a backup:

1. Go to ***ClusterControl GUI → choose the cluster → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily full backup - Valkey
- **Cluster**: (leave as it is)
- **Backup method**: (leave as it is)
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Compression level**: 6
- **Use PIGZ for parallel gzip**: Off
- **Enable encryption**: On
- **Retention**: On (default)

#### (3) Storage

- **Storage location**: Store on controller (default). You can choose the storage host from the drop down list.
- **Storage directory**: /root/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size and perform restoration if necessary.

### 1.73.8 Step 6: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server.

You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group where you are belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with summarized warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.73.9 Step 7: Manage your cluster

ClusterControl provides many other features for managing your cluster, including:

- Database upgrades and patching: Perform rolling upgrades to keep your cluster up to date without downtime. This feature is available at ***ClusterControl GUI → choose the cluster → Actions → Upgrades***.
- Database node management: Perform start/stop/restart node, reboot host, bootstrap cluster, activate/deactivate read-only, resync node, change primary source and rebuild replica. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Database user and access control management: Manage database users and privileges globally. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Access control***.
- Backup management: Create, schedule, restore, upload backup to cloud, and set retention period. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions_** and **ClusterControl GUI → choose the cluster → Backups → More_**.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.73.10 Step 8: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. For Valkey Cluster, you can scale out the cluster with another replica. Common use cases are to use the replica node for to scale out the read-only workloads, backups, reporting or analytics purposes.

Adding a new replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however expect increasing load on the source node due to initial RDB syncing from the primary.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → choose the cluster → Actions → Add new → Replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 6379
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Add node

- **Primary node**: Choose one of the primary node from the dropdown.
- **Node**: Specify the IP address or hostname of the node that you want to add and press ++enter++.

#### (3) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

### 1.73.11 Conclusion

Congratulations! You've successfully deployed, monitored, managed and scaled your first Valkey Cluster using ClusterControl. From here, you can explore additional features such as failover management, access control list (ACL) rules, advanced monitoring, advanced backup management, database restore and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant. Valkey

## 1.74 SQL Server standalone

In this guide, we will walk you through setting up your first database cluster running on SQL Server standalone using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

SQL Server is a relational database management system (RDBMS). Applications and tools connect to a SQL Server instance or database, and communicate using Transact-SQL (T-SQL).

**See also**

- SQL Server on Linux

---

### 1.74.1 Prerequisites

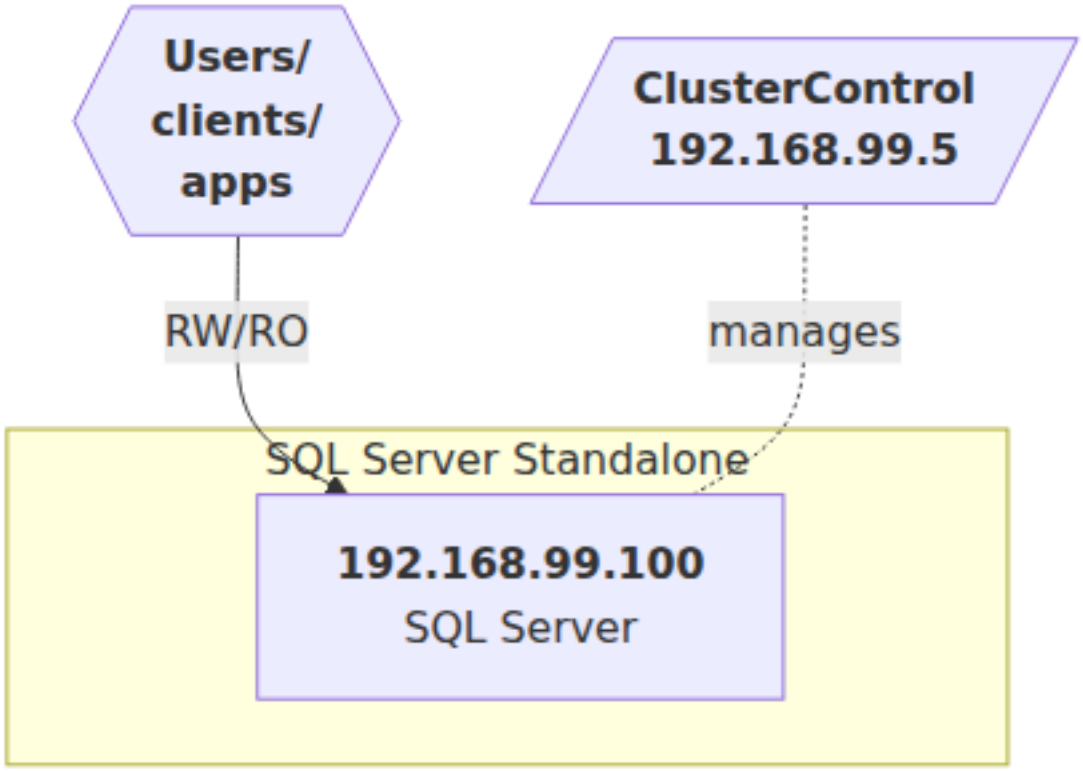
Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quickstart* or use the *Installer Script*.
- You need one host (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.
- SSH access to the server involved.
- The node must have internet access during the deployment stage.

---

### 1.74.2 Architecture

Below is a simplified diagram of the final architecture:




### 1.74.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key:

```
ssh-keygen -t rsa
```

2. Copy the SSH key to the SQL Server node:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.100
```

3. Test the connection:

```
ssh root@192.168.99.100 "ls /root"
```

4. Host mapping - The first prerequisite is updating the hosts file in all the nodes, including the ClusterControl node, with the hostnames and IP addresses of the database nodes that will participate in the availability group. The following is the expected content of `/etc/hosts` on every host:

```
127.0.1.1 ClusterControl
127.0.0.1 localhost

# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.99.100 SQL-Server
```

### 1.74.4 Step 2: Deploy SQL Server standalone

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose **SQL Server** from the **Database** dropdown and the **Vendor** and **Version** (e.g., "Microsoft" and "2022") from the wizard. Click **Continue**.
4. In the **Deploy SQL Server cluster**, configure the cluster as follows:

#### (1) Cluster details

- **Name**: My SQL Server 2022 Cluster
- **Tags**: sql, ag,

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: Enabled
- **Disable firewall**: Checked
- **Disable SELinux/AppArmor**: Checked

#### (3) Node configuration

- **Admin username**: `SQLServerAdmin` (default)
- **Admin username**: Set or copy the generated password for your reference.
- **Server Port**: `1433` (default)
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: `On` (default)

#### (4) Add nodes

- **Primary node**: (hostname is required as IP is not supported)
  - Specify "SQL-Server" as the primary node and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if the node is reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

6. Click **Finish** to begin the deployment.
7. ClusterControl will start to provision the host and install SQL Server. Monitor the progress from **Activity Center → Jobs**.

---

### 1.74.5 Step 3: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of the node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View the most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm, but the recovery procedures will be skipped.
- Topology viewer: Roll over each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.

---

### 1.74.6 Step 4: Connecting to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- The SQL Server 192.168.99.100 is on port 1433. We can use SQL Server Management Studio (SSMS), Azure Data Studio, or the SQLCMD tool to connect to the SQL Server. For the SSMS connection, authentication, choose SQL Server Authentication, and use the SQL credentials specified during deployment.

**Tip**

You can use popular SQL Server Management Studio (SSMS), Azure Data Studio, or SQLCMD tool to perform database administration tasks.

---

### 1.74.7 Step 5: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For SQL Server standalone, ClusterControl supports full, differential, and transaction log backups. To schedule a backup:

1. Go to ***ClusterControl GUI → choose the cluster → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily full backup - SQL Server
- **Cluster**: (leave as it is)
- **Backup method**: (choose between full, differential, and transaction log)
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Retention**: On (default)
- **Include system databases**: On (default)

#### (3) Storage

- **Storage location**: Store on node (default)
- **Storage directory**: /var/lib/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size, and perform restoration if necessary.

---

### 1.74.8 Step 6: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks, and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however, a mail transfer agent (sendmail, postfi,x or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group that you belong to from the **User group** dropdown.
3. Choose your email address to from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with a summary of warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie), or web hooks. See *Integration → Notification Services*.

---

### 1.74.9 Step 7: Manage your cluster

ClusterControl provides many other features for managing your cluster, including:

- Database node management: Perform start/stop/restart node, reboot host. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Backup management: Create, schedule, restore, upload backup to the cloud, and set a retention period. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions**_ and **ClusterControl GUI → choose the cluster → Backups → More**_.
- Maintenance management: Activate, deactivate, remark, and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from the ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

---

### 1.74.10 Conclusion

Congratulations! You've successfully deployed, monitored, managed, and scaled your first SQL Server Standalone using ClusterControl. From here, you can explore additional features such as failover management, access control list (ACL) rules, advanced monitoring, advanced backup management, database restore, and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant.

## 1.75 SQL Server Always On availability group

In this guide, we will walk you through setting up your first database cluster running on SQL Server Always On availability group using ClusterControl. By the end of this tutorial, you will have a fully operational database cluster that you can monitor, manage, and scale using ClusterControl.

SQL Server is a relational database management system (RDBMS). Applications and tools connect to a SQL Server instance or database, and communicate using Transact-SQL (T-SQL).

The Always On availability groups feature is a high-availability and disaster-recovery solution that provides an enterprise-level alternative to database mirroring. Always On availability groups maximize the availability of a set of user databases for an enterprise. An availability group supports a failover environment for a discrete set of user databases, known as availability databases, that fail over together. An availability group supports a set of read-write primary databases and one to eight sets of corresponding secondary databases. Optionally, secondary databases can be made available for read-only access and/or some backup operations.

**See also**

- SQL Server on Linux
- SQL Server Always On availability group

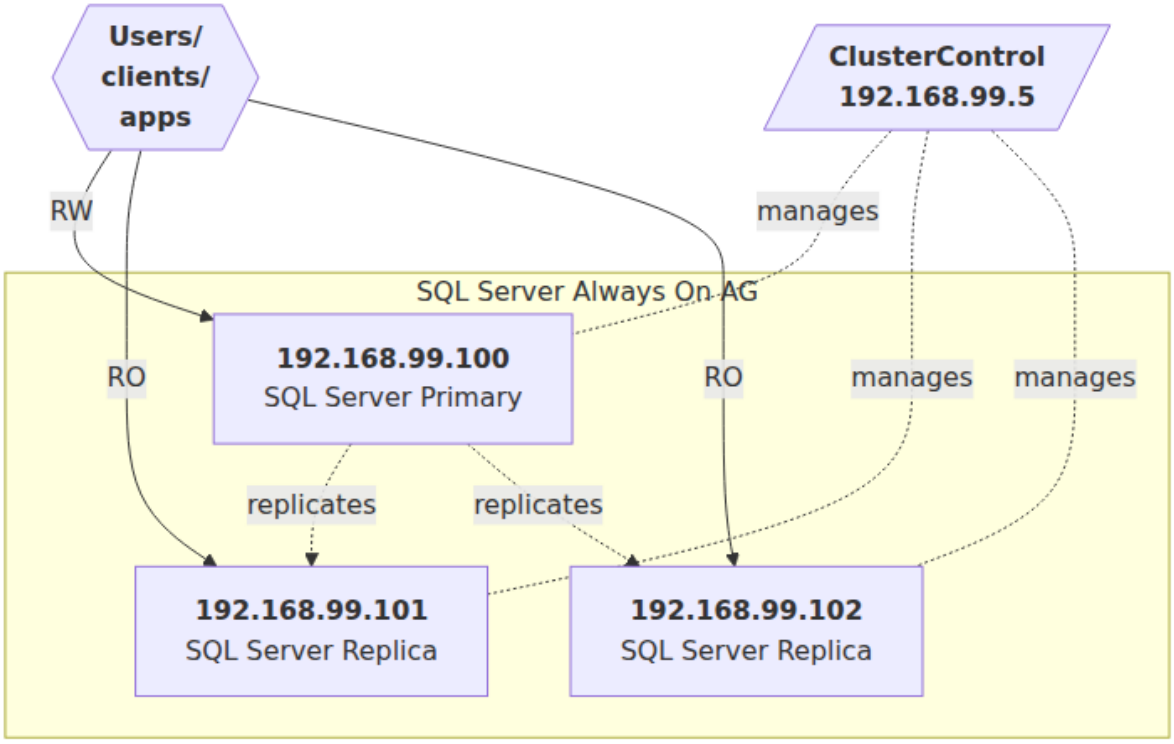
### 1.75.1 Prerequisites

Ensure the following requirements are met before proceeding:

- ClusterControl should be installed and running. If not, follow the installation instructions in *Quickstart* or use the *Installer Script*.
- ClusterControl only supports deploying and managing SQL Server for Linux.
- You need at least three hosts (bare-metal or virtual machines) installed with the supported operating system. See *Operating System*.
- SSH access to all servers involved.
- All nodes must have internet access during the deployment stage.
- NTP is configured and running on all hosts to ensure that all nodes in the cluster maintain synchronized time.

### 1.75.2 Architecture

Below is a simplified diagram of the final architecture:




### 1.75.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key:

```
ssh-keygen -t rsa
```

2. Copy the SSH key to all SQL Server nodes:

```
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.100  # Primary
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.101  # Replica 1
ssh-copy-id -i ~/.ssh/id_rsa root@192.168.99.102  # Replica 2
```

3. Test the connection:

```
ssh root@192.168.99.100 "ls /root"
ssh root@192.168.99.101 "ls /root"
ssh root@192.168.99.102 "ls /root"
```

4. Host mapping - The first prerequisite is updating the hosts file in all the nodes, including the ClusterControl node, with the hostnames and IP addresses of the database nodes that will participate in the availability group. The following is the expected content of `/etc/hosts` on every host:

```
127.0.1.1 ClusterControl
127.0.0.1 localhost

# The following lines are desirable for IPv6 capable hosts
```

(continues on next page)

(continued from previous page)

```
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.99.100 SQL-primary                # Hostname for node1
192.168.99.101 SQL-replica1                # Hostname for node2
192.168.99.102 SQL-replica2                # Hostname for node3
```

### 1.75.4 Step 2: Deploy a SQL Server Always On availability group

1. Open your browser and navigate to your ClusterControl server's IP or domain name.
2. From the ClusterControl dashboard, click on **Deploy a cluster** in the top-right corner and choose **Create a database cluster**.
3. Choose **SQL Server** from the **Database** dropdown and the **Vendor** and **Version** (e.g., "Microsoft" and "2022") from the wizard. Click **Continue**.
4. In the **Deploy SQL Server cluster**, configure the cluster as follows:

#### (1) Cluster details

- **Name**: My SQL Server 2022 Cluster
- **Tags**: sql, ag,

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa`
- **SSH port**: 22
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: Enabled
- **Disable firewall**: Checked
- **Disable SELinux/AppArmor**: Checked

#### (3) Node configuration

- **Admin username**: `SQLServerAdmin` (default)
- **Admin username**: Set or copy the generated password for your reference.
- **Server Port**: `1433` (default)
- **Repository**: Use vendor repositories (default)
- **Enable SSL encryption**: `On` (default)

#### (4) Add nodes

- **Primary node**: (hostname is required as IP is not supported)
  - Specify "SQL-primary" as the primary node and press ++enter++.
- **Replica node**:
  - Specify "SQL-replica1" as replica and press ++enter++.
  - Specify "SQL-replica2" as another replica and press ++enter++.
- Wait until everything is green. If a red warning appears, inspect the error and fix it.
- Proceed only if all nodes are reachable (shown in green).

#### (5) Preview

- Review the summary of your deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

6. Click **Finish** to begin the deployment.
7. ClusterControl will start to provision the hosts, install SQL Server, and configure the Always On availability group. Monitor the progress from **Activity Center → Jobs**.

### 1.75.5 Step 3: Connecting to the database cluster

At this point, your application should be able to access the database cluster via the following endpoints:

- The primary SQL Server is 192.168.99.100 on port 1433. We can use SQL Server Management Studio (SSMS), Azure Data Studio, or the `sqlcmd` tool to connect to the SQL Server. For the SSMS connection, authentication, choose "SQL Server Authentication", and use the database credentials specified during the deployment.
- If you want to connect to the SQL server locally in the server, SSH to the server and use the following command:

```
/opt/mssql-tools18/bin/sqlcmd -U SQLServerAdmin -P '
↪<SQLServerAdmin_password>' -C
```

- For read-only workloads, you may send the database connections to the replica nodes, 192.168.99.101 or 192.168.99.102, on port 1433.

**Note**

Note Most of the SQL Server drivers and application connectors come with a built-in support for Always On availability group. Check your application connector's documentation for details.

### 1.75.6 Step 4: Create a database and add into availability group

When a new database is created on the primary server, it is not automatically included in the Always On availability group and, therefore, not replicated to the replicas. Replication to a replica only occurs after the database has been added to the Always On availability group. ClusterControl, by default, establishes an availability group named `s9s_ag1`. This step outlines the process of creating a database and subsequently adding it to the `s9s_ag1` availability group to enable replication to the replica servers.

1. SSH to the primary SQL server and connect to the SQL Server by using `sqlcmd`:

```
/opt/mssql-tools18/bin/sqlcmd -U SQLServerAdmin -P '
↪<SQLServerAdmin_password>' -C
```

2. Create a new database called `myshop`:

```
CREATE DATABASE [myshop]
GO
```

3. Go to the **Backups** tab and click **Create Backup** to take a FULL database backup, SQL Server requires a valid transaction log chain and recovery baseline for replication and synchronization.
4. Add the database into the Availability Group:

```
ALTER AVAILABILITY GROUP [s9s_ag1]
ADD DATABASE [myshop]
GO
```

5. Verify that the database is inside the Availability Group:

```
SELECT * FROM [master].[sys].[availability_databases_cluster]
GO
```

6. Verify the synchronization health and status under ***ClusterControl GUI → Clusters → choose the cluster → Nodes***.

**Tip**

Remember to perform these steps if you create a new database, and you want it to be replicated to the replica servers. For PITR operations, remove the database from the Availability Group before restore. After recovery on the Primary replica, clean old database copies from the secondary replicas before rejoining the database to ensure proper synchronization.

### 1.75.7 Step 5: Monitor your cluster

After the cluster has been deployed, you will be redirected to the **Home** page. This page provides an at-a-glance view of your cluster's status and important aspects of a database, including:

- Cluster health: The **Home** page provides the cluster state
- Node health: Rollover on the honeycomb diagram and get a more detailed view of a particular node. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: View the most recent alarms of your cluster.
- Automatic recovery status: Green means ClusterControl will raise an alarm and perform recovery procedures if it detects the database service is down. Red means ClusterControl will raise an alarm, but the recovery procedures will be skipped.
- Topology viewer: Roll over each cluster entry to see the summary of the cluster visualized in topology view to better understand your overall database state. Also available under ***ClusterControl GUI → Clusters → choose the cluster → Nodes → Topology***.

### 1.75.8 Step 6: Enable automatic backups

To ensure your cluster data is protected, you should schedule an automatic backup. For SQL Server Always On availability group, ClusterControl supports full, differential, and transaction log backups. To schedule a backup:

1. Go to ***ClusterControl GUI → choose the cluster → Backups*** to open the backup overview page of the cluster.
2. Go to ***Create Backup → Schedule a Backup*** to schedule an automated backup. It will open a backup wizard to follow.
3. In the **Create a backup schedule** wizard, configure the backup as below:

#### (1) Configuration

- **Schedule name**: Daily full backup - SQL Server
- **Cluster**: (leave as it is)
- **Backup method**: (choose between full, differential, and transaction log)
- **Upload backup to cloud**: Off

#### (2) Advanced settings

- **Compression**: On (default)
- **Retention**: On (default)
- **Include system databases**: On (default)

#### (3) Storage

- **Storage location**: Store on node (default)
- **Storage directory**: /var/lib/backups (default)
- **Backup subdirectory**: BACKUP-%I (default)

#### (4) Schedule

- **Set backup schedule**: Simple
- **Every**: day at 2:00
- **Select timezone**: Use your preferred timezone

#### (5) Preview

- Review the summary of the backup configuration. You can always go back to any previous section to modify your configurations if you wish. The backup configuration settings will be kept until you exit the wizard.

4. Click **Create** to schedule the backup.

Once configured, ClusterControl will take care of the automated backups, ensuring your data is safe. All successful backups will be listed in the ***Backups → All Backups*** page, where you can see the backup logs, size, and perform restoration if necessary.

### 1.75.9 Step 7: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks, and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail; however, a mail transfer agent (sendmail, postfix, or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Choose a user group that you belong to from the **User group** dropdown.
3. Choose your email address from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures you are always notified when critical issues arise and can act quickly to resolve them. You will also get an email with a summary of warning events every day at the configured delivery time.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie), or web hooks. See *Integration → Notification Services*.

### 1.75.10 Step 8: Manage your cluster

ClusterControl provides many other features for managing your cluster, including:

- Database node management: Perform start/stop/restart node, reboot host, promote replica, rebuild replica. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes globally or on individual database node. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Backup management: Create, schedule, restore, upload backup to the cloud, and set a retention period. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions_** and **ClusterControl GUI → choose the cluster → Backups → More_**.
- Maintenance management: Activate, deactivate, remark, and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from the ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.75.11 Step 9: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. For SQL Server Always On availability group, you can scale out the cluster with another replica. Common use cases are to use the replica node for to scale out the read-only workloads, backups, reporting, or analytics purposes.

Adding a new replica node is an online and non-blocking operation. It shall not cause downtime to the running cluster however, it will increase load on the source node due to initial syncing from the primary.

ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

1. To add a new replica, go to ***ClusterControl GUI → choose the cluster → Actions → Add new → Replica node***.
2. In the **Create a replica node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 1433
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Add node

- **Primary node**: Choose the primary node from the dropdown.
- **Node**: Specify the node's hostname that you want to add and press ++enter++.

#### (3) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new replica node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

### 1.75.12 Conclusion

Congratulations! You've successfully deployed, monitored, managed, and scaled your first SQL Server Always On availability group using ClusterControl. From here, you can explore additional features such as failover management, access control list (ACL) rules, advanced monitoring, advanced backup management, database restore, and many more. As you grow, ClusterControl will help ensure that your database infrastructure remains highly available and performant.

## 1.76 Elasticsearch Standalone Cluster

This guide will take you through the process of setting up your first standalone (also known as single-node) Elasticsearch cluster using ClusterControl. By following this tutorial, you'll have a fully functional (but non-replicated) database node that you can monitor, manage, and later scale if your requirements change.

A standalone Elasticsearch node is the simplest deployment option, but it isn't recommended for production workloads or day-to-day operations in medium- to large-scale organisations. Use it in production only for low-stakes workloads where downtime and potential data loss are acceptable.

This tutorial applies to the deployment of a database cluster for Elasticsearch by Elastics.

**See also**

- Elasticsearch: One-node clusters

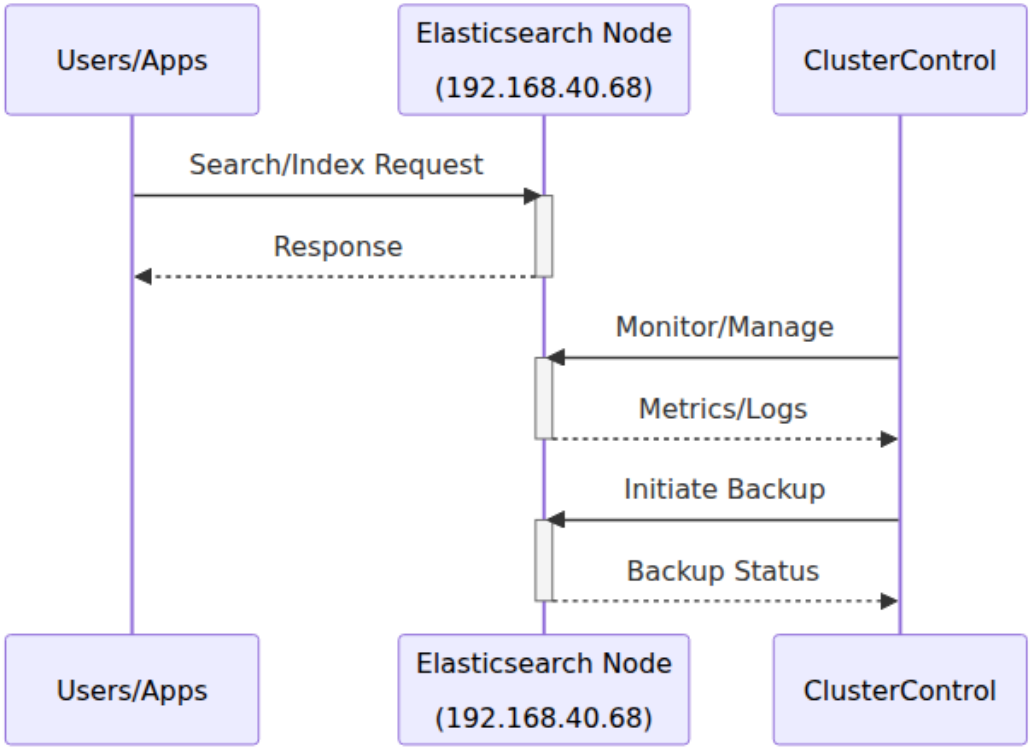
### 1.76.1 Prerequisites

Before proceeding, ensure you have:

- ClusterControl installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least two hosts (bare-metal or virtual machines):
  - One for the ClusterControl server.
  - One for the Elasticsearch single-node or server.
- SSH access to all servers.
- Internet access on the database host to install required packages.
- Network Time Protocol (NTP) configured and running on both hosts to keep their clocks synchronized.

### 1.76.2 Architecture

Below is a sequence diagram of the final architecture for a single-node deployment:




Elasticsearch in ClusterControl is deployed without any external supporting load balancers to be added. Although Elasticsearch has its load balancing and high-availability capabilities or mechanisms that can act to provide its functionality. But for a single node, it's goes to its default node type which can function as as a master node, data node, ingest node, or even a coordinating node.

### 1.76.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

Copy the public key to your standalone database node (replace 192.168.40.68 with your node's IP/hostname):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.40.68
```

If the target node uses a custom SSH key or port, you can add options:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.40.68
```

For some advance setup where the user (for example `mymainacc`) is only allowed to access non-root with only public keys are allowed and password challenge is disabled but has sudo privileges, as `root` OS user, you can do this:

```
[root@pupnode7 ~]# ssh -i /home/mymainacc/.ssh/id_rsa␣
↪mymainacc@192.168.40.68   "sudo bash -c '
umask 077
mkdir -p   /root/.ssh
```

(continues on next page)

(continued from previous page)

```
cat >>     /root/.ssh/authorized_keys
'" < ~/.ssh/id_rsa.pub
```

this copies your root public key to the target node you will be setting up for single-node Elasticsearch cluster deployment

2. Test passwordless SSH from the ClusterControl server:

```
ssh root@192.168.40.68 "stat \$PWD/"
```

Ensure there is no password prompt. If the command returns directory system status, you're set.

### 1.76.4 Step 2: Deploy a new cluster (single node)

1. Open a web browser and go to the ClusterControl server's IP or hostname.
2. On the ClusterControl dashboard, click ***Deploy a cluster (top-right) → Create a database cluster***. This opens the Deploy cluster wizard.
3. Select **Elasticsearch** from the **Database** dropdown. Specify also the version you desired to choose by clicking **Version** dropdown. Click **Continue**.
4. In the **Deploy MySQL Replication** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: For example, `ES-stand-alone-node`.
- **Tags**: (Optional) e.g., `standalone`, `production`, `dc1`.

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa` (ClusterControl will also autofill this field)
- **SSH port**: `22` (default port)
- **SSH sudo password**: (leave blank if you rely on key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **HTTP port**: `9200` (disabled text-field with default port specified)
- **Transfer port**: `9200` (disabled text-field with default port specified)
- **Admin user**: `admin`
- **Admin password**: Password to be assigned to the database `admin` user
- **Repository**: Use vendor repositories (default)

#### (4) Add node

- **Eligible master**: (Fill in the IP/hostname or FQDN of the node. For example, 192.168.40.65 and press ++enter++)
- **Data nodes**: (leave it empty)

#### (5) Snapshot storage configuration

- **Repository name**: (Fill in the repository name you want. For example, `es-s9s-repo`)
- **Storage host**: (Click the drop-down field and choose your single-node IP/hostname or FQDN entry)
- **Default snapshot location**: (Path of the shared filesystem use to store your snapshots. For example, `/mnt/data/backups/es-snapshot-repositories`)
- **Configure shared filesystem**: On (default)

#### (5) Preview

- Review your configuration. You can go back and adjust if necessary.

5. Click **Finish** to start deployment.
6. ClusterControl will now install and configure Elasticsearch standalone cluster on 192.168.40.68. You can track progress in the **Activity center**. After a few minutes, your new single-node cluster will appear on the **Home** page.

### 1.76.5 Step 3: Monitor your cluster

Once deployed, you'll see:

- Cluster health: The **Home** page provides the cluster state, even though it's a single node, it's treated as a "cluster" of size one.
- Node health: Hover on the honeycomb diagram or check the Nodes tab. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: Any triggered alarms will appear if there are configuration or resource issues.
- Automatic recovery status: If enabled, ClusterControl can attempt to restart a crashed MySQL server automatically.
- Topology viewer: You'll see a very simple topology with only one node.

### 1.76.6 Step 4: Import data

- You can import data into the standalone server in a variety of ways:

#### For bulk import, using NDJSON file

NDJSON stands for newline delimited JSON. For example, loading data from Wikimedia Foundation from here, I can load data as such:

Adding index first to the file,

```
jq -c --arg idx enwikiquote '{index:{_index:$idx,_id: .id}}, .'␣
↪first1000-enwikiquote.json > first1000-enwikiquote.ndjson
```

Then load with NDJSON

```
curl --cacert /etc/elasticsearch/certs/elasticsearch-ca.pem -u␣
↪elastic-s9s:myPassw0rd -H 'Content-Type: application/x-ndjson' --
↪data-binary @first1000-enwikiquote.ndjson -X POST
↪'https://192.168.40.68:9200/_bulk?pretty'
```

### Using `elasticdump`

Install `elasticdump` first.

For example, you have migrated from another host (in this example, its 192.168.10.100), you can dump your data using `elasticdump` as follows:

```
~/node_modules/elasticdump/bin/elasticdump \
     --cert /etc/elasticsearch/certs/elasticsearch-ca.pem \
     --input=https://elastic-s9s:myPassw0rd@192.168.10.100:9200/enwikiquote␣
↪\
     --output=/backups/enwikiquote.json --type=data \
     --limit=10000
```

Then create an empty index from your target node, in this example, its 192.168.40.65

```
curl --cacert /etc/elasticsearch/certs/elasticsearch-ca.pem \
     -u elastic-s9s:myPassw0rd \
     -H 'Content-Type: application/json' -X PUT
↪'https://192.168.40.68:9200/enwikiquote_play' \
     -d '{ "settings": { "number_of_replicas": 0 } }'
```

Then it's now ready to dump to the target node,

```
NODE_TLS_REJECT_UNAUTHORIZED=0 ~/node_modules/elasticdump/bin/elasticdump \
  --input=/backups/enwikiquote.json \
  --
↪output=https://elastic-s9s:myPassw0rd@192.168.40.68:9200/enwikiquote_play␣
↪\
  --type=data \
  --limit=10000 \
  --concurrency=4
```

```
  You can also pass the NODE_TLS_REJECT_UNAUTHORIZED=0 if you are using a␣
↪self-signed certificate
```

Always remember, when importing data to your single node, keep in mind that ClusterControl shall mark your standalone node to degraded when it detects indices health are **Yellow** whenever your indices are set to have replicas. Make sure when creating your indices during import to mark your replicas as 0 by doing the following example (where index is named `enwikiquote`):

```
curl --cacert /etc/elasticsearch/certs/elasticsearch-ca.pem -u␣
↪elastic-s9s:myPassw0rd -X PUT
↪'https://192.168.40.68:9200/enwikiquote/_settings?pretty' -H
↪'Content-Type: application/json' -d'
```

(continues on next page)

(continued from previous page)

```
{
  "index" : {
    "number_of_replicas" : 0
  }
}
'
```

### 1.76.7 Step 5: Connect to the database

Your application or client will connect directly to the standalone server:

- Host: 192.168.40.68
- Port: 9200
- User/Password: The credentials specified during deployment. You can also inspect the file `/etc/cmon.d/cmon_$CID.cnf` where $CID is the cluster id of your Elasticsearch deployment.

No load balancer or additional ports are involved.

### 1.76.8 Step 6: Enable automatic backups

Elasticsearch does not offer the traditional sense of backup. Instead, it uses snapshots which have more capability and robust way to create a consistent backup copy. It is the most recommended method for creating consistent point-in-time backups for an entire cluster or specific indices, and its native to Elasticsearch.

Even a single-node, standalone Elasticsearch cluster needs reliable backups, and ClusterControl makes it easy to schedule them automatically.

1. Go to ***ClusterControl GUI → choose the cluster → Backups***.
2. Click ***Create Backup → Schedule a Backup***. The Create a backup schedule wizard will open and proceed to configure your backup as below:

#### (1) Configuration

- **Schedule name**: `Daily ES-standalone cluster snapshot`
- **Cluster**: (defaults to your Elasticsearch standalone cluster)
- **Repository**: (defaults to the repository you created during deployment)
- **Backup method**: `elasticsearch-snapshot` (default) [disabled text field]

#### (2) Advanced settings

- **Retention**: On (default)
- **Retention [textfield]**: 4 (Set to your desired number of days to retain backup)

#### (3) Schedule

- **Set backup schedule**: Simple
- **Every**: day at `02:00`
- **Timezone**: select your local timezone

#### (4) Preview

- Verify all settings. You can go back to adjust if needed.

3. Click Create to schedule it.

ClusterControl will now automatically perform your backups. You may also restore your backups from the elasticsearch snapshots in the future if necessary for disaster recovery.

### 1.76.9 Step 7: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Select a User group (your group) from the **User group** dropdown.
3. Select your email address to from the **Users in the selected group** dropdown.
4. Click ***Enable***.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures timely notifications when something goes wrong.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.76.10 Step 8: Manage your node

ClusterControl provides monitoring for your cluster and system overview which displays workloads based on metrics. Once your Elasticsearch cluster is done deployed, agents using Prometheus exporters are deployed to gather metrics and provide more granular monitoring of your cluster and system workload for your standalone cluster in Elastitcsearch.

Apart from the monitoring, you can manage your node with the most available options for this cluster:

- Database node management: Perform start/stop/restart node, reboot host. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes globally. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.

- Backup management: Create, schedule, restore, store snapshot in an off-cluster storage location such as AWS S3 or any S3 Compatiable storage, while allowing you to set retention period for your backup snapshots. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions**_ and **ClusterControl GUI → choose the cluster → Backups → More**_.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.76.11 Conclusion

Congratulations! You've deployed, monitored, and managed a standalone Elasticsearch server using ClusterControl. Currently, a single node or standalone node in ClusterControl for Elasticsearch is limited to only one node, without any feature to scale, but make sure you follow for such updates or request for features if necessary for your requirements. While it limits you to some extent in your single node, its available functions does not stop you from doing your favorite things to do. Make sure you always practice to do the following:

- Keep your backups snapshots current and up to date.
- Monitor performance and resource usage.
- Secure your node by restricting access and using strong passwords.
- Expand to multi-node topologies if your needs grow (this time by deploying a multi-node or high-availability cluster in Elasticsearch)

Enjoy your streamlined single-node database deployment for Elasticsearch, backed by ClusterControl's powerful operations for backups, alerts, scaling, and more!

## 1.77 Elasticsearch High-Availability Cluster

This guide will take you through the process of setting up your first high-availability cluster, Elasticsearch cluster using ClusterControl. By following this tutorial, you'll have a fully functional high-availability Elasticsearch cluster that you can monitor, manage, and later scale if your requirements change.

A high-availability cluster in Elasticsearch offers you service ensuring your cluster to keep available when individual nodes, racks, or whole zones go down. A highly available cluster in Elasticsearch provides no single-point of failure (SPOF), data redundancy, automatic failover and self-healing (also resiliency), zone/rack awareness, rolling maintenance and upgrade, horizontal scalability, consistent cluster state, and durable snapshots and fast restore.

This tutorial applies to the deployment of a database cluster for Elasticsearch by Elastics.

**See also**

- Elasticsearch: Backup, high availability, and resilience tools

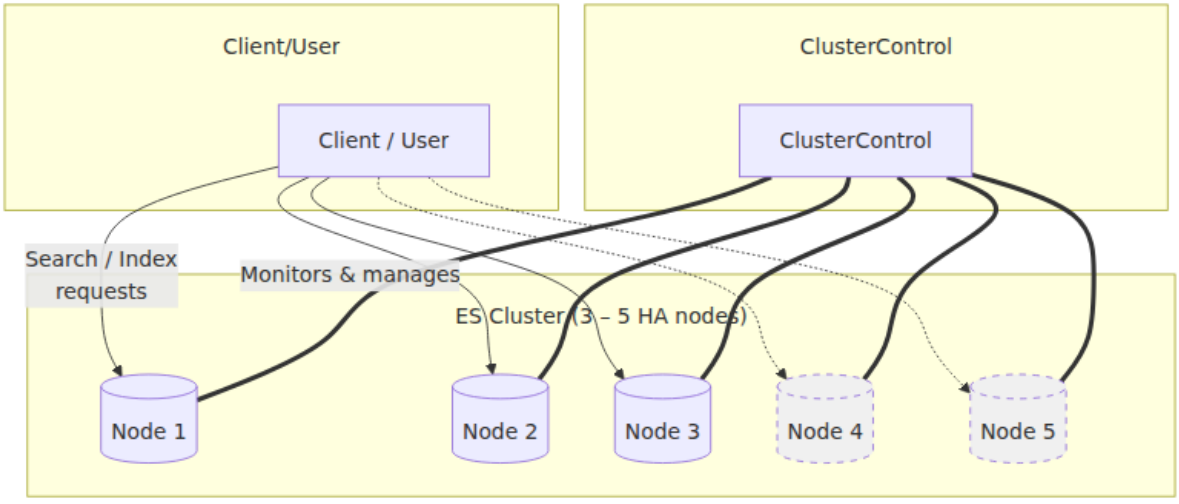
### 1.77.1 Prerequisites

Before proceeding, ensure you have:

- ClusterControl installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least two hosts (bare-metal or virtual machines):
  - One for the ClusterControl server.
  - One for the Elasticsearch single-node or server.
- SSH access to all servers.
- Internet access on the database host to install required packages.
- Network Time Protocol (NTP) configured and running on both hosts to keep their clocks synchronized.

### 1.77.2 Architecture

Below is a simple flow diagram on how shall our deployment managed and monitored with ClusterConrol while the **Client/User** interacts with our Elasticsearch nodes (or master/data nodes) emphasizing a high-availability cluster.




Elasticsearch in ClusterControl is deployed without any additional supporting load balancers. Elasticsearch has its built-in capability for load balancing but its not further discussed or is related for this topic.

On the diagram above, the nodes ES1, ES2, ES3, ES4, and ES5 shall have the following hostnames for this example deployment:

- ES1: 192.168.40.70 (master-data)
- ES2: 192.168.40.71 (master-data)
- ES3: 192.168.40.72 (master-data)

- ES4: 192.168.40.73 (data)
- ES5: 192.168.40.74 (data)

### 1.77.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

Copy the public key to your standalone database node (replace 192.168.40.68 with your node's IP/hostname):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.40.70
```

If the target node uses a custom SSH key or port, you can add options:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.40.670
```

For some advance setup where the user (for example `mymainacc`) is only allowed to access non-root with only public keys are allowed and password challenge is disabled but has sudo privileges, as `root` OS user, you can do this:

```
[root@pupnode7 ~]# ssh -i /home/mymainacc/.ssh/id_rsa␣
↪mymainacc@192.168.40.70   "sudo bash -c '
umask 077
mkdir -p   /root/.ssh
cat >>     /root/.ssh/authorized_keys
'" < ~/.ssh/id_rsa.pub
```

this copies your root public key to the target node you will be setting up for single-node Elasticsearch cluster deployment

2. Test passwordless SSH from the ClusterControl server:

```
ssh root@192.168.40.70 "stat \$PWD/"
```

Ensure there is no password prompt. If the command returns directory system status, you're set.

3. Apply the same steps to the rest of the ndoes from nodes ES2, ES3, ES4, and ES5

### 1.77.4 Step 2: Deploy a new cluster (high-availability)

1. Open a web browser and go to the ClusterControl server's IP or hostname.
2. On the ClusterControl dashboard, click ***Deploy a cluster (top-right) → Create a database cluster***. This opens the Deploy cluster wizard.
3. Select **Elasticsearch** from the **Database** dropdown. Specify also the version you desired to choose by clicking **Version** dropdown. Click **Continue**.
4. In the **Deploy MySQL Replication** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: For example, `Elasticsearch-HA`.
- **Tags**: (Optional) e.g., `standalone`, `production`, `dc1`.

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa` (ClusterControl will also autofill this field)
- **SSH port**: `22` (default port)
- **SSH sudo password**: (leave blank if you rely on key-based auth)
- **SSH sudo / OS elevation command**: Choose either uses `sudo`(default) or `doas` or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **HTTP port**: `9200` (disabled text-field with default port specified)
- **Transfer port**: `9200` (disabled text-field with default port specified)
- **Admin user**: `admin`
- **Admin password**: Password to be assigned to the database `admin` user
- **Repository**: Use vendor repositories (default)

#### (4) Add node

- **Eligible master**: (Fill in the IP/hostname or FQDN of the master nodes. In this exercise, input ES1, ES2, and ES3)
- **Data nodes**: (Fill in the IP/hostname or FQDN of the data nodes. In this exercise, input ES4, ES5)

#### (5) Snapshot storage configuration

- **Repository name**: (Fill in the repository name you want. For example, `es-s9s-repo`)
- **Storage host**: (Click the drop-down field and choose your desired node to host the NFS shared directory using IP/hostname or FQDN entry)
- **Default snapshot location**: (Path of the shared filesystem use to store your snapshots. For example, `/mnt/data/backups/es-snapshot-repositories`)
- **Configure shared filesystem**: On (default)

#### (5) Preview

- Review your configuration. You can go back and adjust if necessary.

5. Click **Finish** to start deployment.

6. ClusterControl will now install and configure your Elasticsearch High-Availability (HA) Cluster. You can track progress in the **Activity center**. After a few minutes, your Elasticsearch HA Cluster will appear on the **Home** page.

### 1.77.5 Step 3: Monitor your cluster

Once deployed, you'll see:

- Cluster health: The **Home** page provides the cluster state.
- Node health: Hover on the honeycomb diagram or check the Nodes tab. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: Any triggered alarms will appear if there are configuration or resource issues.
- Automatic recovery status: If enabled, ClusterControl can attempt to restart a crashed MySQL server automatically.
- Topology viewer: You'll see a very simple topology with only one node.

### 1.77.6 Step 4: Import data

- You can import data into the standalone server in a variety of ways:

#### For bulk import, using NDJSON file

NDJSON stands for newline delimited JSON. For example, loading data from Wikimedia Foundation from here, I can load data as such:

Adding index first to the file,

```
        $ wget␣
↪https://dumps.wikimedia.org/other/cirrussearch/20250414/abwiki-20250414-cirrussearch-co
        $ gzip -d abwiki-20250414-cirrussearch-content.json.gz
        $ jq -nc --arg idx abwiki '
          foreach inputs as $doc (0;                # counter starts at 0
            . + 1;                                  # next counter value
            ({index:{_index:$idx,_id:.}}),           # emit meta line
            $doc                                     # emit original line
          )
        '  abwiki-20250414-cirrussearch-content.json > abwiki.ndjson
    ```
    Then load with NDJSON
    ```bash
    curl --cacert /etc/elasticsearch/certs/elasticsearch-ca.pem -u␣
↪elastic-s9s:myPassw0rd -H 'Content-Type: application/x-ndjson' --
↪data-binary @abwiki.ndjson -X POST
↪'https://192.168.40.72:9200/_bulk?pretty'
    ```
```
```

#### Using `elasticdump`

Install `elasticdump` first.

For example, you have migrated from another host (in this example, its 192.168.10.100), you can dump your data using `elasticdump` as follows:

```
~/node_modules/elasticdump/bin/elasticdump \
     --cert /etc/elasticsearch/certs/elasticsearch-ca.pem \
     --input=https://elastic-s9s:myPassw0rd@192.168.10.100:9200/abwiki \
     --output=/backups/abwiki.json --type=data \
     --limit=10000
```

Then create an empty index from your target node, in this example, its 192.168.40.72

```
curl --cacert /etc/elasticsearch/certs/elasticsearch-ca.pem \
     -u elastic-s9s:myPassw0rd \
     -H 'Content-Type: application/json' -X PUT
↪'https://192.168.40.72:9200/abwiki_play' \
     -d '{ "settings": { "number_of_replicas": 1 } }'
```

Then it's now ready to dump to the target node,

```
NODE_TLS_REJECT_UNAUTHORIZED=0 ~/node_modules/elasticdump/bin/elasticdump \
  --input=/backups/abwiki.json \
  --output=https://elastic-s9s:myPassw0rd@192.168.40.72:9200/abwiki_play \
  --type=data \
  --limit=10000 \
  --concurrency=4
```

You can also pass the `NODE_TLS_REJECT_UNAUTHORIZED=0` if you are using a self-signed certificate

For high-availability cluster, make sure your indices are replicated so that it shall always have a copy in case the primary goes down.

### 1.77.7 Step 5: Connect to the database

Your application or client will connect directly to any of your nodes (master or data nodes). In this example, we go and connect to node ES1,

- Host: 192.168.40.70
- Port: 9200
- User/Password: The credentials specified during deployment. You can also inspect the file `/etc/cmon.d/cmon_$CID.cnf` where $CID is the cluster id of your Elasticsearch deployment.

No load balancer or additional ports are involved.

### 1.77.8 Step 6: Enable automatic backups

Elasticsearch does not offer the traditional sense of backup. Instead, it uses snapshots which have more capability and robust way to create a consistent backup copy. It is the most recommended method for creating consistent point-in-time backups for an entire cluster or specific indices, and its native to Elasticsearch.

For high-availability cluster, it is very important to always have a reliable backup and a copy of your data in case disaster or catastrophe occurs. In that regard, ClusterControl makes it easy to schedule them automatically.

1. Go to ***ClusterControl GUI → choose the cluster → Backups***.
2. Click ***Create Backup → Schedule a Backup***. The Create a backup schedule wizard will open and proceed to configure your backup as below:

#### (1) Configuration

- **Schedule name**: `Daily ES-HA cluster snapshot`
- **Cluster**: (defaults to your Elasticsearch standalone cluster)
- **Repository**: (defaults to the repository you created during deployment)
- **Backup method**: `elasticsearch-snapshot` (default) [disabled text field]

#### (2) Advanced settings

- **Retention**: On (default)
- **Retention [textfield]**: 4 (Set to your desired number of days to retain backup)

#### (3) Schedule

- **Set backup schedule**: Simple
- **Every**: day at `02:00`
- **Timezone**: select your local timezone

#### (4) Preview

- Verify all settings. You can go back to adjust if needed.

3. Click Create to schedule it.

ClusterControl will now automatically perform your backups. You may also restore your backups from the elasticsearch snapshots in the future if necessary for disaster recovery.

### 1.77.9 Step 7: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill up all necessary information about the SMTP server. You can also opt for Sendmail, however a mail transfer agent (sendmail, postfix or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Select a User group (your group) from the **User group** dropdown.
3. Select your email address to from the **Users in the selected group** dropdown.

4. Click ***Enable***.
5. Set the **Digest delivery time** when you want a digested (summarized events) to be sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest" and you may ignore the **Info** events.

This ensures timely notifications when something goes wrong.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie) or web hooks. See *Integration → Notification Services*.

### 1.77.10 Step 8: Manage your node

ClusterControl provides monitoring for your cluster and system overview which displays workloads based on metrics. Once your Elasticsearch cluster is done deployed, agents using Prometheus exporters are deployed to gather metrics and provide more granular monitoring of your cluster and system workload for your standalone cluster in Elastitcsearch.

Apart from the monitoring, you can manage your node with the most available options for this cluster:

- Database node management: Perform start/stop/restart node, reboot host. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes globally. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Backup management: Create, schedule, restore, store snapshot in an off-cluster storage location such as AWS S3 or any S3 Compatiable storage, while allowing you to set retention period for your backup snapshots. These features are available at **ClusterControl GUI → choose the cluster → Backups → Actions**_ and **ClusterControl GUI → choose the cluster → Backups → More**_.
- Maintenance management: Activate, deactivate, remark and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.77.11 Step 9: Scale your cluster

One of the key benefits of using ClusterControl is the ease of scaling your cluster. Scaling your cluster in ClusterControl can be done by either scaling out or scaling in (or done horizontal scaling). When you remove nodes, this means you want to scale in, and this can be done in ClusterControl by just going to ***ClusterControl GUI → choose the cluster → Nodes → Actions → Remove node ***. While in this section, we are going to focus on adding a node, which means in this case, we are going to scale out.

When scaling out in Elasticsearch using ClusterControl, you are required to choose and define which role you should have this node carries. The following roles available to choose from are the following:

- master-data
- master

- data
- coordinator

Adding a node to your cluster increases the high-availability of your cluster and also allows more sharding or replication for your data. Giving a large number of nodes in the cluster incurs also penalties such as processing time or overhead that could be the source of bottlenecks in the future. So in this case, plan ahead and make sure that the scaling of cluster is properly setup.

Lastly, ClusterControl assumes that the new node that you want to add meets the requirements as decribed under *Prerequisites* and configured with a proper SSH key-based authentication as shown under *Step 1: Set up SSH key-based authentication*.

Let's go over on how you can scale out your cluster in Elasticsearch:

1. To add a new node, go to ***ClusterControl GUI → choose the cluster → Actions → Add new → Replica node → Add node***.
2. In the **Create a database node** wizard, configure the following:

#### (1) Node Configuration

- **Port**: 9200 (default)
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Add node

- **Node**: Specify the IP address or hostname or FQDN of the node that you want to add and press "Enter".
- **Node role**: Displays after the **Node** is specified. Select in this drop-down item to which role you would like this node assigned.

#### (4) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configurations if you wish. The deployment configuration settings will be kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will start provisioning the new database node. You can monitor the progress from **Activity center**, where you will see detailed logs of the deployment. This process can take a few minutes. Once the deployment is complete, the node will appear in the **Nodes** page.

### 1.77.12 Conclusion

Congratulations! You've deployed, monitored, and managed a high-availability Elasticsearch cluster using ClusterControl. With high-availability cluster, you can scale in or scale out depends in your necessity and requirements, as these subjects are dynamically changes depending on the demand. Aside from that, you can manage your cluster by scheduling backups or executing backups based on-demand. You can monitor the nodes, setup maintenance mode when version upgrade is necessary. While utilizing ClusterControl makes it more easier and convenient for you, make sure not to miss such any updates or file a request for features if necessary for your requirements. We also advise that you always practice to do the following:

- Keep your backups snapshots current and up to date.
- Monitor performance and resource usage.
- Secure your node by restricting access and using strong passwords.
- Scale-in or scale-out your cluster.

Enjoy your streamlined high-availability Elasticsearch cluster database deployment, backed by ClusterControl's powerful operations for backups, alerts, scaling, and more!

## 1.78 ClickHouse Standalone

This guide will take you through the process of setting up your first standalone (also known as single-node) ClickHouse cluster using ClusterControl. By following this tutorial, you'll have a fully functional (but non-replicated) database node that you can monitor, manage, back up, and later scale if your requirements change.

ClickHouse is a column-oriented database management system (DBMS) for online analytical processing (OLAP). Applications and clients connect to a ClickHouse server and communicate using SQL over the native (TCP) protocol or the HTTP interface.

A standalone ClickHouse node is the simplest deployment option, but it isn't recommended for production workloads or day-to-day operations in medium- to large-scale organisations. Use it in production only for low-stakes workloads where downtime and potential data loss are acceptable.

**Note**

Note Even when you deploy a single ClickHouse node, ClusterControl configures an **embedded ClickHouse Keeper** (a one-node Raft ensemble) on the same host so that replicated table engines and distributed DDL still work if you later scale out. You do not need to deploy a separate Keeper node for a standalone deployment. ClusterControl also deploys ClickHouse in **TLS-only** mode, so the plaintext native (9000) and HTTP (8123) ports are disabled and clients connect over the secure ports.

**See also**

- ClickHouse: Quick Start
- ClickHouse Keeper

### 1.78.1 Prerequisites

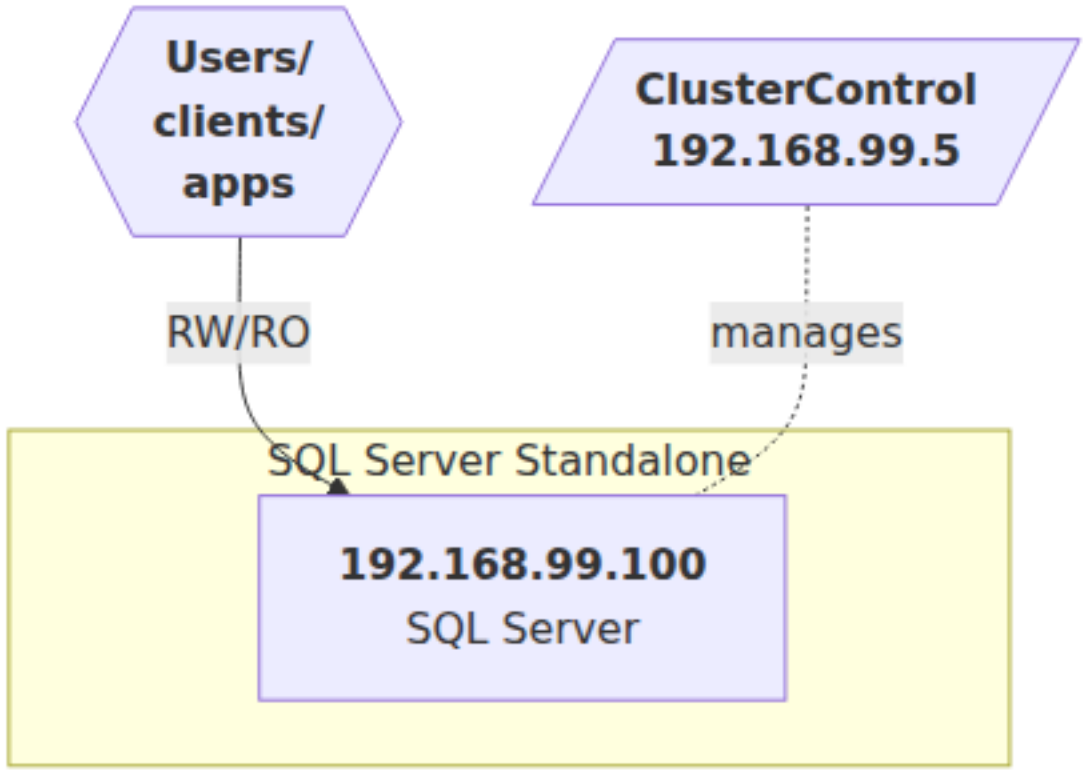
Before proceeding, ensure you have:

- ClusterControl installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least two hosts (bare-metal or virtual machines):
  - One for the ClusterControl server.
  - One for the ClickHouse single node or server. See *Operating System* for supported distributions.
- SSH access to all servers.
- Internet access on the database host to install the required packages from the ClickHouse repositories.

- Network Time Protocol (NTP) configured and running on both hosts to keep their clocks synchronized.

### 1.78.2 Architecture

Below is a simplified diagram of the final architecture for a single-node deployment:




ClickHouse in ClusterControl is deployed without any external supporting load balancers. A single node runs the ClickHouse server together with an embedded ClickHouse Keeper on the same host, so it can function on its own and still be scaled out into a replicated topology later.

### 1.78.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

Copy the public key to your standalone database node (replace 192.168.40.20 with your node's IP/hostname):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.40.20
```

If the target node uses a custom SSH key or port, you can add options:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.40.20
```

For some advanced setups where the user (for example `mymainacc`) is only allowed non-root access with public keys, password challenge is disabled, but the user has sudo privileges, you can copy the root public key as the `root` OS user like this:

```
[root@ccnode ~]# ssh -i /home/mymainacc/.ssh/id_rsa␣
↪mymainacc@192.168.40.20 "sudo bash -c '
umask 077
mkdir -p   /root/.ssh
cat >>     /root/.ssh/authorized_keys
'" < ~/.ssh/id_rsa.pub
```

This copies your root public key to the target node you will be setting up for the single-node ClickHouse deployment.

2. Test passwordless SSH from the ClusterControl server:

```
ssh root@192.168.40.20 "stat \$PWD/"
```

Ensure there is no password prompt. If the command returns the directory status, you're set.

### 1.78.4 Step 2: Deploy a new cluster (single node)

1. Open a web browser and go to the ClusterControl server's IP or hostname.
2. On the ClusterControl dashboard, click ***Deploy a cluster (top-right) → Create a database cluster***. This opens the Deploy cluster wizard.
3. Select **ClickHouse** from the **Database** dropdown. Specify the version you want to deploy by clicking the **Version** dropdown (for example, `24.8` or `25.3`). Click **Continue**.
4. In the **Deploy ClickHouse cluster** wizard, configure the database cluster as below:

#### (1) Cluster details

- **Name**: For example, `ClickHouse-standalone-node`.
- **Tags**: (Optional) e.g., `standalone`, `production`, `dc1`.

#### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa` (ClusterControl will also autofill this field)
- **SSH port**: `22` (default port)
- **SSH sudo password**: (leave blank if you rely on key-based auth)
- **SSH sudo / OS elevation command**: Choose either `sudo` (default), `doas`, or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (3) Node configuration

- **Server port**: `9440` (default). This is the ClickHouse native (secure TCP) port.
- **Keeper port**: `9281` (default). This is the embedded ClickHouse Keeper port.
- **Admin user**: `admin` (default)

- **Admin password**: Password to be assigned to the database `admin` user (minimum 6 characters)
- **Data directory**: `/var/lib/clickhouse` (default)

#### (4) Add nodes

- **ClickHouse node**: Fill in the IP/hostname or FQDN of the node (for example, `192.168.40.20`) and press ++enter++.
- Wait until the node turns green. If a red warning appears, inspect the error and fix it before continuing.
- Leave the node list at a single entry to keep this a standalone deployment.

#### (5) Preview

- Review your configuration. You can go back and adjust any previous section if necessary. The deployment settings are kept until you exit the wizard.

5. Click **Finish** to start deployment.
6. ClusterControl will now install and configure the standalone ClickHouse node (with an embedded Keeper) on 192.168.40.20. You can track progress in the **Activity Center → Jobs**. After a few minutes, your new single-node cluster will appear on the **Home** page.

### 1.78.5 Step 3: Monitor your cluster

Once deployed, you'll see:

- Cluster health: The **Home** page provides the cluster state. Even though it's a single node, it's treated as a "cluster" of size one.
- Node health: Hover over the honeycomb diagram or check the Nodes tab. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: Any triggered alarms will appear if there are configuration or resource issues.
- Automatic recovery status: If enabled, ClusterControl can attempt to restart a crashed ClickHouse server automatically.
- Topology viewer: You'll see a very simple topology with only one node.

### 1.78.6 Step 4: Import data

You can import data into the standalone server in a variety of ways. All examples connect over the secure native port (`9440`) using TLS, since ClusterControl deploys ClickHouse TLS-only.

#### Using `clickhouse-client`

Create a table and insert rows interactively or from a script. Replace the host, user, and password with your own:

```
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "CREATE TABLE default.trips (id UInt64, city String, fare␣
↪Float64) ENGINE = MergeTree ORDER BY id"
```

If you deployed with a self-signed certificate, add `--accept-invalid-certificate` (or point `--config-file` at a client config that trusts the ClusterControl CA).

#### Bulk import from a CSV file

Stream a local CSV file into an existing table using the native client:

```
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "INSERT INTO default.trips FORMAT CSV" < trips.csv
```

For very large files, ClickHouse also reads compressed input directly. For example, pipe a `gzip`-compressed file through `zcat` into the same command.

#### Using the HTTPS interface

You can also insert or query over the HTTP(S) interface on port `8443`. For example, to run a query with `curl`:

```
curl --cacert /etc/clickhouse-server/certs/ca.crt \
    -u admin:myPassw0rd \

↪'https://192.168.40.20:8443/?query=INSERT%20INTO%20default.trips%20FORMAT%20CSV
↪' \
    --data-binary @trips.csv
```

Use `-k` in place of `--cacert ...` if you are testing with a self-signed certificate and accept the security trade-off.

### 1.78.7 Step 5: Connect to the database

Your application or client will connect directly to the standalone server over TLS:

- Host: `192.168.40.20`
- Native (secure TCP) port: `9440`
- HTTPS port: `8443`
- User/Password: The `admin` credentials specified during deployment. You can also inspect the file `/etc/cmon.d/cmon_$CID.cnf`, where `$CID` is the cluster ID of your ClickHouse deployment.

No load balancer or additional ports are involved. A quick connectivity test with the native client:

```
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "SELECT version()"
```

### 1.78.8 Step 6: Enable automatic backups

Even a single-node, standalone ClickHouse cluster needs reliable backups, and ClusterControl makes it easy to schedule them automatically. ClickHouse backups in ClusterControl use the native `BACKUP` SQL command, which writes a consistent copy to the local `backups` disk (typically `/var/lib/clickhouse/backups/`) or to S3-compatible object storage.

1. Go to ***ClusterControl GUI → choose the cluster → Backups***.

2. Click ***Create Backup → Schedule a Backup***. The **Create a backup schedule** wizard will open. Configure your backup as below:

#### (1) Configuration

- **Schedule name**: `Daily ClickHouse standalone backup`
- **Cluster**: (defaults to your ClickHouse standalone cluster)
- **Backup method**: Choose between **Clickhouse native (full)** and **Clickhouse native (incr)** (incremental)
- **Backup node**: (defaults to your single node)

#### (2) Advanced settings

- **Upload backup to cloud**: Off (turn On to store the backup on S3-compatible object storage)
- **Retention**: On (default)
- **Retention [textfield]**: `4` (set to your desired number of days to retain backups)

#### (3) Schedule

- **Set backup schedule**: Simple
- **Every**: day at `02:00`
- **Timezone**: select your local timezone

#### (4) Preview

- Verify all settings. You can go back to adjust if needed.

3. Click **Create** to schedule it.

ClusterControl will now automatically perform your backups. All successful backups are listed on the ***Backups → All Backups*** page, where you can review the backup logs, size, and restore from them in the future if necessary for disaster recovery.

> **Note**
>
> Note Controller storage is not supported for ClickHouse native backups. Backups are stored on the local ClickHouse `backups` disk by default, or on S3-compatible object storage when you enable **Upload backup to cloud**.

### 1.78.9 Step 7: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks, and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill in all the necessary information about the SMTP server. You can also opt for Sendmail; however, a mail transfer agent (sendmail, postfix, or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Select a user group (your group) from the **User group** dropdown.
3. Select your email address from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** for when you want a digested (summarized events) email sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest", and you may ignore the **Info** events.

This ensures timely notifications when something goes wrong.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie), or web hooks. See *Integration → Notification Services*.

### 1.78.10 Step 8: Manage your node

ClusterControl provides monitoring for your cluster and a system overview that displays workloads based on metrics. Once your ClickHouse cluster is deployed, agents using Prometheus exporters are deployed to gather metrics and provide more granular monitoring of your cluster and system workload for your standalone node.

Apart from the monitoring, you can manage your node with the available options for this cluster:

- Database node management: Perform start/stop/restart node, reboot host. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.
- Backup management: Create, schedule, and restore backups, store them in an off-cluster storage location such as AWS S3 or any S3-compatible storage, and set a retention period for your backups. These features are available at ***ClusterControl GUI → choose the cluster → Backups → Actions*** and ***ClusterControl GUI → choose the cluster → Backups → More***.
- Maintenance management: Activate, deactivate, remark, and schedule maintenance mode for all nodes. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from the ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.78.11 Conclusion

You have now deployed, monitored, and managed a standalone ClickHouse server using ClusterControl. A single-node deployment runs the ClickHouse server together with an embedded Keeper, so it works on its own and can be scaled out into a replicated topology later. A single node has no redundancy, but it still gives you what you need to run analytics workloads. As you operate it, keep the following in mind:

- Keep your backups current and up to date.

- Monitor performance and resource usage.
- Secure your node by restricting access and using strong passwords.
- Expand to a multi-node, replicated topology if your needs grow.

From here, ClusterControl handles the day-to-day operations such as backups, alerts, and scaling, so you can focus on your data.

## 1.79 ClickHouse Replicated Cluster

This guide will take you through the process of setting up your first replicated ClickHouse cluster using ClusterControl. By following this tutorial, you'll have a fully functional, multi-node ClickHouse cluster that keeps your data in sync across every node, so the loss of a single node doesn't cost you data or availability.

ClickHouse is a column-oriented database management system (DBMS) for online analytical processing (OLAP). In a replicated cluster, two or more ClickHouse servers hold the same copy of your data. Replication is asynchronous and multi-master, so you can write to any node and the change is propagated to the others. Keeping track of which parts each replica already has and which it still needs to fetch is the job of **ClickHouse Keeper**, a Raft-based coordination service that replaces ZooKeeper.

Compared with a single-node deployment, a replicated cluster gives you data redundancy and lets you take a node down for maintenance without an outage. It is the right starting point whenever downtime or data loss is not acceptable.

**Note**

Note ClusterControl deploys a single-shard, multi-replica topology: every data node holds a full copy of the data. By default, **ClickHouse Keeper is embedded** on each data node (a co-located Raft participant), so you do not need to provide separate hosts to get a working replicated cluster. If you prefer to isolate coordination from your data nodes, you can add dedicated Keeper nodes during deployment instead. For a production Raft quorum, use an odd number of Keepers (3 or 5); a 2-node cluster works but has no fault tolerance for the quorum. ClusterControl also deploys ClickHouse in **TLS-only** mode, so the plaintext native (9000) and HTTP (8123) ports are disabled and clients connect over the secure ports.

**See also**

- ClickHouse: Data Replication
- ClickHouse Keeper

### 1.79.1 Prerequisites

Before proceeding, ensure you have:

- ClusterControl installed and running. If not, follow the instructions in *Quickstart* or use the *Installer Script*.
- At least four hosts (bare-metal or virtual machines):
  - One for the ClusterControl server.

  - Three for the ClickHouse data nodes. A three-node cluster gives you both data redundancy and a healthy Keeper quorum. See *Operating System* for supported distributions.

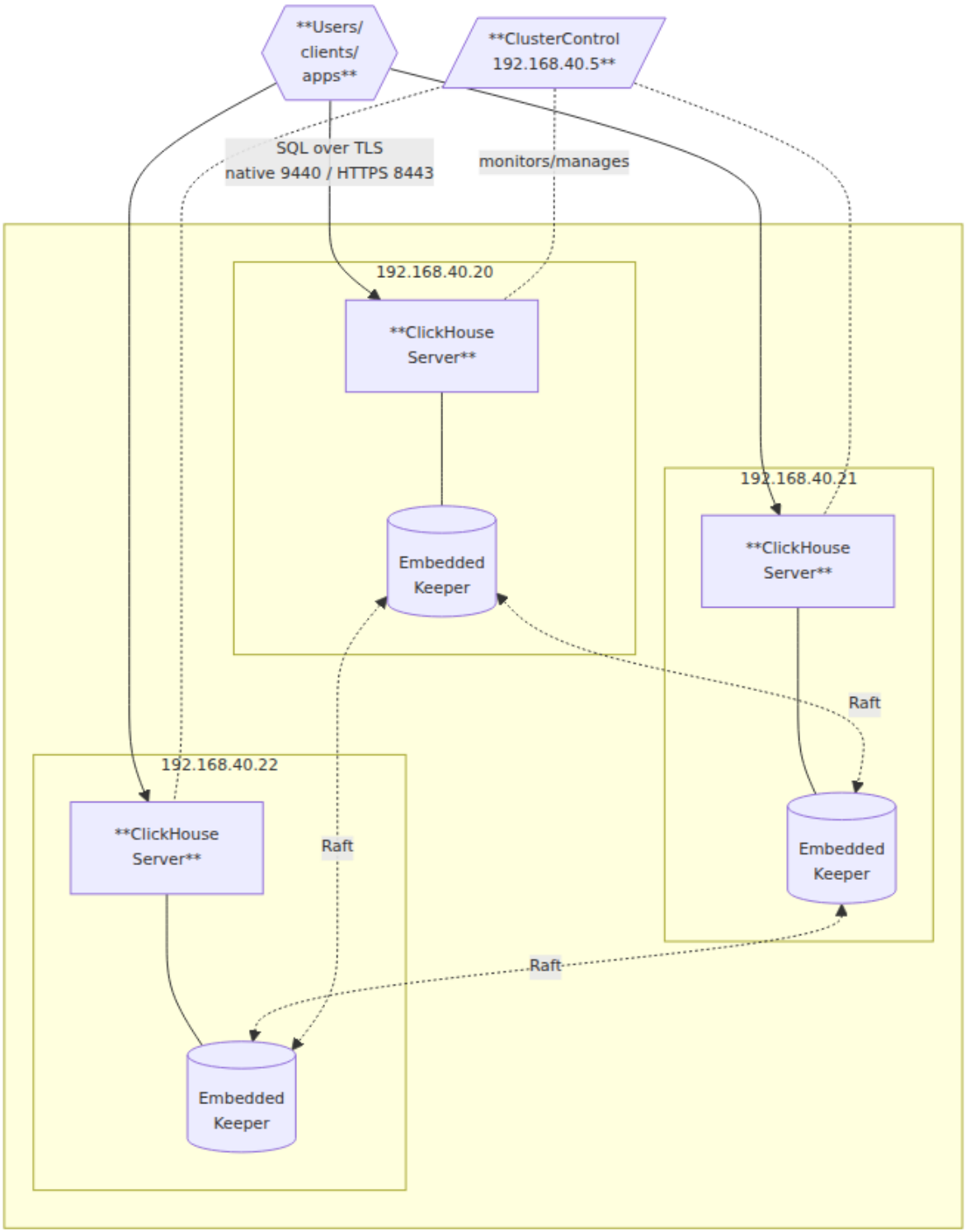
- SSH access to all servers.
- Internet access on the database hosts to install the required packages from the ClickHouse repositories.
- Network Time Protocol (NTP) configured and running on all hosts to keep their clocks synchronized. This matters more in a replicated cluster, where the nodes coordinate with each other.

**Note**

Note ClusterControl only supports managing x86-64 (64-bit) hosts for ClickHouse.

### 1.79.2 Architecture

Below is a simplified diagram of the final architecture for a three-node replicated deployment:




ClusterControl deploys ClickHouse without any external supporting load balancers. All three data nodes belong to one shard with `internal_replication` enabled, so each node stores a complete copy of the data. The embedded Keepers form a Raft ensemble and coordinate replication between the nodes.

On the diagram above, the three ClickHouse nodes will use the following addresses for this example deployment:

- Node 1: 192.168.40.20 (data + embedded Keeper)
- Node 2: 192.168.40.21 (data + embedded Keeper)

- Node 3: 192.168.40.22 (data + embedded Keeper)

### 1.79.3 Step 1: Set up SSH key-based authentication

1. On the ClusterControl server, generate a new SSH key as the root user:

```
ssh-keygen -t rsa
```

Copy the public key to your first ClickHouse node (replace 192.168.40.20 with your node's IP/hostname):

```
ssh-copy-id -i /root/.ssh/id_rsa root@192.168.40.20
```

If the target node uses a custom SSH key or port, you can add options:

```
ssh-copy-id -i /root/.ssh/id_rsa -p 22 -o 'IdentityFile /
↪root/myprivatekey.pem' root@192.168.40.20
```

For some advanced setups where the user (for example `mymainacc`) is only allowed non-root access with public keys, password challenge is disabled, but the user has sudo privileges, you can copy the root public key as the `root` OS user like this:

```
[root@ccnode ~]# ssh -i /home/mymainacc/.ssh/id_rsa␣
↪mymainacc@192.168.40.20 "sudo bash -c '
umask 077
mkdir -p   /root/.ssh
cat >>     /root/.ssh/authorized_keys
'" < ~/.ssh/id_rsa.pub
```

This copies your root public key to the target node you will be setting up for the replicated ClickHouse deployment.

2. Test passwordless SSH from the ClusterControl server:

```
ssh root@192.168.40.20 "stat \$PWD/"
```

Ensure there is no password prompt. If the command returns the directory status, you're set.

3. Repeat steps 1 and 2 for the remaining nodes, 192.168.40.21 and 192.168.40.22. ClusterControl needs passwordless SSH to every node it manages.

### 1.79.4 Step 2: Deploy a new cluster (replicated)

1. Open a web browser and go to the ClusterControl server's IP or hostname.
2. On the ClusterControl dashboard, click ***Deploy a cluster (top-right) → Create a database cluster***. This opens the Deploy cluster wizard.
3. Select **ClickHouse** from the **Database** dropdown. Specify the version you want to deploy by clicking the **Version** dropdown (for example, `24.8` or `25.3`). Click **Continue**.
4. In the **Deploy ClickHouse cluster** wizard, configure the database cluster as below:

### (1) Cluster details

- **Name**: For example, `ClickHouse-replicated`. This name also becomes the cluster identifier used by `ON CLUSTER` statements later on, so pick something without spaces if you can.
- **Tags**: (Optional) e.g., `replicated`, `production`, `dc1`.

### (2) SSH configuration

- **SSH user**: `root`
- **SSH user key path**: `/root/.ssh/id_rsa` (ClusterControl will also autofill this field)
- **SSH port**: `22` (default port)
- **SSH sudo password**: (leave blank if you rely on key-based auth)
- **SSH sudo / OS elevation command**: Choose either `sudo` (default), `doas`, or `pbrun`
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

### (3) Node configuration

- **Server port**: `9440` (default). This is the ClickHouse native (secure TCP) port.
- **Keeper port**: `9281` (default). This is the ClickHouse Keeper client port.
- **Admin user**: `admin` (default)
- **Admin password**: Password to be assigned to the database `admin` user (minimum 6 characters). The admin user is created on all nodes via `CREATE USER ... ON CLUSTER`.
- **Data directory**: `/var/lib/clickhouse` (default)

### (4) Add nodes

- **ClickHouse node**: Fill in the IP/hostname or FQDN of each data node — `192.168.40.20`, `192.168.40.21`, and `192.168.40.22` — pressing ++enter++ after each one. Specifying two or more data nodes tells ClusterControl to deploy a replicated cluster with embedded Keeper on each node.
- **Keeper node**: (Optional) Leave this empty to use embedded Keeper on the data nodes. Only fill it in if you want dedicated Keeper hosts separate from your data nodes; if so, add an odd number (3 or 5) for a proper quorum.
- Wait until every node turns green. If a red warning appears, inspect the error and fix it before continuing. You can only proceed once all nodes are reachable.

### (5) Preview

- Review your configuration. You can go back and adjust any previous section if necessary. The deployment settings are kept until you exit the wizard.

5. Click **Finish** to start deployment.
6. ClusterControl will now install and configure the three ClickHouse nodes, wire up the embedded Keepers into a Raft ensemble, and configure replication between the nodes. You can track progress

in the **Activity Center → Jobs**. After a few minutes, your new replicated cluster will appear on the **Home** page.

### 1.79.5 Step 3: Monitor your cluster

Once deployed, you'll see:

- Cluster health: The **Home** page provides the overall cluster state across all three nodes.
- Node health: Hover over the honeycomb diagram or check the Nodes tab. You can also see more detailed histograms under ***ClusterControl GUI → Clusters → choose the cluster → Dashboards***.
- Recent alarms: Any triggered alarms will appear if there are configuration or resource issues.
- Automatic recovery status: If enabled, ClusterControl can attempt to restart a crashed ClickHouse server automatically.
- Topology viewer: You'll see all three replicas and their Keeper participants.

To confirm the replicated topology from the database side, connect to any node and inspect the `system.clusters` table. It lists every replica in your cluster along with the cluster identifier you set during deployment:

```
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "SELECT cluster, shard_num, replica_num, host_name FROM␣
↪system.clusters"
```

### 1.79.6 Step 4: Import data

In a replicated cluster you don't load data into each node separately. Instead, you create a table using the `ReplicatedMergeTree` engine (typically with `ON CLUSTER` so the DDL runs on every node at once), then insert into any single node — ClickHouse and Keeper take care of copying the data to the other replicas. All examples connect over the secure native port (`9440`) using TLS, since ClusterControl deploys ClickHouse TLS-only.

#### Create a replicated table

Create the table on all nodes in one statement. The `{shard}` and `{replica}` placeholders are macros that ClusterControl configures per node, so the same command works everywhere. Replace `ClickHouse-replicated` with the cluster name you used during deployment:

```
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "CREATE TABLE default.trips ON CLUSTER 'ClickHouse-replicated'
             (id UInt64, city String, fare Float64)
             ENGINE = ReplicatedMergeTree('/
↪clickhouse/tables/{shard}/{database}/{table}', '{replica}')
             ORDER BY id"
```

If you deployed with a self-signed certificate, add `--accept-invalid-certificate` (or point `--config-file` at a client config that trusts the ClusterControl CA).

### Insert and verify replication

Insert a few rows into one node, then read them back from a different node to confirm replication is working:

```
# Write to node 1
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "INSERT INTO default.trips VALUES (1,'Berlin',12.5),(2,'Oslo',
↪9.0)"

# Read from node 2
clickhouse-client --host 192.168.40.21 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "SELECT count() FROM default.trips"
```

The count on node 2 should match what you inserted on node 1.

### Bulk import from a CSV file

Stream a local CSV file into the replicated table using the native client. You only need to load it once, into any node:

```
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "INSERT INTO default.trips FORMAT CSV" < trips.csv
```

For very large files, ClickHouse also reads compressed input directly. For example, pipe a `gzip`-compressed file through `zcat` into the same command.

**Tip**

Because every node holds a full copy of the data, this cluster does not need a `Distributed` table for reads — any node can answer a query on its own.

## 1.79.7 Step 5: Connect to the database

Your application or client can connect to any node in the cluster over TLS, since each one holds a complete copy of the data. In this example we connect to node 1:

- Host: `192.168.40.20` (or any other node, `192.168.40.21` / `192.168.40.22`)
- Native (secure TCP) port: `9440`
- HTTPS port: `8443`
- User/Password: The `admin` credentials specified during deployment. You can also inspect the file `/etc/cmon.d/cmon_$CID.cnf`, where `$CID` is the cluster ID of your ClickHouse deployment.

For high availability, point your application at more than one node (most ClickHouse drivers accept a list of hosts and will fail over automatically), or place a load balancer in front of the cluster. A quick connectivity test with the native client:

```
clickhouse-client --host 192.168.40.20 --port 9440 --secure \
    --user admin --password 'myPassw0rd' \
    --query "SELECT version()"
```

### 1.79.8 Step 6: Enable automatic backups

A replicated cluster protects you against losing a node, but it is not a substitute for backups — a bad `DROP`, a schema mistake, or corruption replicates just as fast as good data. ClusterControl makes it easy to schedule backups automatically. ClickHouse backups in ClusterControl use the native `BACKUP` SQL command, which writes a consistent copy to the local `backups` disk (typically `/var/lib/clickhouse/backups/` ) or to S3-compatible object storage.

1. Go to ***ClusterControl GUI → choose the cluster → Backups***.
2. Click ***Create Backup → Schedule a Backup***. The **Create a backup schedule** wizard will open. Configure your backup as below:

#### (1) Configuration

- **Schedule name**: `Daily ClickHouse replicated backup`
- **Cluster**: (defaults to your ClickHouse replicated cluster)
- **Backup method**: Choose between **Clickhouse native (full)** and **Clickhouse native (incr)** (incremental)
- **Backup node**: Select which node the backup should run on. Since each replica holds a full copy, backing up one node captures all your data.

#### (2) Advanced settings

- **Upload backup to cloud**: Off (turn On to store the backup on S3-compatible object storage)
- **Retention**: On (default)
- **Retention [textfield]**: `4` (set to your desired number of days to retain backups)

#### (3) Schedule

- **Set backup schedule**: Simple
- **Every**: day at `02:00`
- **Timezone**: select your local timezone

#### (4) Preview

- Verify all settings. You can go back to adjust if needed.

3. Click **Create** to schedule it.

ClusterControl will now automatically perform your backups. All successful backups are listed on the ***Backups → All Backups*** page, where you can review the backup logs, size, and restore from them in the future if necessary for disaster recovery.

**Note**

Note ClickHouse native backups are written to the local filesystem of the node that runs them, so they are node-local. Keep your full and incremental backups on the **same node**: an incremental backup can only build on a parent that lives on the same node it runs on. Controller storage is not supported for ClickHouse native backups; use the local `backups` disk (default) or enable **Upload backup to cloud** for S3-compatible object storage.

### 1.79.9 Step 7: Configure alerts

To keep track of any issues or incidents in your cluster, it's important to set up alerting. ClusterControl supports sending alarms and alerts to email, web hooks, and third-party notification services like Slack or Telegram. In this example, we are going to use email.

Firstly, configure the mail server. Go to ***ClusterControl GUI → Settings → You don't have a mail server configured. Configure now → SMTP Server***. Fill in all the necessary information about the SMTP server. You can also opt for Sendmail; however, a mail transfer agent (sendmail, postfix, or exim) must be installed on the ClusterControl server.

Once configured, we can configure the alert and recipients as below:

1. Go to ***ClusterControl GUI → choose the cluster → Settings → Email Notifications***.
2. Select a user group (your group) from the **User group** dropdown.
3. Select your email address from the **Users in the selected group** dropdown.
4. Click **Enable**.
5. Set the **Digest delivery time** for when you want a digested (summarized events) email sent to you every day.
6. Set all **Critical events** to "Deliver" (default), all **Warning events** to "Digest", and you may ignore the **Info** events.

This ensures timely notifications when something goes wrong.

**Tip**

You can also configure alarms to be sent to third-party notification systems (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie), or web hooks. See *Integration → Notification Services*.

### 1.79.10 Step 8: Manage your nodes

ClusterControl provides monitoring for your cluster and a system overview that displays workloads based on metrics. Once your ClickHouse cluster is deployed, agents using Prometheus exporters are deployed to gather metrics and provide more granular monitoring of your cluster and system workload across all nodes.

Apart from the monitoring, you can manage your nodes with the available options for this cluster:

- Database node management: Perform start/stop/restart node, reboot host. These features are available at ***ClusterControl GUI → choose the cluster → Nodes → Actions***.
- Configuration management: Perform database configuration changes. This feature is available at ***ClusterControl GUI → choose the cluster → Manage → Configuration***.

- Backup management: Create, schedule, and restore backups, store them in an off-cluster storage location such as AWS S3 or any S3-compatible storage, and set a retention period for your backups. These features are available at ***ClusterControl GUI → choose the cluster → Backups → Actions*** and ***ClusterControl GUI → choose the cluster → Backups → More***.
- Maintenance management: Activate, deactivate, remark, and schedule maintenance mode for all nodes. Because your data is replicated, you can put one node into maintenance for patching or upgrades while the others keep serving traffic. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → Schedule maintenance***.
- SSH console: Access your nodes directly from the ClusterControl GUI via web SSH console. This feature is available at ***ClusterControl GUI → choose the cluster → Nodes → Actions → SSH Console***.

### 1.79.11 Step 9: Scale your cluster

Scaling out a replicated ClickHouse cluster means adding another replica. The new ClickHouse node joins the existing shard, pulls a full copy of the data from its peers, and starts serving queries, which improves both your redundancy and your read capacity.

ClusterControl assumes that the new node meets the requirements described under *Prerequisites* and is configured with proper SSH key-based authentication as shown in *Step 1: Set up SSH key-based authentication*.

To add a replica:

1. Go to ***ClusterControl GUI → choose the cluster → Actions… → Add new → Data node***.
2. In the **Create a data node** wizard, configure the following:

#### (1) Node configuration

- **Server port**: `9440` (default)
- **Install software**: On (default)
- **Disable firewall**: Checked (default)
- **Disable SELinux/AppArmor**: Checked (default)

#### (2) Add node

- **Node**: Specify the IP address, hostname, or FQDN of the node that you want to add (for example, `192.168.40.23`) and press ++enter++.

#### (3) Preview

- Review the summary of the deployment. You can always go back to any previous section to modify your configuration. The deployment settings are kept until you exit the wizard.

3. Click **Finish** to trigger the deployment job.
4. ClusterControl will provision the new node, add it as a replica to the existing shard, and regenerate the cluster topology so the new node is included in `remote_servers`. You can monitor the progress from **Activity Center → Jobs**. Once complete, the node appears on the **Nodes** page and begins syncing data from the other replicas.

**Tip**

Adding another replica also adds a Keeper participant when embedded Keeper is used. Keep the Keeper ensemble at an odd size (3 or 5) for the healthiest quorum.

### 1.79.12 Conclusion

You have now deployed, monitored, and managed a replicated ClickHouse cluster using ClusterControl. Unlike a single node, a replicated cluster keeps a full copy of your data on every node and coordinates them with ClickHouse Keeper, so you get data redundancy and can take a node offline without an outage. As you operate it, keep the following in mind:

- Keep your backups current, and remember that ClickHouse native backups are node-local.
- Monitor performance and resource usage across all nodes, not just one.
- Secure your nodes by restricting access and using strong passwords.
- Scale out by adding replicas as your read traffic grows, keeping the Keeper quorum at an odd number.

From here, ClusterControl handles the day-to-day operations such as backups, alerts, maintenance, and scaling, so you can focus on your data.

## 1.80 ClusterControl and Database Clusters Deployment in an Offline Environment

This tutorial outlines how to perform firstly the offline installation of Severalnines ClusterControl and secondly deploying database clusters on hosts (virtual machines or bare-metal servers) that do not have an internet connection.

The document is divided into two parts where part one contains instructions on setting up a software repository host that will serve up a database, third-party tools, and OS software packages to database hosts. Part two contains instructions on how to set up ClusterControl on a host that does not have an internet connection. All hosts in this article are running on CentOS/RockyLinux/AlmaLinux 8 & 9, which uses the YUM/DNF package manager.

Part one and part two can be performed independently of each other. However, the repo server must be ready prior to deploying database clusters from ClusterControl.

We will use the following naming convention for the hosts:

- Satellite repository server (connected to the Internet, reachable by ClusterControl and database servers via local network)
- ClusterControl server (offline)
- Database servers (offline)

### 1.80.1 Setting up Satellite Repository Server

The satellite server is considered the offline repository server to be used by ClusterControl when deploying database servers and the corresponding tools. This server should have an internet connection since we need to pull the necessary repositories into it. Let's call this "repo server". This server will store and host thousands of packages so do allocate sufficient disk space to the repository directory (`/var/www/html/repos`) in advance.

### Setting up OS, OS tools and third-party utility repositories

1. Set up a web server on the repo server to serve up packages to the yum package manager remotely:

```
$ dnf install -y nginx
$ systemctl start nginx
$ systemctl enable nginx
$ systemctl status nginx
```

2. Set up software required for a remote repository server:

```
$ yum install -y createrepo yum-utils
$ yum -y install epel-release
```

3. Configure the nginx in the Satellite Server:

```
$ sudo vi /etc/nginx/conf.d/repos.conf
server {
        listen   80;
        server_name  192.168.100.100;
        root   /var/www/html/repos;
    index index.html;
    location / {
                autoindex on;
        }
}
```

4. Restart the nginx service:

```
$ systemctl restart nginx
$ chcon -Rt httpd_sys_content_t /var/www/html/repos/
```

5. Set up repository directory to be served by the web server:

```
$ mkdir -p /var/www/html/repos/rhel
```

6. Perform repository syncing with the appropriate repositories (the following commands will consume time to finish):

```
$ dnf reposync -g --delete -p /var/www/html/repos/rhel/ --
↪repoid=baseos --newest-only --download-metadata
$ dnf reposync -g --delete -p /var/www/html/repos/rhel/ --
↪repoid=appstream--newest-only --download-metadata
$ dnf reposync -g --delete -p /var/www/html/repos/rhel/ --
↪repoid=extras --newest-only --download-metadata
-- For AlmaLinux/RockyLinux/CentOS 9 (CodeReadyBuilder)
$ dnf config-manager set-enabled crb
$ dnf reposync -g --delete -p /var/www/html/repos/rhel/ --
↪repoid=crb--newest-only --download-metadata
-- For AlmaLinux/RockyLinux/CentOS 8 (Powertools)
$ dnf config-manager set-enabled powertools
$ dnf reposync -g --delete -p /var/www/html/repos/rhel/ --
↪repoid=powertools--newest-only --download-metadata
```

### Adding Severalnines repository to the repo server

1. Import the public key of Severalnines

```
$ rpm --import http://repo.severalnines.com/severalnines-repos.asc
```

2. Download the s9s-repo fom the Severalnines Repository

```
$ wget http://www.severalnines.com/downloads/cmon/s9s-repo.repo -P /
↪etc/yum.repos.d/
```

3. Edit the `s9s-repo.repo` and add the following content:

```
[s9s-tools]
name=s9s-tools (RHEL_8)
type=rpm-md
baseurl=http://repo.severalnines.com/s9s-tools/{os_codename}
gpgcheck=1
gpgkey=http://repo.severalnines.com/s9s-tools/{os_codename}/repodata/repomd.xml.key
enabled=1
```

> **Note**
>
> Note Replace `{os_codename}` based on the list of Operating System code below: `RHEL_7`,`RHEL_8`,`RHEL_9`,`CentOS_7`,`CentOS_8`,`CentOS_9`.

4. Create directory and perform repository sync:

```
$ mkdir -p /var/www/html/repos/s9s
$ dnf reposync -g --delete -p /var/www/html/repos/s9s/ --
↪repoid=s9s-repo --newest-only --download-metadata
$ dnf reposync -g --delete -p /var/www/html/repos/s9s/ --
↪repoid=s9s-tools --newest-only --download-metadata
```

### Adding PostgreSQL repository to the repo server

1. Download the PostgreSQL release package from this page:

```
$ yum install https://download.postgresql.org/pub/repos/yum/reporpms/
↪{os_code}-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

> **Note**
>
> Note Replace the `{os_code}` for each verion of Operating System; `EL-7`, `EL-8`, `EL-9`.

2. Clear the repository cache and list out the repositories, make sure the newly added PostgreSQL repository is in the list:

```
$ yum clean all
$ yum repolist
```

3. Set up the repository directory to be served by the web server for PostgreSQL:

```
$ mkdir -p /var/www/html/repos/postgresql
```

4. Perform repository syncing:

```
$ dnf reposync --delete -p /var/www/html/repos/postgresql --
↪repoid=pgdg-common --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/postgresql --
↪repoid=pgdg16 --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/postgresql --
↪repoid=pgdg15 --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/postgresql --
↪repoid=pgdg14 --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/postgresql --
↪repoid=pgdg13 --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/postgresql --
↪repoid=pgdg12 --newest-only --download-metadata
```

### Adding MongoDB repository to the repo server

The package name of the MongoDB server as seen in the yum command is `yum install mongodb-org`. Therefore, the `/etc/yum.repos.d/mongodb-org.repo` can only have one entry pointing to the appropriate version of MongoDB (one and only one of either 4.4, 5.0 or 6.0).

1. Create a repo file located at `/etc/yum.repos.d/mongodb-org.repo` and make sure it has the following lines (the example shows we activate MongoDB 6.0):

```
# cat mongodb-org.repo
[mongodb-org-6.0-latest]
name=MongoDB lastest 6.0 Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/
↪mongodb-org/6.0/x86_64/
gpgcheck=0
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-6.0.asc

#[mongodb-org-5.0]
#name=MongoDB Repository

↪#baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/5.0/x86_64/
#gpgcheck=1
#enabled=1
#gpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc
```

**Note**

Note Disable GPGkey checking if you don't have an internet connection. Furthermore, only one version (i.e., either 6.0, 5.0, 4.4) can be supported at any given time. The reason is, that all the versions are called "mongodb-org".

2. Clear the repository cache and list out the repositories, make sure the newly added MongoDB repository is in the list:

```
$ yum clean all
$ yum repolist
```

3. Set up the repository directory to be served by the web server for MongoDB:

```
$ mkdir -p /var/www/html/repos/mongodb
```

4. Perform repository syncing:

```
$ dnf reposync --delete -p /var/www/html/repos/mongodb --
↪repoid=mongodb-org-6.0-latest --newest-only --download-metadata
```

### Adding Percona MongoDB repository to the repo server

The package name of the Percona MongoDB server as seen in the yum command is `yum install percona-server-mongodb`. Therefore, the `/etc/yum.repos.d/mongodb-org.repo` can only have one entry pointing to the appropriate version of MongoDB (one and only one of either 4.4, 5.0 or 6.0).

1. Create a repo file located at `/etc/yum.repos.d/percona-mongodb.repo` and make sure it has the following lines (the example shows we activate Percona MongoDB 6.0):

```
[psmdb-60-release-x86_64]
name = Percona Server for MongoDB 6.0 release/x86_64 YUM repository
baseurl =␣
↪http://repo.percona.com/psmdb-60/yum/release/$releasever/RPMS/x86_64
enabled = 1
gpgcheck = 0
#gpgkey = file:///etc/pki/rpm-gpg/PERCONA-PACKAGING-KEY

[psmdb-60-release-noarch]
name = Percona Server for MongoDB 6.0 release/noarch YUM repository
baseurl =␣
↪http://repo.percona.com/psmdb-50/yum/release/$releasever/RPMS/noarch
enabled = 1
gpgcheck = 0
#gpgkey = file:///etc/pki/rpm-gpg/PERCONA-PACKAGING-KEY

[psmdb-60-release-sources]
name = Percona Server for MongoDB 6.0 release/sources YUM repository
baseurl =␣
↪http://repo.percona.com/psmdb-60/yum/release/$releasever/SRPMS
enabled = 0
gpgcheck = 0
#gpgkey = file:///etc/pki/rpm-gpg/PERCONA-PACKAGING-KEY

[tools-release-x86_64]
name = Percona Tools release/x86_64 YUM repository
baseurl =␣
↪http://repo.percona.com/tools/yum/release/$releasever/RPMS/x86_64
```

(continues on next page)

(continued from previous page)

```
enabled = 1
gpgcheck = 0
#gpgkey = file:///etc/pki/rpm-gpg/PERCONA-PACKAGING-KEY

[tools-release-noarch]
name = Percona Tools release/noarch YUM repository
baseurl =␣
↪http://repo.percona.com/tools/yum/release/$releasever/RPMS/noarch
enabled = 1
gpgcheck = 0
#gpgkey = file:///etc/pki/rpm-gpg/PERCONA-PACKAGING-KEY

[tools-release-sources]
name = Percona Tools release/sources YUM repository
baseurl = http://repo.percona.com/tools/yum/release/$releasever/SRPMS
enabled = 0
gpgcheck = 0
#gpgkey = file:///etc/pki/rpm-gpg/PERCONA-PACKAGING-KEY
```

**Note**

Note Disable GPGkey checking if you don't have an internet connection. Furthermore, only one version (i.e., either 6.0, 5.0, 4.4) can be supported at any given time. The reason is, that all the versions are called "percona-server-mongodb".

2. Clear the repository cache and list out the repositories, make sure the newly added MongoDB repository is in the list:

```
$ yum clean all
$ yum repolist
```

3. Set up the repository directory to be served by the web server for Percona:

```
$ mkdir -p /var/www/html/repos/percona
```

4. Perform repository syncing:

```
$ dnf reposync --delete -p /var/www/html/repos/percona --
↪repoid=psmdb-60-release-x86_64 --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/percona --
↪repoid=tools-release-x86_64 --newest-only --download-metadata
```

### Adding Oracle MySQL repository to the repo server

1. Download the appropriate Oracle MySQL community release package from this page:

```
$ yum install https://dev.mysql.com/get/mysql84-community-release-
↪{os_code}-1.noarch.rpm
```

**Note**

Note Replace the `{os_code}` for each verion of Operating System; `el7`, `el8`, `el9`.

2. Clear the repository cache and list out the repositories, make sure the newly added MySQL 8 repository is in the list:

```
$ yum clean all
$ yum repolist
```

3. Set up the repository directory to be served by the web server for Oracle:

```
$ mkdir -p /var/www/html/repos/oracle
```

4. Perform repository syncing:

```
$ dnf reposync --delete -p /var/www/html/repos/oracle --
↪repoid=mysql80-community --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/oracle --
↪repoid=mysql-connectors-community --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/oracle --
↪repoid=mysql-tools-community --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/oracle --
↪repoid=mysql-cluster-8.0-community --newest-only --download-metadata
```

### Adding Percona MySQL repository to the repo server

1. Download the appropriate Percona MySQL release package from this page:

```
$ yum install␣
↪https://repo.percona.com/yum/percona-release-latest.noarch.rpm
```

2. Enable the Percona Server for MSQL 8.0 (the latest version at the time of this writing):

```
$ percona-release setup ps80
```

3. Clear the repository cache and list out the repositories, make sure the newly added Percona Server repository is in the list:

```
$ yum clean all
$ yum repolist
```

4. Set up the repository directory to be served by the web server for Percona (if not exists):

```
$ mkdir -p /var/www/html/repos/percona
```

5. Perform repository syncing:

```
$ dnf reposync --delete -p /var/www/html/repos/percona --
↪repoid=ps-80-release-x86_64 --newest-only --download-metadata
```

### Adding MariaDB repository to the repo server

1. Install the MariaDB repository by using the following script, taken from this page:

```
$ curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo␣
↪bash
```

2. Clear the repository cache and list out the repositories, make sure the newly added MariaDB repository is in the list:

```
$ yum clean all
$ yum repolist
```

3. Set up the repository directory to be served by the web server for MariaDB:

```
$ mkdir -p /var/www/html/repos/mariadb
```

4. Perform repository syncing:

```
$ dnf reposync --delete -p /var/www/html/repos/mariadb --
↪repoid=mariadb-main --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/mariadb --
↪repoid=mariadb-maxscale --newest-only --download-metadata
$ dnf reposync --delete -p /var/www/html/repos/mariadb --
↪repoid=mariadb-tools --newest-only --download-metadata
```

### Creating Repository

Now we have downloaded all the necessary packages, it is time to create the repository on the repo server:

```
$ createrepo /var/www/html/repos/rhel
$ createrepo /var/www/html/repos/s9s
$ createrepo /var/www/html/repos/postgresql
$ createrepo /var/www/html/repos/mongodb
$ createrepo /var/www/html/repos/percona
$ createrepo /var/www/html/repos/oracle
$ createrepo /var/www/html/repos/mariadb
```

Just to give some heads up on the disk usage, here is how much you need for the repository directory:

```
$ du -sh /var/www/html/repos
38G   /var/www/html/repos/
```

At this point, the repo server should be ready with all the necessary packages for OS, OS tools, third-party dependencies (Perl, Python, wget, tar, socat, net-tools, etc) and databases such as MySQL, MariaDB, PostgreSQL, MongoDB (from MongoDB Org and Percona). We should see the following directory listing under `/var/www/html/repos`:

```
$ tree -d -L 2 /var/www/html/repos
.
├── mariadb
│   ├── mariadb-main
```

(continues on next page)

(continued from previous page)

```
|   ├── mariadb-maxscale
|   ├── mariadb-tools
|   └── repodata
├── mongodb
|   ├── mongodb-org-6.0-latest
|   └── repodata
├── oracle
|   ├── mysql57-community
|   ├── mysql80-community
|   ├── mysql-cluster-7.6-community
|   ├── mysql-cluster-8.0-community
|   ├── mysql-connectors-community
|   ├── mysql-tools-community
|   └── repodata
├── percona
|   ├── ps-80-release-x86_64
|   ├── psmdb-60-release-x86_64
|   ├── repodata
|   └── tools-release-x86_64
├── postgresql
|   ├── pgdg13
|   ├── pgdg14
|   ├── pgdg-common
|   └── repodata
└── rhel
    ├── base
    ├── epel
    ├── repodata
    └── updates
```

### 1.80.2 Setting Up Database Nodes

Next, we need to set up the following things on all database nodes:

- Configure passwordless SSH from the ClusterControl node to all database nodes.
- Configure the package repository to be pointing to the satellite server.

Suppose there are 3 database nodes to deploy a PostgreSQL Streaming Replication, with the following host details:

- 192.168.100.100 – Repo server
- 192.168.100.200 – ClusterControl
- 192.168.100.211 – PostgreSQL db1
- 192.168.100.212 – PostgreSQL db2
- 192.168.100.213 – PostgreSQL db3

### Configure Passwordless SSH

Make sure the appropriate user (root or sudo user) can SSH from the ClusterControl node to the database nodes.

1. On the ClusterControl server, generate an SSH key to be used for passwordless SSH:

```
$ whoami
root
$ ssh-keygen -t rsa # press Enter on all prompts
```

2. Copy this SSH key to all the database nodes:

```
$ ssh-copy-id root@192.168.100.211 # db1
$ ssh-copy-id root@192.168.100.212 # db2
$ ssh-copy-id root@192.168.100.213 # db3
```

   If the database server does not support SSH password authentication, you need to copy the SSH public key located at `/root/.ssh/id_rsa.pub` on the ClusterControl server into `/root/.ssh/authorized_keys` on every database node. See *SSH Key-based Authentication* for more details.

3. Verify that you should be able to execute the following command without error:

```
$ ssh root@192.168.100.211 "ls -l /sbin"
```

We have now configured the passwordless SSH.

### Configure OS Repository

Perform the following commands on all nodes (ClusterControl and all database nodes):

1. Create a backup directory to existing the repository definition files:

```
$ mkdir ~/repos.d.backup
```

2. Move existing repository definition files to the backup directory:

```
$ mv /etc/yum.repos.d/*.repo ~/repos.d.backup
```

3. Create a repository definition file pointing to the repo server created in Setting up Satellite Repository Server.

```
$ vi /etc/yum.repos.d/satellite.repo
```

   And add the following lines:

```
[rhel]
name=RHEL Local Repo Server
baseurl=http://192.168.100.100/repos/rhel
enabled=1
gpgcheck=0
```

4. Refresh the repository list:

```
$ sudo yum clean all
$ sudo yum repolist # make sure only the above repository is listed
```

### Configure Severalnines Repository

Perform the following commands on ClusterControl nodes:

1. Create a repository definition file pointing to the repo server created in Setting up Satellite Repository Server.

```
$ vi /etc/yum.repos.d/s9s.repo
```

And add the following lines:

```
```ini
[s9s-repo]
name = Severalnines Release Repository
baseurl=http://192.168.100.100/s9s-repo/
enabled=1
gpgcheck=0

[s9s-tools]
name = Severalnines Tools Repository
baseurl=http://192.168.100.100/s9s-tools/
enabled=1
gpgcheck=0
```
```

2. Refresh the repository list:

```
$ sudo yum clean all
$ sudo yum repolist   # make sure only the above repository is listed
```

### Configure Database Repository

Perform the following commands on all database nodes, based on the database type that each node wants to deploy.

1. Create a backup directory to existing the repository definition files:

```
$ mkdir ~/repos.d.backup
```

2. Move existing repository definition files to the backup directory:

```
$ mv /etc/yum.repos.d/*.repo ~/repos.d.backup
```

3. Create a repository definition file pointing to the repo server created in *Setting up Satellite Repository Server*.

   **For MariaDB (MariaDB Server, MariaDB Galera):**

```
$ vi /etc/yum.repos.d/mariadb.repo
```

And add the following lines:

```
[mariadb]
name=MariaDB Local Repo Server
baseurl=http://192.168.100.100/repos/mariadb
enabled=1
gpgcheck=0
```

**For MongoDB Inc (MongoDB):**

```
$ vi /etc/yum.repos.d/mongodb.repo
```

And add the following lines:

```
[mongodb]
name=MongoDB Local Repo Server
baseurl=http://192.168.100.100/repos/mongodb
enabled=1
gpgcheck=0
```

**For Percona (Percona Server, Percona XtraDB Cluster, Percona Server for MongoDB):**

```
$ vi /etc/yum.repos.d/percona.repo
```

And add the following lines:

```
[percona]
name=Percona Local Repo Server
baseurl=http://192.168.100.100/repos/percona
enabled=1
gpgcheck=0
```

**For PostgreSQL:**

```
$ vi /etc/yum.repos.d/postgresql.repo
```

And add the following lines:

```
[postgresql]
name=PostgreSQL Local Repo Server
baseurl=http://192.168.100.100/repos/postgresql
enabled=1
gpgcheck=0
```

**For Oracle (MySQL Replication, MySQL Cluster):**

```
$ vi /etc/yum.repos.d/oracle.repo
```

And add the following lines:

```
[oracle]
name=Oracle Local Repo Server
baseurl=http://192.168.100.100/repos/oracle
```

(continues on next page)

(continued from previous page)

```
enabled=1
gpgcheck=0
```

4. Refresh the repository list:

```
$ sudo yum clean all
$ sudo yum repolist # make sure only the above repository is listed
```

The repository configuration is now complete. We may proceed to perform ClusterControl installation and subsequently deploy our database clusters without an Internet connection.

### 1.80.3 ClusterControl Offline Installation

For offline installation, ClusterControl now support using the `install-cc` script with offline mode:

1. Install MySQL or MariaDB server for CMON database
2. Download `install-cc` and install the ClusterControl

#### Install MySQL/MariaDB

MariaDB will be used by ClusterControl to store configuration and monitoring data for its operation.

1. On the ClusterControl server, run the following commands to install MariaDB and configure the MariaDB root password:

```
$ dnf clean all
$ dnf repolist
$ dnf install mariadb mariadb-server
$ systemctl enable mariadb
$ systemctl start mariadb
$ mysqladmin -uroot password yourR00tP4ssw0rd
```

   Or you can install the MySQL and configure the password:

```
$ dnf install mysql-community-server
$ systemctl start mysqld
$ systemctl status mysqld
$ systemctl enable mysqld
```

2. Check the default `root` password for MySQL in `/var/log/mysqld.log`. Later you can reset the root password based on the policy.

```
mysql -u root -p
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'xxx';
```

#### Install ClusterControl

1. Download the `install-cc` from this link and put the file into ClusterControl hosts. Give execute permission to the file:

```
$ chmod +x install-cc
```

2. The `install-cc` support offline installation by adding the parameter `OFFLINE=1`. Execute the following in ClusterControl host for the installation:

```
$ S9S_ADMIN_EMAIL="user@example.com" OFFLINE=1 S9S_ROOT_PASSWORD="XXX"␣
↪S9S_CMON_PASSWORD="YYY" ./install-cc
```

> **Note**
>
> Note Fill the `S9S_ADMIN_EMAIL` using your email, `S9S_ROOT_PASSWORD` with the correct password and `S9S_CMON_PASSWORD` you defined.

3. Run the installer script to finish configuring ClusterControl and all of its components (we prefixed the command with a variable `S9S_ADMIN_EMAIL` to be passed to the script to simplify the installation).

   Answer all the questions to complete the installation. Specify the MySQL/MariaDB root password when asked, as configured under the Install MySQL/MariaDB section.

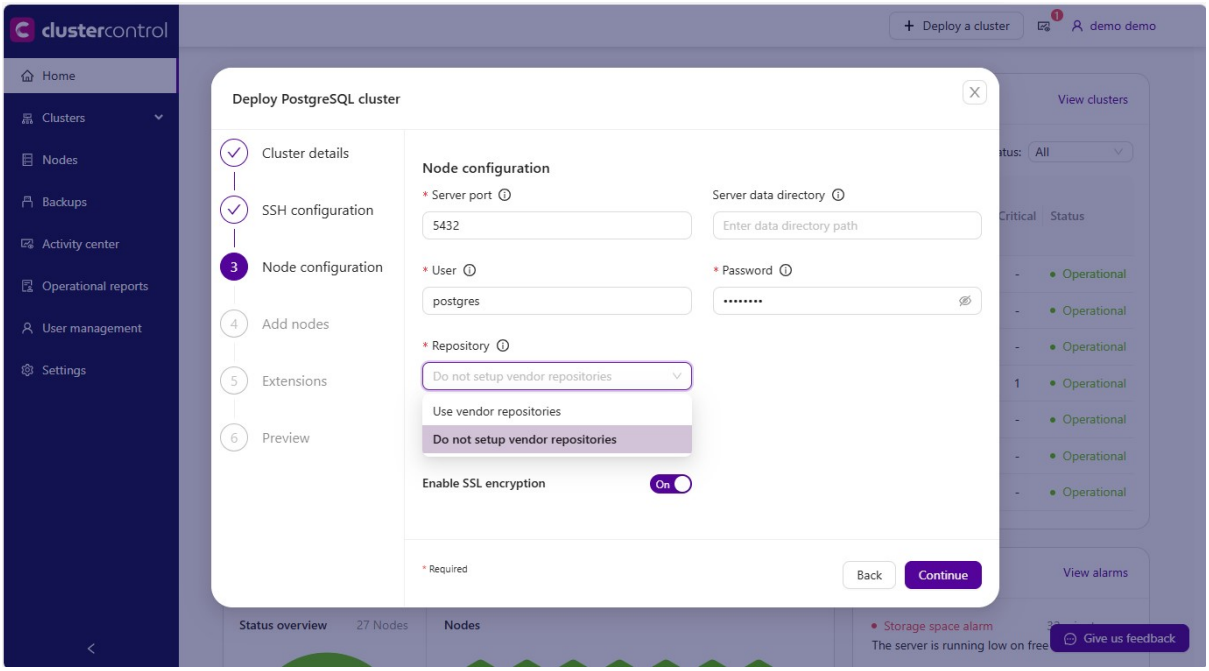
   You should see the following lines:

```
ClusterControl installation completed!
Open your web browser to https://10.0.2.15 and create a default Admin␣
↪User.
If you want to uninstall ClusterControl then please follow the␣
↪instructions here,␣
↪https://severalnines.com/docs/administration.html#uninstall
```

   You may ignore the error because the installer script was trying to reach out to the internet for reverse lookup which is expected. Open the ClusterControl UI and register a super admin user to start deploying database clusters.

### 1.80.4 Offline Database Deployment using ClusterControl

Once you are logged in to ClusterControl UI, you may proceed to deploy a database cluster by going to ***ClusterControl → Deploy a cluster*** and filling up all necessary details. One thing in particular for this kind of environment, you have to choose **"Do Not Setup Vendor Repositories"** under the ***Repository*** dropdown, since we already preconfigured the repository definition as shown in the *Configure Database Repository* section:




If you would like to deploy a cluster by using CluserControl CLI for the deployment, do not forget to specify `--use-internal-repos` flag, as shown below:

```
$ s9s cluster --create \
        --cluster-type=postgresql \
        --nodes=
↪"192.168.100.211?master;192.168.100.212?slave;192.168.100.213?slave;" \
        --db-admin="postgres" \
        --db-admin-passwd="mySuperStongP455w0rd" \
        --cluster-name=ft_replication_23986 \
        --os-user=root \
        --os-key-file=/root/.ssh/id_rsa \
        --provider-version=13 \
        --use-internal-repos \
        --log
```

That's it. The database deployment should be able to use the satellite repository server for an offline installation.

### Setting up Prometheus exporters

For the offline installation of Prometheus and its exporters, you need to download all the required packages first and put the packages in the ClusterControl node (put in under `/var/cache/cmon/packages`).

The URL of the packages can be found in `/usr/share/cmon/templates/packages.conf` as shown below:

```
$ cat /usr/share/cmon/templates/packages.conf
#
# ClusterControl central URL database for
# 3rd party packages installed directly
# (not using APT/YUM repos)
#
```

(continues on next page)

(continued from previous page)

```
[maxscale]
# clustercontrol will substitute the following keywords:
# @VENDOR@  : debian/ubuntu...
# @RELEASE@ : 5,6,7,xenial,wily...
url_deb=
↪"https://downloads.mariadb.com/MaxScale/2.5.7/packages/@VENDOR@/@RELEASE@/maxscale-2.5.
tected]_64.deb"
url_rpm=
↪"https://downloads.mariadb.com/files/MaxScale/2.5.7/packages/rhel/@RELEASE@/maxscale-2.
tected]_64.rpm"

[haproxy]
url_source="http://www.haproxy.org/download/1.8/src/haproxy-1.8.9.tar.gz"

[keepalived]
url_source="http://www.keepalived.org/software/keepalived-1.2.24.tar.gz"

[epel]
# To be substituted:
# @RELEASE@ : 6,7,8
url_rpm=
↪"http://dl.fedoraproject.org/pub/epel/epel-release-latest-@[email pro-
tected]"

[prometheus]
# requiredVersion: if such or greater version found
# clustercontrol will use the existing installed package
requiredVersion=2.29
url=
↪"https://github.com/prometheus/prometheus/releases/download/v2.29.2/prometheus-2.29.2.l

[haproxy_exporter]
requiredVersion=0.9
url=
↪"https://github.com/prometheus/haproxy_exporter/releases/download/v0.9.0/haproxy_export

[node_exporter]
requiredVersion=1.0.1
url=
↪"https://github.com/prometheus/node_exporter/releases/download/v1.0.1/node_exporter-1.0

[mysqld_exporter]
requiredVersion=0.13
url=
↪"https://github.com/prometheus/mysqld_exporter/releases/download/v0.13.0/mysqld_exporte

[postgres_exporter]
requiredVersion=0.4.7
```

(continues on next page)

(continued from previous page)

```
url=
↪"https://github.com/wrouesnel/postgres_exporter/releases/download/v0.4.7/postgres_expor

[pgbouncer_exporter]
requiredVersion=0.4.0
url=
↪"https://github.com/prometheus-community/pgbouncer_exporter/releases/download/v0.4.0/pg

[proxysql_exporter]
requiredVersion=1.1
url=
↪"https://github.com/percona/proxysql_exporter/releases/download/v1.1.0/proxysql_exporte

[process_exporter]
requiredVersion=0.10.10
url=
↪"https://github.com/kedazo/process_exporter/releases/download/0.10.10/process_exporter-

[mongodb_exporter]
requiredVersion=0.11.0
url=
↪"https://github.com/kedazo/mongodb_exporter/releases/download/v0.11.0/mongodb_exporter-

[redis_exporter]
requiredVersion=1.15.0
url=
↪"https://github.com/oliver006/redis_exporter/releases/download/v1.15.0/redis_exporter-v

## for MSSQL
[mssql_exporter]
requiredVersion=0.5.4
url=
↪"https://github.com/severalnines/mssql_exporter/releases/download/0.5.4/mssql_exporter-

[daemon]
url_rpm="http://libslack.org/daemon/download/daemon-0.6.4-1.x86_64.rpm"
```

After the dependencies exist on the server, you may proceed to enable Prometheus agent-based monitoring by going to ***ClusterControl → Dashboards → Enable Agent-based Monitoring***.

In this article, we explore how to deploy a secondary cluster and failover with ClusterControl. A secondary cluster provides high availability and disaster recovery, ensuring minimal downtime in case of primary cluster failure.

## 1.81 Use cases

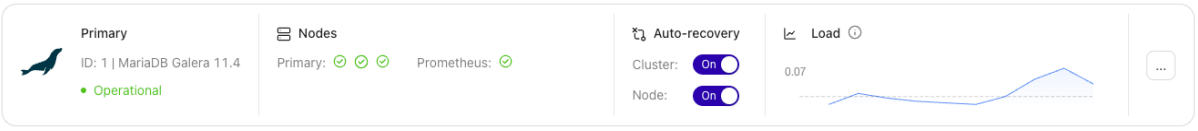
Deploying and failing over to a secondary cluster is useful in the following scenarios:

- Ensuring high availability in case of primary cluster failure.
- Implementing disaster recovery strategies for business continuity.

- Performing planned maintenance without downtime.
- Distributing read traffic across multiple clusters.
- Enabling quick recovery from unexpected infrastructure issues.

## 1.82 Deploying a Secondary Cluster

To deploy a Secondary Cluster from ClusterControl, first, you need to have a Primary cluster in place.

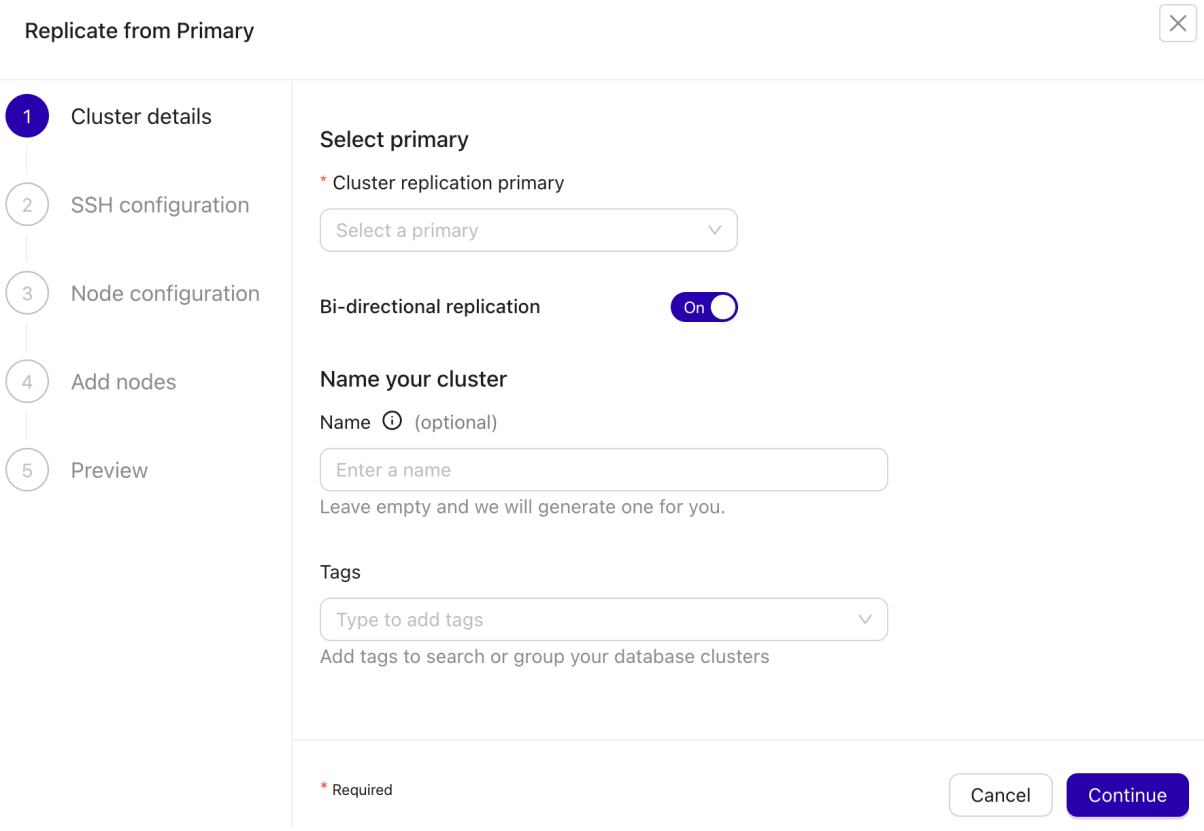



For this, follow the corresponding instructions here: *Create Database Cluster*.

After you have the Primary cluster deployed, go to ***ClusterControl GUI → Clusters → Cluster Actions → Clone/Replicate the cluster → Create replica cluster***.

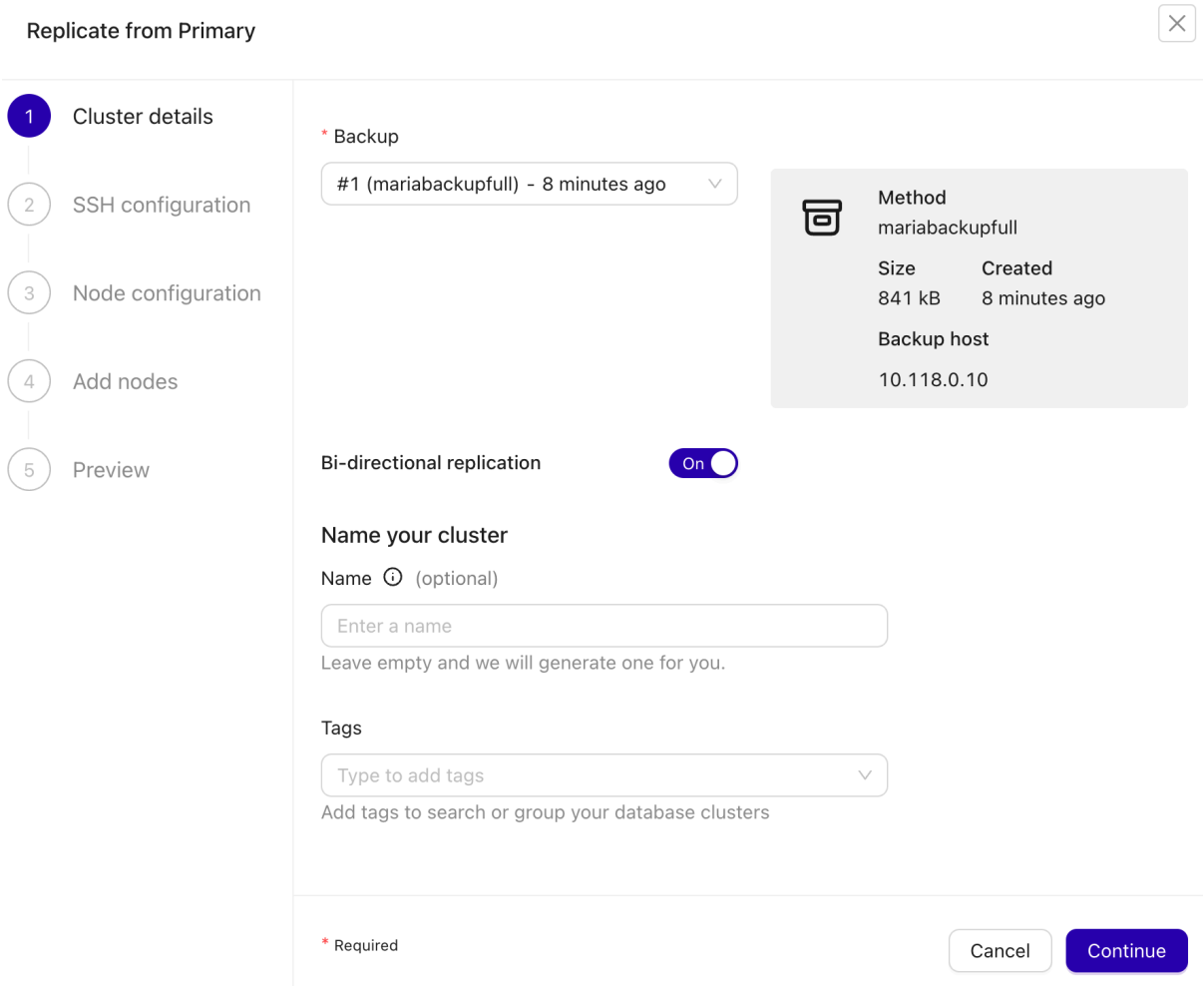
There, you can choose if you want to get data from a primary cluster or from an existing backup.

### 1.82.1 Replicate from Primary




1. When replicating from a primary cluster, choose the node from where you are going to replicate data from and specify if it is a bidirectional replication.
2. Configure the SSH credentials and security configuration.
3. Specify the node configuration like server port, data directory, version, etc.
4. Add the node IP address or hostname to be added to the secondary cluster.

### 1.82.2 Replicate from Backups

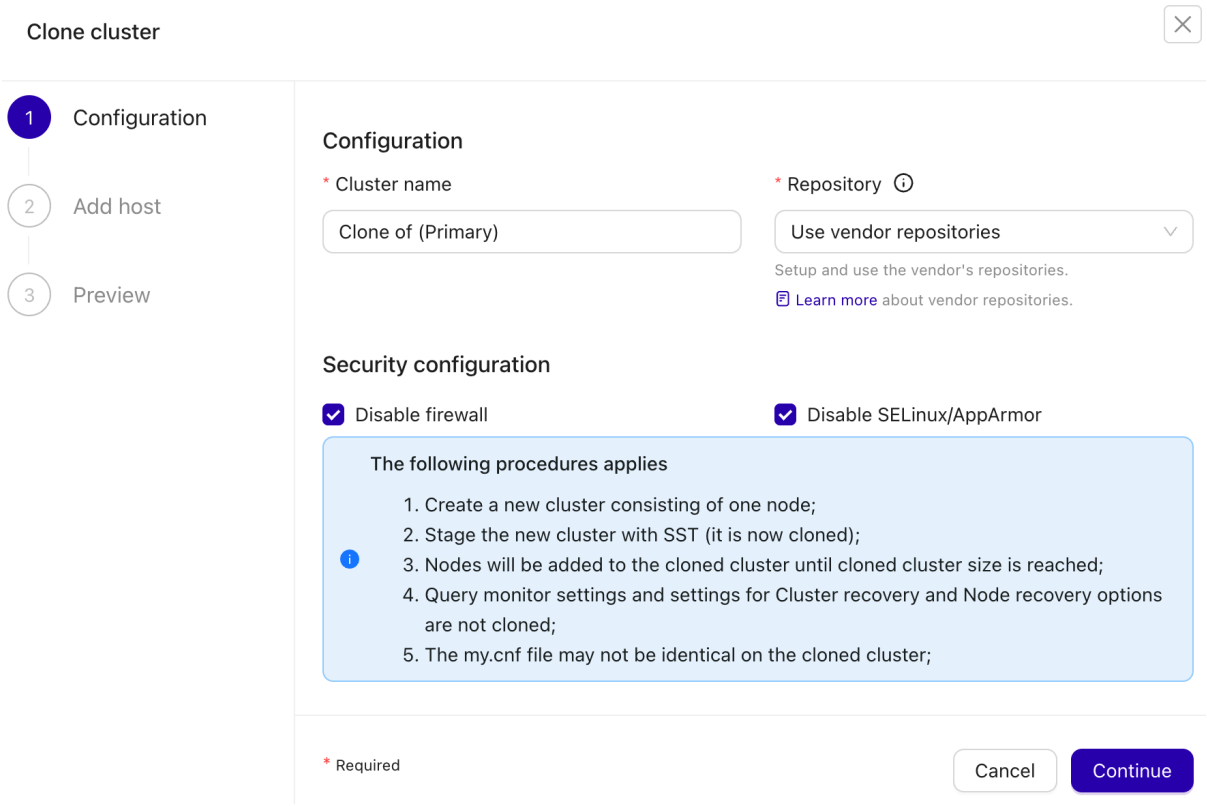



1. When replicating from backups, choose the backup to be restored in the new cluster and specify if it is a bidirectional replication.
2. Configure the SSH credentials and security configuration.
3. Specify the node configuration like server port, data directory, version, etc.
4. Add the node IP address or hostname to be added to the secondary cluster.

### 1.82.3 Clone cluster

In case you don't want to have the latest data replicated to the new cluster, you can just clone the existing cluster and keep it as a separated cluster for testing or datawarehousing.

For this, after you have the primary cluster deployed, go to ***ClusterControl → Clusters → Cluster Actions → Clone/Replicate the cluster → Clone cluster***.




This will create a exact copy of the primary cluster without configuring a replication between the primary and the new cluster.

## 1.83 Failover

To failover to the Secondary cluster, you need to disable read-only to be able to write data on it, and change the endpoint to use the new cluster.

**Note**

Note If you are using a load balancer, depending on the configuration, most probably the endpoint change won't be necessary.

When the Primary cluster is back, if the replication configured between the clusters was bi-directional, the cluster will get the latest data when the replication lag reach 0. Otherwise, you have to rebuild the Primary cluster from the Secondary one and enable read-only after failover back, then update the endpoint again.

## 1.84 Cluster Replication Status using CLI

To see the Cluster replication status in the CLI, run:

```
s9s replication --list --long
```

You will get an output like this:

```
CID      SLAVE                MASTER               STATUS  MASTER_CLUSTER  LAG
1        10.10.10.122:3306    10.10.10.125:3306    Online  3               0
```

(continues on next page)

(continued from previous page)

```
3       10.10.10.125:3306   10.10.10.122:3306   Online  1               0
Total: 2 replication link(s)
```

In this article, we explore how to perform a PostgreSQL major version upgrade using ClusterControl. A major upgrade involves moving from one major release series to another (for example, PostgreSQL 15 to 16). Because such upgrades may introduce new system catalog structures and changes to internal mechanisms, it is crucial to plan carefully and follow best practices to minimize downtime and avoid data corruption.

## 1.85 Use cases

Here are some common scenarios in which a major PostgreSQL upgrade is necessary:

- **Feature requirements**: New PostgreSQL versions often include performance improvements, new SQL features, and improved security defaults.
- **End-of-Life (EOL)**: Upgrading ensures you stay on a supported version of PostgreSQL to receive security patches and critical bug fixes.
- **Regulatory and compliance**: Some organizations have strict rules about running supported, up-to-date software versions.
- **Performance enhancements**: Certain major versions bring substantial improvements in indexing, partitioning, and query optimizers.

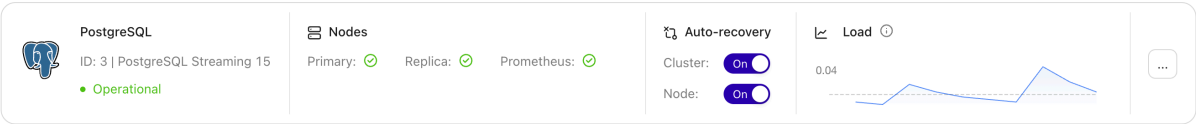
## 1.86 Important considerations

Before undertaking a major upgrade, consider the following points:

- **Backup everything**: Ensure you have a full backup of all databases.
- **Plan downtime**: Major upgrades typically require some downtime or a strategic failover approach.
- **Validate extensions**: Extensions, custom functions, and plugins might need new versions compatible with the target PostgreSQL release.
- **Test**: Always test the upgrade on a non-production environment first.
- **Roll-back plan**: The safest rollback approach is to restore from a full backup of the old version or keep an untouched replica on the old version until the new version is fully validated.

## 1.87 Deploying a PostgreSQL cluster

First, you need to have a PostgreSQL cluster *deployed by ClusterControl* or *imported into ClusterControl*.




After you have the PostgreSQL cluster in place, go to ***ClusterControl → Clusters → Cluster Actions → Upgrades*** where you can chose between `minor` or `major` upgrades.

## 1.88 Major upgrades for PostgreSQL

Choose **Major Upgrades** to performs an upgrade to a later major version for the database software and data files. The upgrade job will upgrade all the nodes in the cluster by default. During the steps, you have to specify temporary ports and upgrade methods.

**Warning**

Attention

Depending on the upgrade method you might require the storage capacity to hold two data sets.

### 1.88.1 Upgrade methods

There are three methods for upgrading your PostgreSQL cluster in ClusterControl:

- **copy**: This is the default method with `pg_upgrade`. The `--copy` option is used which will copy files at a time. It is faster than doing regular backup and restore. This requires disk storage capacity to hold both the old and the new files.

  **Warning**

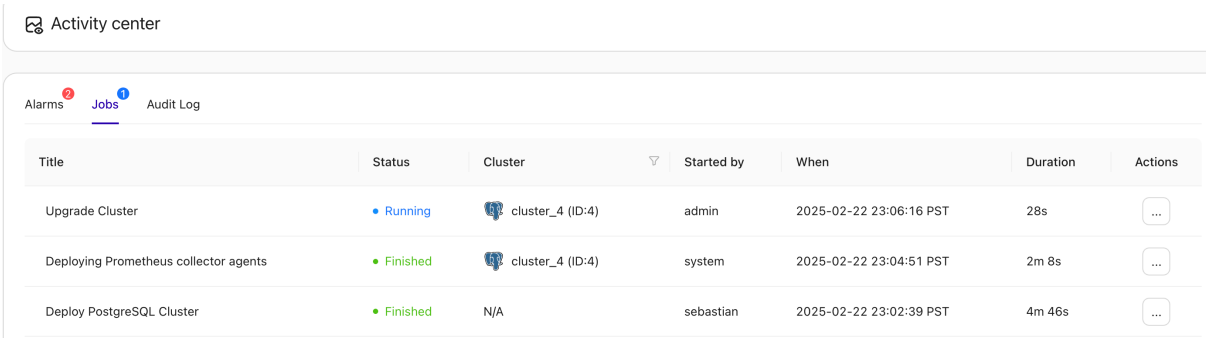
  Attention

  ```
  Data stored in user-defined tablespaces is not being copied to the␣
  ↪new cluster during an upgrade. It remains in its original file␣
  ↪system location but is duplicated into a subdirectory indicating␣
  ↪the version number of the new cluster. To manually relocate files␣
  ↪that are stored in a tablespace after upgrading, move the files to␣
  ↪the new location and update the symbolic links to point to the␣
  ↪files.
  ```

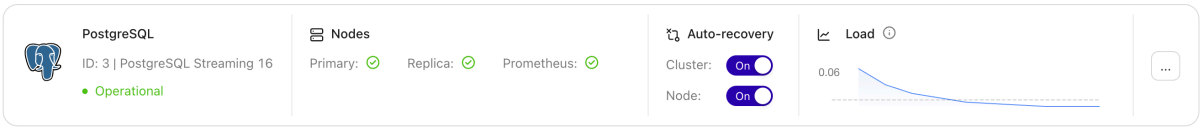
- **link**: This method uses the `--link` option with `pg_ugprade`. Hard links will be used instead of copying files which is faster and requires no extra disk storage. It is crucial to refrain from starting the old cluster after this step to avert any potential data corruption with the new cluster since they share the data.
- **pgdumpall**: It uses pgdumpall to copy files by performing a backup on the old cluster and then restoring it on the new cluster. This requires disk storage capacity to hold the backup.

### 1.88.2 Monitoring upgrade process

To monitor the upgrade job in ClusterControl, go to ***ClusterControl → Activity center → Jobs*** where you can see the running jobs in the UI.




When the job in completed, you will see the cluster upgraded in the ClusterControl UI.




## 1.89 Using the ClusterControl CLI

You can also perform major upgrades by using the ClusterControl CLI from the ClusterControl node.

```
s9s cluster \
    --upgrade-cluster \
    --upgrade-to-version=16 \
    --upgrade-method=link \
    --cluster-id=1 \
    --log
```

This command will upgrade cluster packages to a newer major version 16 for cluster ID 1 with `link` as upgrade method.

### 1.89.1 Monitoring the upgrade

```
s9s job \
    --list \
    --long
```

You will get something like this:

```
Upgrade Cluster
Installing software                                                    [
                                                                    20.00%

Created   : 2025-02-23 07:06:16    ID    : 19         Status : RUNNING
Started   : 2025-02-23 07:06:16    User  : admin      Host   : 127.0.0.1
Ended     :                       Group : admins     Cluster: 4
Tags      : -
RPC       : 2.0
```

**Note**

Note During the upgrade process, the cluster will not be operational. A major upgrade is performed at your own risk. Severalnines will not support any breaking issues that might occur during this operation.

In this article, we explore how to configure and manage backups in the cloud using ClusterControl. Storing and retrieving your database backups offsite helps ensure robust disaster recovery strategies, reduces on-premise storage needs, and simplifies the process of restoring data in the event of a system failure or data corruption.

## 1.90 Use cases

Storing backups in the cloud is useful for various scenarios, including:

- **Disaster Recovery**: Keeping copies of your data offsite in cloud storage helps ensure you have reliable restore points in the event of on-premise system failures.
- **Storage Optimization**: Offloading backup files to cloud storage can help reduce local disk usage and associated costs.
- **Compliance Requirements**: Many organizations must maintain offsite backups to meet compliance mandates.
- **Scalability**: Cloud storage services can easily scale as your backup size grows, eliminating the complexity of managing additional on-premise infrastructure.
- **Testing & Development**: Quickly restoring backups from cloud storage to dev/test environments makes it easy to replicate real-world data for application testing.

## 1.91 Configuring Cloud Storage

Before you can perform backups to or restores from the cloud, you need to configure storage settings in ClusterControl. Depending on your preferred cloud provider (**Amazon S3**, **Google Cloud Storage**, **Azure Blob Storage**), the general steps are similar.

1. **Obtain Your Cloud Credentials**
   - For **Amazon S3**, gather the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_REGION`.
   - For **Google Cloud**, you should have a service account key (JSON file) that you can point to.
   - For **Azure**, you need to create an storage account and get information such as the `application_id`, `client_secret`, `tenant_id`, etc.
   - For **S3-Compatible Storage Provider**, required information is similar to the **Amazon S3** one.
2. **Configure Cloud storage credentials in ClusterControl**

In the ClusterControl web interface, navigate to Settings > Cloud storage credentials, and add a new storage profile for your cloud provider. Here you can specify:

- Provider: For example, S3
- Credential/Keyfile: GCP JSON key, or AWS key references
- Region: if needed

Once saved, ClusterControl will be aware of your cloud storage location for backup.

## 1.92 Creating a Backup in the Cloud

Once the cloud storage is configured, perform a backup by following these steps:

1. Navigate to Backup > Create backup. (Or select your cluster and go to Backups)
2. Choose Backup on Demand
3. Select your database cluster
4. Choose the backup method (`mysqldump`, `pg_basebackup`, `xtrabackup`)
5. Enable Upload backup to cloud
6. Follow the next steps
7. Click Create

Monitor the backup process from the Backup tab. Once completed, your backup will be available in the configured cloud storage provider.

## 1.93 Restoring from the Cloud

To restore a backup from the cloud:

1. Navigate to Backup > Backup Storage and verify the stored backups.
2. Select the backup you want to restore.
3. Click Restore and choose the target database cluster.
4. Confirm the restore process.

ClusterControl will download the backup from the cloud and restore it to the specified database cluster.

## 1.94 Automating Backups

To automate backups, set up a schedule backup:

1. Navigate to Backup > Create backup
2. Choose Schedule a Backup
3. Select your database cluster
4. Choose the backup method (`mysqldump`, `pg_basebackup`, `xtrabackup`)
5. Enable Upload backup to cloud
6. Follow the next steps
7. Set the schedule (`daily`, `weekly`, etc)
8. Click Create

ClusterControl will automatically create backups at the specified intervals and store them in the cloud.

## 1.95 Verifying Backups

It is a best practice to periodically verify that your backups can be restored successfully. To automate this, use the Verify backup feature in ClusterControl, or perform the following steps manually:

1. Spinning up a temporary replica or development environment
2. Download the backup from the cloud (it can be done from the ClusterControl UI)
3. Restoring the cloud backup
4. Performing integrity checks on the restored data

## 1.96 Accessing Cloud Backups via CLI

Below is an example of how to create and restore backups from the command line. These commands should be run on the ClusterControl server.

- List clusters to find the `--cluster-id`

```
s9s cluster --list --long
```

- Create a backup job to the configured cloud storage

```
s9s backup \
    --create \
    --cluster-id=10 \
    --backup-method=xtrabackupfull \
    --cloud-only \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --wait
```

- List Available Backups

```
s9s backup --list --long
```

- Restore a Backup

```
s9s backup \
    --restore \
    --backup-id=5 \
    --cluster-id=10 \
        --wait
```

In this article, we explore how to perform a **Database Point-in-time Recovery (PITR)** using ClusterControl. Point-in-time recovery allows you to restore your database to a specific time, often just before a disruptive event (for example, accidental data deletion or corruption). By combining full backups with transaction logs (`binlogs`, `WAL`, or equivalent), ClusterControl can help you recover to a precise moment to minimize data loss.

## 1.97 Use cases

Point-in-time recovery is useful in scenarios such as:

- **Accidental Data Modification**: A user or application drops a table or updates critical rows by mistake.
- **Data Corruption**: Software or hardware errors that corrupt database pages or files, requiring you to roll back to a clean state.
- **Ransomware or Security Incidents**: Restoring to the moment before an intrusion or malicious activity was detected.
- **Testing & Compliance**: Validating that your backups and binlogs/WAL archives can reconstruct the database to any point in time to meet audit or compliance requirements.

## 1.98 Configuring point-in-time recovery

### 1.98.1 Enable transaction log archiving

Depending on your database type, you must ensure the transaction logs are retained long enough to allow PITR. Below are examples for MySQL-based and PostgreSQL-based systems:

#### MySQL/MariaDB

- Make sure binary logging is enabled and that the retention is appropriately set in the MySQL configuration file (usually `my.cnf`):

```
[mysqld]
log_bin                = binlog
binlog_format          = ROW
expire_logs_days       = 7
server_id              = 100
```

- Restart the MySQL/MariaDB server for changes to take effect.

#### PostgreSQL

- Configure continuous WAL archiving by editing `postgresql.conf`:

```
wal_level = replica
archive_mode = on
archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p /
↪mnt/server/archivedir/%f'
```

- Ensure sufficient storage space for WAL archives and that the retention policy aligns with your PITR requirements.
- Restart PostgreSQL for changes to take effect.

**Note**

Note You can also enable binary logging from the ClusterControl UI.

### 1.98.2 Enable full backup

Before PITR can be performed, you need consistent base backups:

#### Choose a backup method

- In the ClusterControl UI, create or scheduled a backup using a supported method (`mysqldump`, `xtrabackup`, `pg_basebackup`, etc).
- Navigate to **Backups** on your cluster.
- Create or schedule a backup job.
- Verify the backup storage location (local or/and cloud).

#### Validate backups

- Check that backups are running successfully (in **Jobs** or **Backups** sections).
- Periodically restore test backups to ensure they are valid (ClusterControl Verify backup feature helps on this task).

## 1.99 Performing point-in-time recovery

Once full backups and transaction log archiving are in place, initiate a PITR from ClusterControl.

### 1.99.1 Using the ClusterControl GUI

1. **Identify the desired recovery time**
   - Determine the timestamp just before an unwanted change or event.
   - Alternatively, note the binlog or WAL position if you have that information.
2. **Navigate to the Backups section**
   - Under the **Backups** section, locate a full backup that predates your desired recovery point.
   - Select **Restore**.
3. **Confirm how to restore the backup**
   - Restore on node or on standalone host.
4. **Specify the recovery point**
   - Enable PITR and provide a timestamp or binlog/WAL sequence.
   - Adjust any additional settings as needed (`Temporary directory`).
5. **Initiate the PITR job**
   - ClusterControl will restore the base backup.
   - After the base restore completes, it will replay the transaction logs or WAL files up until the specified time.
   - Monitor the **Jobs** tab or **Logs** to track progress.

### 1.99.2 Using the ClusterControl CLI

Below is an example flow for using ClusterControl CLI for MySQL/MariaDB point-in-time recovery tasks:

1. List clusters and backups:

```
s9s backup --list --long
```

   Identify the `--cluster-id` (**CID**) and `--backup-id` (**ID**) of your full backup.

2. Initiate PITR:

```
s9s backup \
        --restore \
        --cluster-id=2 \
        --backup-id=3 \
        --pitr-stop-time="2025-02-10 23:20:09" \
        --nodes="10.118.0.3:3306" \
        --log
```

   Where:

   - `--cluster-id`: ID for the cluster from the previous step.
   - `--backup-id`: ID for the backup from the previous step.
   - `--pitr-stop-time`: the timestamp in `YYYY-MM-DD HH:MM:SS` format to which you want to roll forward.
   - `--nodes`: specify which node to restore on.
   - `--log`: Watch the logs in real time.

3. Monitor the restore job:

```
s9s job --list --long
```

   Check the job status to confirm when the PITR operation completes.

### 1.99.3 Verifying the recovery

After the point-in-time recovery job completes:

1. In the ClusterControl UI or CLI logs, ensure there are no errors or warnings.
2. Query your restored database to confirm data is as expected for the specified point in time.
3. Reconnect your application to the restored node/cluster and validate functionality.

In this article, we explore how to configure a dedicated read-only user for integration with the ClusterControl platform through the RPC interface. Establishing a read-only user is essential for secure and efficient access, allowing ClusterControl to retrieve necessary information without risk to data integrity or unauthorized modifications.

## 1.100 Use cases

Creating a read-only user is useful in the following example use cases:

- Using a dedicated read-only user for ClusterControl prevents interference with other user accounts or privileges that might be required for application functions.
- The read-only user can run non-invasive health checks on the database, like checking replication status, server load, and resource utilization, to provide valuable insights for proactive management.
- The read-only user can be used to programatically integrate with your existing monitoring and alerting tools.
- For environments with strict compliance requirements, such as financial or healthcare databases, having a read-only user enables ClusterControl integration without violating access policies.
- The read-only user can access ClusterControl via GUI, CLI and RPC API and suitable for non-administrative personnel for various reasons such as assessing the monitored infrastructure, analyzing the current state and trend, auditing and reporting.

## 1.101 Creating a read-only user

Suppose there is a two-node PostgreSQL streaming replication being managed and monitored by a ClusterControl server, 192.168.20.10.

The following commands should be executed on the ClusterControl server.

1. Add the primary IP address of the ClusterControl node inside `/etc/default/cmon`:

```
RPC_BIND_ADDRESSES="192.168.20.10,127.0.0.1"
```

2. Restart `cmon` service to make sure ClusterControl listens on port 9501 for both IP addresses defined above:

```
systemctl restart cmon
```

3. Create a read-only user called “readeruser”:

```
s9s user \
    --create \
    --group=readergroup \
    --create-group \
    --generate-key \
    --new-password=s3cr3tP455 \
    --email-address=reader@somewhere.com \
    --first-name=Reader \
    --last-name=User \
    --batch \
    readeruser
```

> **Note**
>
> Note A new private and public key will be created under `~/.s9s` directory called `readeruser.key` and `readeruser.pub`.

4. List out the user and make sure it is there:

```
$ s9s user --list --long
A ID UNAME                        GROUPS      EMAIL                 ␣
↪REALNAME
- 1 system                        admins      - System User
- 2 nobody                        nobody      - Default User
- 3 admin                         admins      - Default User
- 4 s9s-error-reporter-vagrant    admins      - -
A 5 dba                           admins      - -
- 6 readeruser                    readergroup reader@somewhere.com␣
↪Reader User
```

5. Get the Cmon Directory Tree (CDT) value for your cluster (imagine this as `ls -al` in UNIX). In this example, our cluster name is "PostgreSQL 12", as shown below:

```
$ s9s tree --list --long
MODE        SIZE OWNER                          GROUP  NAME
crwxrwx---+    - system                         admins PostgreSQL 12
srwxrwxrwx     - system                         admins localhost
drwxrwxr--  1, 0 system                         admins groups
urwxr--r--     - admin                          admins admin
urwxr--r--     - dba                            admins dba
urwxr--r--     - nobody                         admins nobody
urwxr--r--     - readeruser                     admins readeruser
urwxr--r--     - s9s-error-reporter-vagrant     admins␣
↪s9s-error-reporter-vagrant
urwxr--r--     - system                         admins system
Total: 22 object(s) in 4 folder(s).
```

6. Assign read permission for readeruser and readergroup for the cluster that you want. Our CDT path is "/PostgreSQL 12" (always start with a "/", similar to UNIX):

```
$ s9s tree --add-acl --acl="group:readergroup:r--" "/PostgreSQL 12"
Acl is added.
$ s9s tree --add-acl --acl="user:readeruser:r--" "/PostgreSQL 12"
Acl is added.
```

> **Note**
>
> Note You could also assign permission to a specific path like for example, `"/PostgreSQL 12/192.168.20.62:5432"`, which allows "readeruser" to read the objects of host 192.168.20.62 under this particular cluster.

The configuration is complete. Next, we need to configure the `s9s` client in the reader's workstation.

## 1.102 CLI client configuration

The following commands should be executed on the reader's workstation (client).

1. Install ClusterControl CLI. See *Installation*.
2. Create a `.s9s` directory and copy the `readeruser.key` and `readeruser.pub` from the ClusterControl server into it:

```
mkdir ~/.s9s
cd ~/.s9s
scp root@192.168.20.10:~/.s9s/readeruser.key .
scp root@192.168.20.10:~/.s9s/readeruser.pub .
```

3. Create an s9s configuration file at `~/.s9s/s9s.conf`:

```
[global]
cmon_user=readeruser
controller=https://192.168.20.10:9501
```

4. At this point, you should be able to run the following commands:

```
$ s9s cluster --ping
PING Ok 4ms
$ s9s user --whoami
readeruser
```

At this point, the configuration is complete on the client-side. You may run the following commands to list out the cluster's objects (from the reader's workstation):

```
s9s cluster --cluster-id=10 --stat
s9s cluster --cluster-id=10 --stat --print-json     # if you want the␣
↪output to be printed in JSON
```

## 1.103 Accessing the RPC API

If you would like to access the cluster object via HTTP, send the request to the RPCv2 API accessible on port 9501 (TLS enabled). Use one of the following two methods:

1. Pass the username and password inside a nested `authenticate` method in an operation:

```
curl -k 'https://192.168.20.10:9501/v2/clusters' \
-XPOST -d \
```

(continues on next page)

(continued from previous page)

```
'{"operation": "getAllClusterInfo",
"authenticate":
{"password": "s3cr3tP455","username": "readeruser"}}'
```

2. This is the recommended way and is used by all our applications to connect to the ClusterControl controller. Users need to maintain cookies/sessions after the first authentication and send other requests with the corresponding cookies/session. To obtain a session, use the `authenticateWithPassword` operation and connect to the `v2/auth` endpoint:

```
curl -k 'https://192.168.20.10:9501/v2/auth' \
-XPOST -d \
'{"operation":"authenticateWithPassword",
"user_name":"readeruser",
"password":"s3cr3tP455"}' \
-c cookies.jar
```

Then, include cookies.jar with every subsequent read-only request:

```
curl -k 'https://192.168.20.10:9501/v2/clusters' \
-XPOST -d \
'{"operation":"getallclusterinfo"}' \
-b cookies.jar
```

If the RPC call is successful, you should receive a response with `"request_status": "Ok"`, similar to the following output:

```
{
    "controller_id": "683e2b4b-038f-4cf7-93a0-4c3f95b3db70",
    "request_processed": "2024-11-07T19:33:12.701Z",
    "request_status": "Ok",
    "request_user_id": 5,
    "total": 3,
    "clusters":
    [
        ...json content...
    ],
    "debug_messages":
    [
        "RPC V2 authenticated user is 'readeruser'."
    ]
}
```

If the user is trying to execute a non-read-only request, for example, sending a stop node job that requires an “execute” permission:

```
curl -k 'https://192.168.20.10:9501/v2/jobs' \
-XPOST -d '{"cluster_id": 21, "job": {"class_name": "CmonJobInstance",
"job_spec":
{"command": "stop",
"job_dat":{"clusterid": 21,
```

(continues on next page)

(continued from previous page)

```
"node":{"hostname": "192.168.20.53"}}}},
"operation": "createJobInstance"}' \
-b cookies.jar
```

The reply should return `"request_status": "AccessDenied"`, similar to the following output:

```
{
    "controller_id": "9eb8b1d1-f9d1-439e-945e-108982812bee",
    "error_string": "Execute access to cluster 21 for user readeruser is␣
↪denied.",
    "request_processed": "2021-05-04T07:01:38.342Z",
    "request_status": "AccessDenied",
    "request_user_id": 7,
    "debug_messages":
    [
        "RPC V2 authenticated user is 'readeruser'."
    ]
}
```

For details on the list of operations supported by ClusterControl, see *ClusterControl RPC API reference manual*.

In this article, we explore the procedure for migrating existing database clusters from the ClusterControl GUI v1 to the new ClusterControl GUI v2. This migration strategy is designed to ensure continuous operational integrity and minimize downtime throughout the transition. This approach is particularly advantageous for organizations and users who prefer a phased adoption strategy, allowing for thorough testing and validation of the new ClusterControl v2 interface and its enhanced functionalities in a controlled environment before a full-scale deployment. By following these guidelines, users can seamlessly transition their critical database infrastructure, leveraging the advancements offered by ClusterControl v2 while maintaining stability and performance.

## 1.104 Limitations of This Method

Before proceeding, it's important to understand the key limitations:

- All operations that depend on Cluster ID may be impacted due to the change of the IDs.
- You begin with new monitoring data as historical metrics and logs are not migrated.
- Manual tuning may be required for certain settings after migration (e.g., failover thresholds).
- Custom scripts and hooks from GUI v1 require manual recreation, as they will not be imported automatically.
- No rollback. Reverting to GUI v1 after migration and disabling it is not straightforward.

## 1.105 Pre-Migration Checks

Thorough pre-migration checks are essential for a seamless and successful migration. They mitigate risks, prevent data loss, minimize downtime, and guarantee system integrity, averting complications, delays, increased costs, and disruptions:

- In ClusterControl GUI v1, verify the health and stability of your clusters.

- Review custom alerting, operational reports schedules, and backup schedules.
- Make a list of integrations: email, LDAP, cloud backups, notifications, 3rd party integrations etc.
- Make a list of users who log in to ClusterControl GUI v1.
- Ensure SSH access to all nodes is functional for the user running ClusterControl in your new environment with ClusterControl GUI v2.

## 1.106 Installing ClusterControl v2

Follow these steps:

1. Prepare a new host or virtual machine.
2. Download and install ClusterControl GUI v2 from the official installer or repository. On the ClusterControl server, run the following commands:

```
wget https://severalnines.com/downloads/cmon/install-cc
chmod +x install-cc
sudo ./install-cc # omit sudo if you run as root
```

3. Configure passwordless SSH access to the database nodes. See *SSH Key-based Authentication*.

## 1.107 Importing Existing Clusters into ClusterControl v2

The *Import Database Cluster* feature in ClusterControl enables users to bring existing database clusters under the existing ClusterControl v1 management with minimal effort. The following cluster types can be imported from ClusterControl v1 to v2:

- MySQL/MariaDB/Percona Server standalone
- MySQL/MariaDB/Percona Server replication
- MariaDB Galera Cluster
- Percona XtraDB Cluster
- PostgreSQL streaming replication
- TimescaleDB streaming replication
- MongoDB replica set
- MongoDB sharded cluster

The rest of the supported cluster types like Redis/Valkey (Sentinel and Cluster), PostgreSQL logical replication, SQL Server and Elasticsearch are only available on ClusterControl v2.

### 1.107.1 Prerequisites

Before importing a database cluster, several prerequisites must be met:

1. Passwordless SSH (SSH using key-based authentication) is configured from the new ClusterControl v2 node to all database nodes. See *SSH Key-based Authentication*.
2. The target cluster must not be in a degraded state. For example, if you have a three-node Galera cluster, all nodes must be alive, accessible, and in sync.

Below are examples for each supported database type:

### 1.107.2 Importing a MySQL/MariaDB standalone or replication

For this to work, ClusterControl expects all instances within a group to use the same MySQL root password. It will also automatically try to identify each server's role (primary, replica, multi, or standalone).

#### ClusterControl GUI

1. To import an existing MySQL replication cluster, go to ***ClusterControl GUI → Deploy a cluster → Import a database cluster*** and under the **Database** dropdown, choose "MySQL Replication".
2. Under **Cluster details**, specify the cluster details you want to assign:
   - **Name**: This is optional. This shall be the name of your cluster. Once imported, ClusterControl will use this as its registry name of the cluster.
   - **Tags**: Add tags to search or group your database clusters.
3. Click **Continue**.
4. Under the **SSH configuration** section, specify the SSH credentials that ClusterControl should use to connect to the database nodes:
   - **SSH user**: The SSH user that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH user.
   - **SSH user key path**: The SSH private key path that ClusterControl will use to perform SSH authentication to the database node. Relative path is not supported. The SSH private key must be physically secured and existed on the ClusterControl node.
   - **SSH port**: The SSH port that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH port.
   - **SSH sudo password**: The sudo password if the **SSH user** requires a password for privilege escalation.
   - **SSH sudo / OS elevation command**: The OS elevation command (`sudo`, `doas`, `pbrun`) that ClusterControl will uses for non-root account.
5. Click **Continue** to proceed to the next step.
6. Under the **Node configuration** section, specify the database credentials and configurations that ClusterControl shall use when deploying the cluster:
   - **Server port**: The database server port that ClusterControl will use to connect to all database nodes.
   - **Server data directory**: The database server data directory path.
   - **Admin/Root user**: The database admin username that ClusterControl will use to connect to all database nodes. This user must be granted global SUPER privilege and the GRANT option for localhost only.
   - **Admin/Root password**: The password for **Admin/Root user**.
   - **Repository**: Choose "Use vendor repositories" (default) will let ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases) and ClusterControl will skip the repository configuration part.

- **information_schema queries**: By default, this is set to off. When enabled, this shall allow cmon or ClusterControl to query from `information_schema` to collect metrics from your MySQL database nodes.
- **Cluster auto-recovery**: By default, this is set to off. If toggled on, the cluster auto-recovery will be enabled allowing for degraded or failing cluster will be attempted to be recovered by ClusterControl.
- **Node auto-recovery**: By default, this is set to off. If toggled on, the node auto-recovery will be enabled allowing for failing nodes such as accidentally terminated system process will be tried to be started to get back to the cluster.

7. Click **Continue** to proceed to the next step.
8. Under the **Add nodes** section, you can specify your existing target database nodes and you are allowed to add more than one node depending on your existing MySQL Replication cluster you need to register to ClusterControl. Since this is an existing database cluster which means it is already setup, ClusterControl will be able to identify and scan the type of topology for your database cluster.
   - **Node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.

> **Note**
>
> Note

You can only proceed to the next step if all of the specified nodes are reachable (shown in green). ::: 9. Click **Continue** to proceed to the **Preview** page. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the cluster by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The settings will be kept until you exit the import wizard.

10. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Import a three-node Oracle MySQL Replication 8.0 cluster, with operating system user "ubuntu". Regardless of the order, ClusterControl will be able to identify the primary node(s) through `read_only` and `super_read_only` variables:

```
s9s cluster --register \
    --cluster-type=mysqlreplication \
    --nodes="10.0.10.20;10.0.10.21;10.0.10.22" \
    --vendor=oracle \
    --provider-version=8.0 \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='PROD - MySQL Replication 8.0' \
```

(continues on next page)

(continued from previous page)

```
    --wait \
    --log
```

### 1.107.3 Importing a PostgreSQL/TimescaleDB cluster

For this to work, ClusterControl expects all instances within a group to use the same database administrator password. It will also automatically discover the database role of each node (primary, replica, TimescaleDB extension).

#### ClusterControl GUI

1. To import an existing PostgreSQL streaming replication, go to ***ClusterControl GUI → Deploy a cluster → Import a database cluster*** and under the **Database** dropdown, choose "PostgreSQL Streaming" for PostgreSQL or "TimescaleDB" for TimescaleDB.
2. Under **Cluster details**, specify the cluster details you want to assign:
   - **Name**: This is optional. This shall be the name of your cluster.
   - **Tags**: Add tags to search or group your database clusters.
3. Click **Continue**.
4. Under the **SSH configuration** section, specify the SSH credentials that ClusterControl should use to connect to the database nodes:
   - **SSH user**: The SSH user that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH user.
   - **SSH user key path**: The SSH private key path that ClusterControl will use to perform SSH authentication to the database node. Relative path is not supported. The SSH private key must be physically secured and existed on the ClusterControl node.
   - **SSH port**: The SSH port that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH port.
   - **SSH sudo password**: The sudo password if the **SSH user** requires a password for privilege escalation.
   - **SSH sudo / OS elevation command**: The OS elevation command (`sudo`, `doas`, `pbrun`) that ClusterControl will uses for non-root account.
5. Click **Continue** to proceed to the next step.
6. Under the **Node configuration** section, specify the database credentials and configurations that ClusterControl shall use when deploying the cluster:
   - **Server port**: The database server port that ClusterControl will use to connect to all database nodes.
   - **User**: The database admin username that ClusterControl will use to connect to all database nodes. This user must be a global super user.
   - **Password**: The password for **User**.
   - **Repository**: Choose "Use vendor repositories" (default) will let ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not

setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases) and ClusterControl will skip the repository configuration part.

7. Click **Continue** to proceed to the next step.
8. Under the **Add nodes** section, you can specify the target database nodes and configure the database topology that you want to import:
   - **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for single-primary replication.
   - **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes.

> **Note**
>
> Note

You can only proceed to the next step if all of the specified nodes are reachable (shown in green). ::: :::{note} Note If you have more than one secondary or replica nodes, make sure to add more under **Replica nodes** text field. ::: 9. Click **Continue** to proceed to the **Preview** page. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the cluster by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The settings will be kept until you exit the import wizard.

10. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Import a three-node PostgreSQL 16 streaming replication, with operating system user "ubuntu" (the first node is the primary) with tags `production` and `postgres`. Then wait until the job is done while showing logs during the deployment process:

```
s9s cluster --register \
    --cluster-type=postgresql \
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=postgresql \
    --provider-version=16 \
    --db-admin='postgres' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --with-tags='production;postgres' \
    --cluster-name='PostgreSQL 16 - Streaming Replication' \
```

```
                --wait --log
    ```
```

### 1.107.4 Importing a MongoDB replica set

For this to work, ClusterControl expects all instances within a group to use the same MongoDB admin user password. It will also automatically discover the database role of each node (mongos, config server, shard server, primary, secondary, arbiter).

#### ClusterControl GUI

1. To import an existing MongoDB replica set, go to ***ClusterControl GUI → Deploy a cluster → Create a database cluster*** and under the **Database** dropdown, choose "MongoDB ReplicaSet".
2. Under **Cluster details**, specify the cluster details you want to assign:
   - **Vendor**: This is under **Vendor and version** label in the panel. This is the kind of vendor that your database cluster binaries for MongoDB ReplicaSet is deployed. This is required and you need to choose from *Percona*, *MongoDB*, and *MongoDB Enterprise*.
   - **Name**: This is optional. This shall be the name of your cluster.
   - **Tags**: Add tags to search or group your database clusters.
3. Click **Continue**.
4. Under the **SSH configuration** section, specify the SSH credentials that ClusterControl should use to connect to the database nodes:
   - **SSH user**: The SSH user that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH user.
   - **SSH user key path**: The SSH private key path that ClusterControl will use to perform SSH authentication to the database node. Relative path is not supported. The SSH private key must be physically secured and existed on the ClusterControl node.
   - **SSH port**: The SSH port that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH port.
   - **SSH sudo password**: The sudo password if the **SSH user** requires a password for privilege escalation.
   - **SSH sudo / OS elevation command**: The OS elevation command (`sudo`, `doas`, `pbrun`) that ClusterControl will uses for non-root account.
5. Click **Continue** to proceed to the next step.
6. Under the **Node configuration** section, specify the database credentials and configurations that ClusterControl shall use when deploying the cluster:
   - **Server port**: The database server port that ClusterControl will use to connect to all database nodes.
   - **User**: The database admin username that ClusterControl will use to connect to all database nodes. This should have global superuser role (preferably root) and allowed for localhost only.
   - **Password**: The password for **User**.
   - **MongoDB Auth DB**: The authentication database to authenticate to.
```

- **Repository**: Choose "Use vendor repositories" (default) will let ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases) and ClusterControl will skip the repository configuration part.

7. Click **Continue** to proceed to the next step.
8. Under the **Add nodes** section, you can specify the target database nodes and configure the database topology that you want to import:
    - **Node**: Only specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.

> **Note**
>
> Note

You can only proceed to the next step if all of the specified nodes are reachable (shown in green). ::: 9. Click **Continue** to proceed to the **Preview** page. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the cluster by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The settings will be kept until you exit the import wizard.

10. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- To import a three-node MongoDB Replica Set using Percona binaries, you only need to specify the primary node. For example, if you have primary running on 192.168.40.190 and you have replica running on 192.168.40.191, there is no need to specify the replica as it will be auto-detected and will be imported by ClusterControl as well. The following command demonstrates this using `ubuntu` as its OS username, using `ed25519` digital signature algorihtm and print the logs of the job deployment:

```
      s9s cluster --register \
              --cluster-type=mongodb \
              --nodes="192.168.40.190" \
              --vendor=percona \
              --provider-version='7.0' \
              --os-user=ubuntu \
              --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
              --db-admin='mydbuser' \
              --db-admin-passwd='mydbPassw0rd' \
              --cluster-name='MongoDB Percona Server ReplicaSet 7.0' \
              --wait --log
  ```
```

(continues on next page)
```

(continued from previous page)

```
- To import a three-node MongoDB Enterprise Replica Set 7.0, with operating␣
↪system user "root" and let the deployment job run in the foreground:

    ```bash
    s9s cluster --register \
        --cluster-type=mongodbenterprise \
        --nodes="192.168.40.155" \
        --vendor=mongodbenterprise \
        --provider-version='7.0' \
        --os-user=root \
        --os-key-file=/root/.ssh/id_rsa \
        --db-admin='mydbuser' \
        --db-admin-passwd='mydbPassw0rd' \
        --cluster-name='MongoDB Enterprise ReplicaSet 7.0'
    ```

 - To import a MongoDB Sharded Cluster using Percona binaries consisting of␣
↪3-node router nodes using `ubuntu` as its OS username, using ed25519␣
↪digital signature algorihtm and print the logs of the job deployment:

    ```bash
        s9s cluster --register \
                --cluster-type=mongodb \
                --
↪nodes="mongos://192.168.1.11;mongos://192.168.1.12;mongos://192.168.1.12;"
                --vendor=percona \
                --provider-version='7.0' \
                --os-user=ubuntu \
                --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
                --db-admin='mydbuser' \
                --db-admin-passwd='mydbPassw0rd' \
                --cluster-name='MongoDB Percona Server Shards 7.0' \
                --wait --log
    ```
```

## 1.108 Importing Configurations from ClusterControl v1

While cluster monitoring is imported during the steps above, the following configurations must be manually migrated or re-created in ClusterControl v2:

### 1.108.1 Users and Access

- Re-create ClusterControl admin/operator accounts and groups:

#### ClusterControl GUI

To create a user, go to ***Create user or team → Create user*** and fill up all the following information:

| Field | Description |
|---|---|
| *Details* | |
| **First name** | The first name of the user. |
| **Last name** | The last name of the user. |
| **Username** | The username of the user. In CC v2, the username shall be used when logging in to the login page, in contrast to CC v1 (old GUI) where you had to use e-mail address and password combination in order to login. |
| **Password** | The user's password. |
| **Email** | Email address of the user. This should be a valid email address (despite not mandatory), for password reset and receiving alerts via email. |
| **Timezone** | Choose a timezone from the dropdown list. This timezone will reflect the date/time presentation of the ClusterControl events and monitoring data points. |
| *Team* | |
| **Team** | Choose a team from the dropdown. This is considered the primary group for this user. A user can be assigned to multiple groups (teams). |

#### ClusterControl CLI

- List all users:

```
s9s user --list --long
```

**Example**

```
$ s9s user --list --long
A ID UNAME      GNAME  EMAIL REALNAME
-  1 system     admins -     System User
-  2 nobody     nobody -     -
A  3 dba        users  -     -
-  4 remote_dba users  -     -
```

- Create a user called "john" with administrative privilege:

```
s9s user \
    --create \
    --group=admins \
    --generate-key \
```

(continues on next page)

(continued from previous page)

```
    --new-password=s3cr3tP455 \
    --email-address=john@example.com \
    --first-name=John \
    --last-name=Doe \
    --batch \
    john
```

```
See [Users and teams](../../../admin-guide/users-teams.md) for details.
```

- LDAP/AD integration: Replicate the LDAP configuration. See *LDAP* for details.

### 1.108.2 Notification Settings

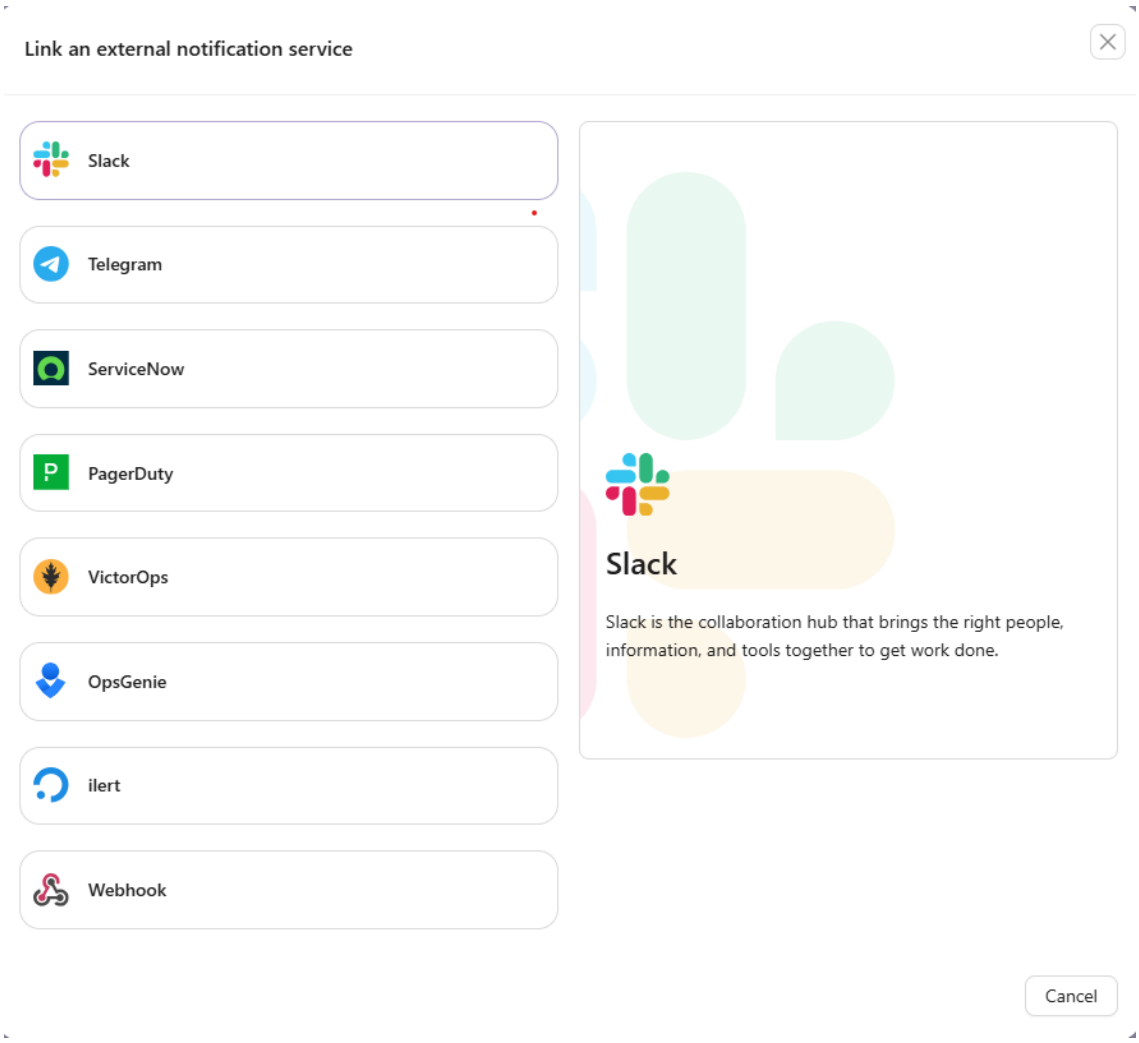
- For email recipients, add them manually, copying from what you have on ClusterControl v1:

#### ClusterControl GUI

1. To configure an SMTP mail server, go to ***ClusterControl GUI → Settings → Configure now***. It will open a wizard called "Configure mail server". Choose **SMTP Server**.
2. In the "Configure mail server: SMTP" window, configure the following:
   - **Server Address**: The SMTP mail server address that you are going to use to send the email.
   - **Port**: The SMTP port for the mail server. Usually, this value is 25 or 587, depending on your SMTP mail server configuration.
   - **User name**: The SMTP user name. Leave empty if no authentication required.
   - **Password**: The SMTP password for **User name**. Leave empty if no authentication required.
   - **Reply to/from**: Specify the sender of the email. This will appear in the 'From' field of the mail header.
   - **TLS/SSL required**: Check this box if ou want to use TLS/SSL for extra security. The mail server must support TLS/SSL.
3. Check the **Send test email** to test out the SMTP configuration. If you receive an email from ClusterControl, click **Save** and proceed to configure the mail recipients.

```
See [Email␣
↪Notifications](../../../user-guide/management/alerts-notifications.md#email-notificatio
↪for details.
```

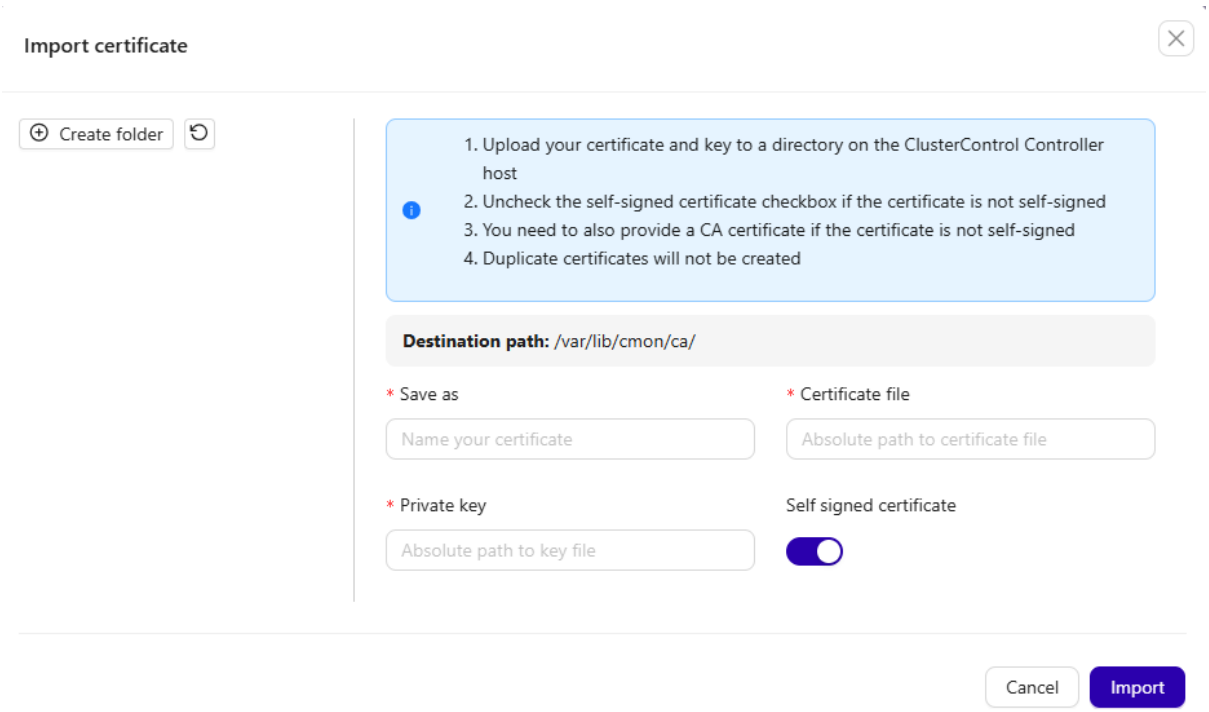
- Third-party integrations (Slack, PagerDuty): Configure again by going to **ClusterControl GUI → Settings → Notification services → Add new integration**. This will give you a dialog box to choose the notification service to use:




See *Notification services* for details.

### 1.108.3 SSL Certificates

Import SSL keys and certificates into ClusterControl's certificate repository. To access this go to **ClusterControl GUI → Settings → Certificate management → More → Import certificate**:



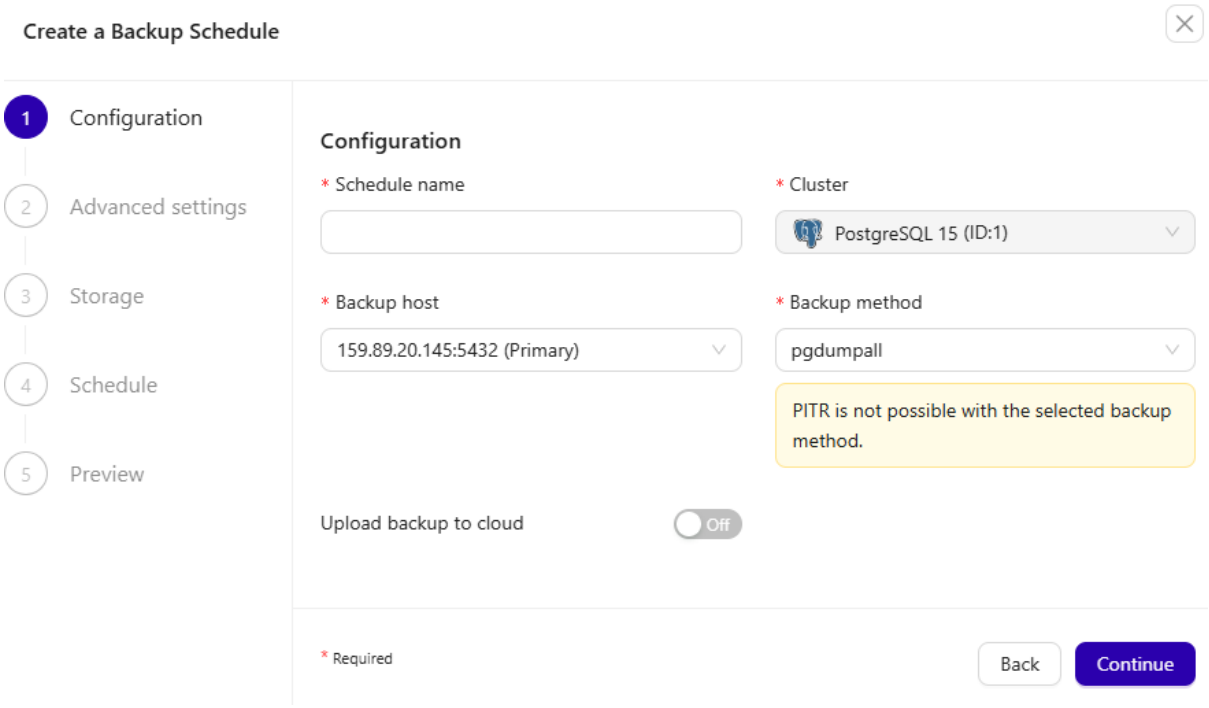

The imported keys and certificates can then be used to enable SSL encryption for server-client connection, replication, or backup at a later stage. Before you perform the important action, bear in mind to:

1. Upload your certificate and key to a directory in the ClusterControl Controller host.
2. Uncheck the **Self-signed Certificate** checkbox if the certificate is not self-signed.
3. You need to also provide a CA certificate if the certificate is not self-signed.
4. Duplicate certificates will not be created.

| Field | Description |
|---|---|
| **Destination Path** | Where you want the certificate to be imported to. Click on the file explorer window on the left to change the path. |
| **Save As** | Certificate name. |
| **Certificate File** | Absolute path to the certificate file. For example: `/home/user/ssl/file.crt`. |
| **Private Key File** | Absolute path to the key file. For example: `/home/user/ssl/file.key`. |
| **Self-signed Certificate** | Uncheck the checkbox if the certificate is not self-signed. |
| **Import** | Start the import process. |

### 1.108.4 Backup Schedules

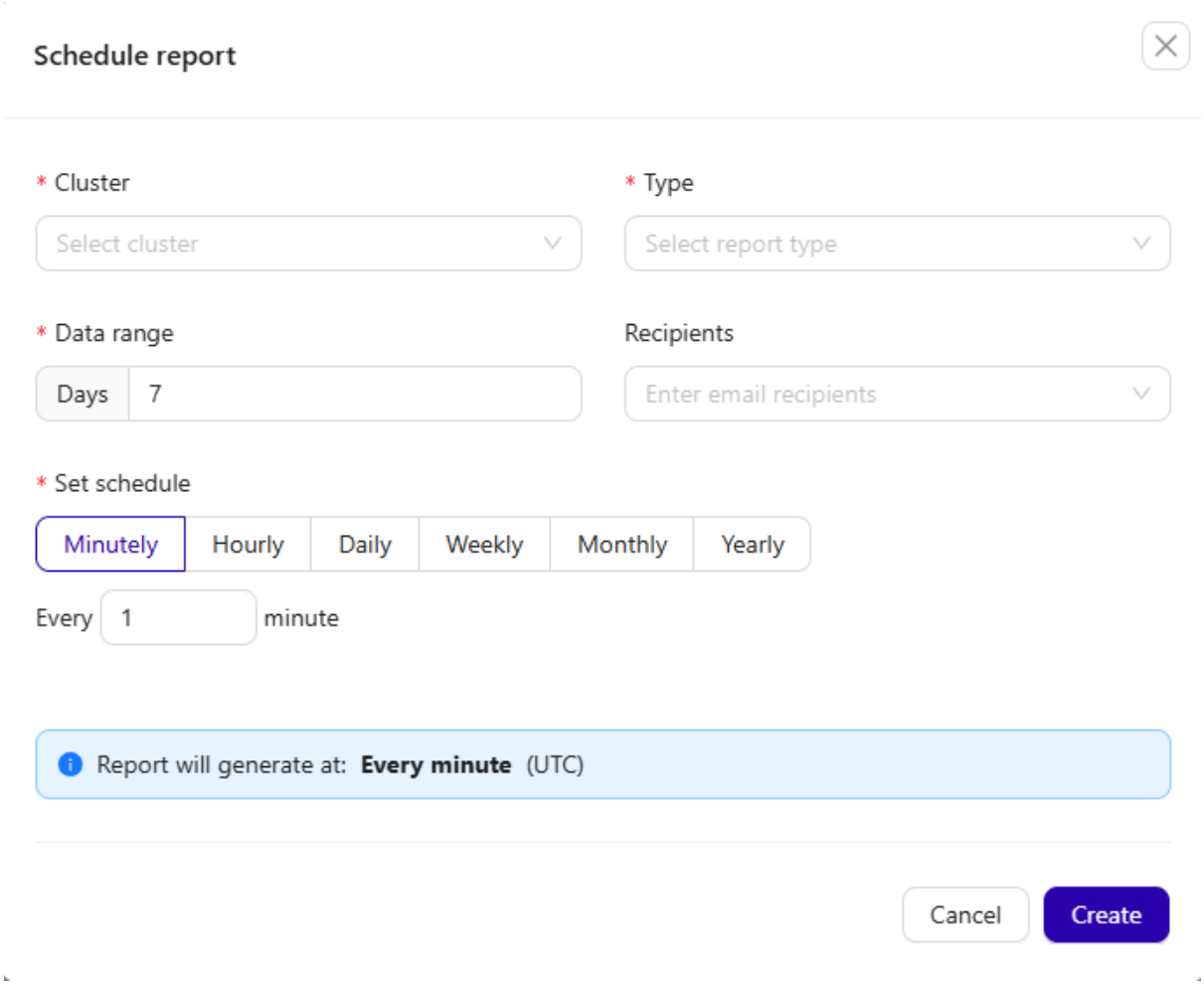
Use the UI or CLI (`s9s backup`) to re-create your backup schedules. To access this go to **ClusterControl GUI → Select cluster → Backups → Create backup → Schedule backup**:




See *Schedule Backup* for details.

### 1.108.5 Scheduled Operational Reports

Set up Scheduled Operational reports (weekly/monthly) and delivery email addresses as needed:




### ClusterControl GUI

1. In the ClusterControl UI, go to ***Sidebar → Operational Reports***.
2. Choose the **Schedules** tab, and create schedule button.
3. Choose the cluster, type of the template reports, time ranges, and fill the email recipients.
4. Set the schedule for delivery of the report.

See *Operational Reports* for details.

## 1.109 Post-Migration Checks

After migration, it is crucial to perform a series of post-migration checks and optimizations to ensure the new system is operating efficiently and reliably. This comprehensive process typically involves:

1. Cluster functionality
   - Regularly verify monitoring systems are operational, accurately collecting data, and capable of timely anomaly notifications. This includes checking dashboards and alert configurations to ensure all metrics are tracked.
   - In GUI v2, proceed to validate the topology diagrams. This crucial step ensures that the visual representation of your system's architecture accurately reflects its actual configuration and intended design.

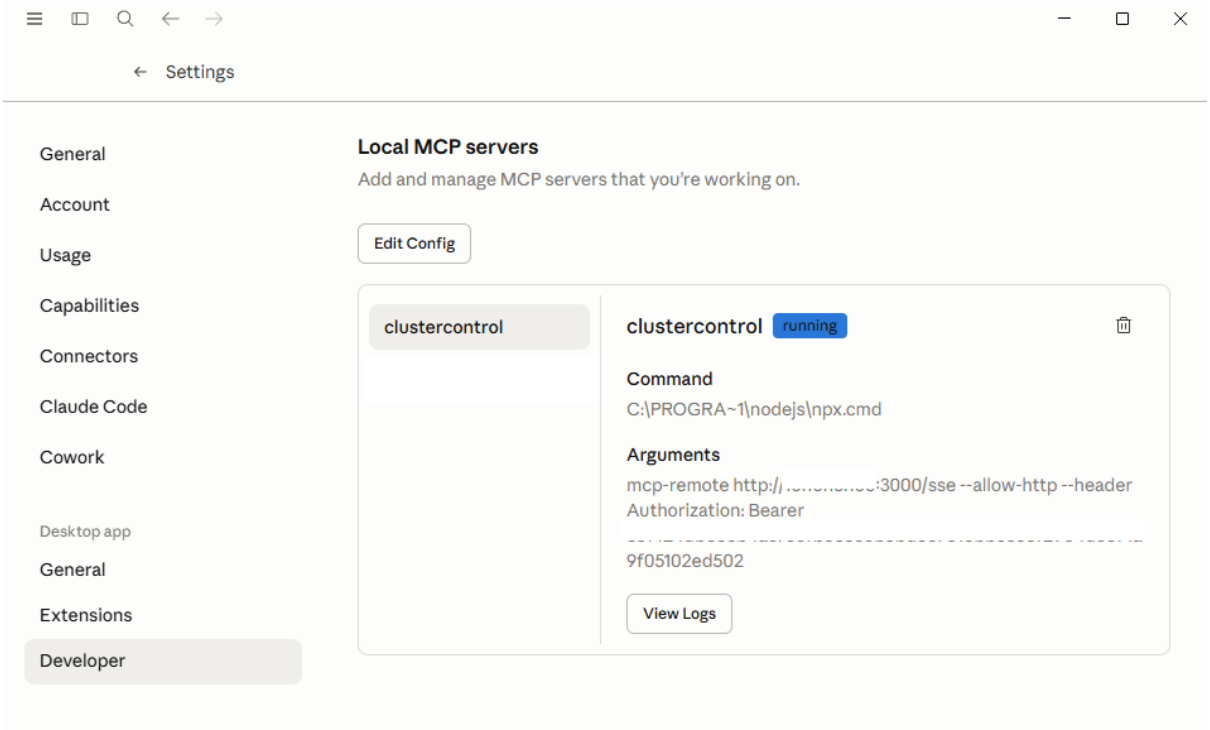
- Simulate a failover to test resilience, observe transition time, and client reconnection. However, before that, ensure data synchronization, cluster health, and application awareness.

2. The Cluster ID will be different from that in GUI v1, requiring immediate updates to all external tools and scripts that rely on it. Failure to update will cause errors.
3. Check alert thresholds, backup schedules, and user privileges:
   - **Alert Thresholds:** Fine-tune CPU, memory, and disk space alerts to ensure effective early warnings without false positives. Verify notification channels and escalation procedures.
   - **Backup Schedules:** Confirm backup frequency/retention, regularly test integrity/restorability, monitor completion, and verify secure offsite storage.
   - **User Privileges:** Conduct access reviews for all the users. Confirming the privileges assigned to users.

In this section, we explore how to manage your database clusters using natural language through the ClusterControl MCP Server. Instead of navigating dashboards or running CLI commands, you simply describe what you want to your AI assistant and it handles the rest.

If you haven't set up the ClusterControl MCP Server yet, please go through the *ClusterControl MCP* documentation first before continuing.

If you have already installed and integrated it with your AI assistant, make sure the integration is working properly before proceeding. For example, if you are using Claude Desktop, go to **Settings → Developer** and confirm that the ClusterControl MCP is listed and running. See screenshot below




Once you can see the MCP Server is running, you are ready to start managing your clusters. The examples below cover the most common operations you can perform through your AI assistant.

Any action that changes something like creating a job, adding a user, scheduling maintenance, restoring a backup, setting node or controller config, etc. Works in two steps:

- Dry run (default): AI Assistants describes exactly what it would do, without touching your CMON controller.

- Execute: After you confirm, AI Assistants calls the tool again with dry_run=false to carry out the action.

High-risk operations (node restart, failover, drop cluster, drop user, delete backup schedule, restore backup, delete ClusterControl account) are marked with a `destructiveHint` annotation and include an additional warning in the dry-run output. MCP clients can use these hints to require your explicit confirmation before running such operations — but this safeguard depends on the client honoring the hints and the dry-run flow; the annotations are advisory metadata, not server-side enforcement.

## 1.110 User and Access Management

Manage database accounts, review privileges, create or revoke access across your clusters.

### 1.110.1 List Functions

Below are the functions that currently supported by clustercontrol-mcp for User and Access Management.

| Function | Purpose |
|---|---|
| `list_db_users` | List all database accounts on a cluster (Required Cluster ID) |
| `create_db_user` | Create a new database user account (Required Cluster ID) |
| `drop_db_user` | Delete a database user account (Required Cluster ID) |
| `grant_db_user` | Grant privileges to a database user (Required Cluster ID) |
| `revoke_db_user` | Revoke specific privileges from a database user (Required Cluster ID) |
| `revoke_all_db_user` | Revoke all privileges from a database user (Required Cluster ID) |
| `list_cmon_users` | List all ClusterControl controller user accounts (Required Cluster ID) |
| `list_cmon_groups` | List all ClusterControl controller user groups (Required Cluster ID) |
| `create_cmon_user` | Create a new ClusterControl controller user account (Required Cluster ID) |
| `set_cmon_user` | Modify a ClusterControl controller user profile (Required Cluster ID) |
| `drop_cmon_user` | Delete a ClusterControl controller user account (Required Cluster ID) |

### 1.110.2 Example Prompt

Here are some examples of using clustercontrol-mcp for User and Access Management:

- List existing users and groups using prompt

```
> List all users and groups that exists in clustercontrol and mask the␣
↪email

<br>
![ClusterControl MCP␣
↪Response](../../../images/clustercontrol-mcp-usman-cmon-1.png)
<br>
```

- Create ClusterControl user and attach to group using prompt

```
> Create a Clustercontrol user with username support and email␣
↪support@severalnines.com using password severalNINES999 in group␣
↪admins

<br>
```

(continues on next page)

(continued from previous page)

```
![ClusterControl MCP␣
↪Response](../../../images/clustercontrol-mcp-usman-cmon-2.png)
<br>
```

## 1.111 Cluster Inventory and Health

Get a complete picture of your clusters, nodes, topology, and active alarms.

### 1.111.1 List Functions

| Function | Purpose |
|---|---|
| list_clust | Full fleet overview with cluster types, states, and versions |
| get_cluste | Detailed view of a specific cluster including all hosts and their roles (Required Cluster ID) |
| get_cluste | Normalized topology view showing nodes and replication links (Required Cluster ID) |
| list_nodes | Flattened node inventory across all clusters or one cluster (Optional Cluster ID) |
| list_alarm | Active alarms for a specific cluster (Required Cluster ID) |
| list_all_a | Active alarms across the entire fleet |
| get_contro | CMON version, RPC version, and connectivity status |
| get_cluste | Contents of one collected log file, addressed by (cluster_id, hostname, filename). Read-only; returns the tail of the last 200 lines by default, with optional max_bytes, port, and tail_lines parameters (Required Cluster ID) |
| export_clust | Writes the full contents of one collected log file to a local output_path on the MCP Server host; the file is created with mode 0600 (Required Cluster ID) |
| get_cmon_l | CMON daemon log entries for a cluster. New optional filters: severity, component, hostname, since |
| get_cmon_l | Summary counts of CMON log entries by severity and component |
| list_clust | Inventory of log files CMON has collected from cluster hosts (mysqld.log, postgresql-*.log, prometheus.log, agent_internal_logs.log, cmnd.log) |

### 1.111.2 Example Prompt

Here are some examples of using clustercontrol-mcp for Cluster Inventory and Health:

- Get information about Cluster databases

```
> List all database clusters

> Get details for cluster `cluster_id` or `cluster_name`

> Get the logs for cluster `cluster_id` or `cluster_name`

> List active alarms for cluster `cluster_id` or `cluster_name`

<br>
![ClusterControl MCP␣
```

(continues on next page)

(continued from previous page)

```
↪Response](../../../images/clustercontrol-mcp-cluster-1.png)
<br>
<br>
![ClusterControl MCP␣
↪Response](../../../images/clustercontrol-mcp-cluster-2.png)
<br>
```

## 1.112 Performance Analysis

Identify slow queries, long-running transactions, deadlocks, and other performance bottlenecks across your database fleet.

### 1.112.1 List Functions

| Function | Purpose |
|---|---|
| `get_cluster_stats` | Time-series CPU, memory, disk, network, SQL, and DB stats (Required Cluster ID) |
| `get_top_queries` | Top resource-consuming query digests (Required Cluster ID) |
| `get_sql_processes` | Live process list across all nodes (Required Cluster ID) |
| `kill_db_process` | Terminate runaway queries (Required Cluster ID) |
| `get_db_growth` | Historical database size trends for capacity planning (Required Cluster ID) |
| `check_no_primary_keys` | Find tables missing a primary key (Required Cluster ID) |
| `check_redundant_index` | Find redundant indexes (Required Cluster ID) |
| `check_myisam_tables` | Find tables using the MyISAM storage engine (Required Cluster ID) |
| `get_tx_deadlocks` | Transaction deadlock analysis with blocking SQL |

### 1.112.2 Example Prompt

Here are some examples of using clustercontrol-mcp for Performance Analysis:

- Show current processlist from the cluster.

```
> Show currently running SQL queries on cluster mariasn (mariadb)

<br>
![ClusterControl MCP␣
↪Response](../../../images/clustercontrol-mcp-job-1.png)
<br>
```

- Kill the running job

```
> Kill the query with pid #14927

<br>
![ClusterControl MCP␣
```

(continues on next page)

(continued from previous page)

```
↪Response](../../../images/clustercontrol-mcp-job-2.png)
<br>
```

## 1.113 Jobs and Operations

Monitor running and completed jobs, track task progress, and review operational history across your clusters.

### 1.113.1 List Functions

| Function | Purpose |
|---|---|
| `list_jobs` | Recent jobs with optional status and command filters (Required Cluster ID) |
| `get_job_lo` | Full log output for a specific job (Required Cluster ID and Job id) |
| `create_job` | Trigger backup, restart, failover, and other cluster operations (Required Cluster ID and command string) |
| `kill_job` | Terminate stuck jobs (Required Cluster ID) |

### 1.113.2 Example Prompt

Here are some examples of using clustercontrol-mcp for Jobs and Operations:

> List recent jobs on cluster `cluster_id` or `cluster_name`
>
> Show log for job `job_id` on cluster `cluster_id` or `cluster_name`

## 1.114 Backups and Scheduling

Manage backup jobs, review backup history, create schedules, and restore from existing backups.

### 1.114.1 List Functions

| Function | Purpose |
|---|---|
| `list_backups` | Recent backups with status and method filters (Required Cluster ID) |
| `get_latest_backup` | Most recent backup, optionally only successful ones (Required Cluster ID) |
| `list_backup_schedules` | View all configured backup schedules (Required Cluster ID) |
| `get_backup_schedule_summa` | Summary of backup schedules for a cluster (Required Cluster ID) |
| `create_backup_schedule` | Create a new recurring backup schedule (Required Cluster ID) |
| `update_backup_schedule` | Update an existing backup schedule (Required Cluster ID) |
| `delete_backup_schedule` | Delete a backup schedule (Required Cluster ID) |
| `delete_backup` | Delete a backup record (Required Cluster ID) |
| `restore_backup` | Restore from an existing backup (Required Cluster ID) |

### 1.114.2 Example Prompt

Here are some examples of using clustercontrol-mcp for Backups and Scheduling:

> When was the last successful backup on cluster `cluster_id` or `cluster_name`?
>
> Create a MariaDB backup schedule on cluster `cluster_id` or `cluster_name` every day at 4:25 AM using the mariabackup(full) backup method.

!!! Tips

```
To successfully create a backup schedule, please mention the exact backup␣
↪method name that you want to use, for example mariabackup(full) for a␣
↪full backup. If you only mention use mariabackup method, the backup␣
↪schedule job will be created but it will fail because the backup method␣
↪is not recognized by ClusterControl.
```

---

## 1.115 ProxySQL

Inspect and manage ProxySQL instances including backend servers, query routing rules, and traffic statistics.

### 1.115.1 List Functions

| Function | Purpose |
|---|---|
| `list_proxysql_top_querie` | Top queries by execution time through the proxy (Required Cluster ID) |
| `list_proxysql_rules` | Query routing rules (Required Cluster ID) |
| `list_proxysql_servers` | Backend server health and hostgroup configuration (Required Cluster ID) |

### 1.115.2 Example Prompt

Here are some examples of using clustercontrol-mcp for ProxySQL :

> List ProxySQL backend servers on cluster `cluster_id` or `cluster_name`
>
> List ProxySQL query routing rules on cluster `cluster_id` or `cluster_name`
>
> Show top queries through ProxySQL on cluster `cluster_id` or `cluster_name`

---

## 1.116 Configuration

View and modify configuration parameters at the cluster, node, and controller level.

### 1.116.1 List Functions

| Function | Purpose |
|---|---|
| `get_cluster_confi` | View cluster-level database settings (Required Cluster ID) |
| `get_node_config` | View node-level database settings (Required Cluster ID and Node) |
| `set_node_config` | Modify a node configuration parameter (with dry-run safety) (Required Cluster ID and Node) |
| `unset_node_config` | Reset a node configuration parameter to default (with dry-run safety) (Required Cluster ID and Node) |
| `get_controller_co` | View CMON controller runtime parameters (Required Cluster ID) |
| `set_controller_co` | Modify CMON controller runtime parameters (with dry-run safety) (Required Cluster ID) |

### 1.116.2 Example Prompt

Here are some examples of using clustercontrol-mcp for Configuration :

> Get configuration for node [node] on cluster `cluster_id` or `cluster_name`
>
> Set `max_connections` to `150` on node [node] in cluster `cluster_id` or `cluster_name`
>
> Reset `max_connections` to default on node [node] in cluster `cluster_id` or `cluster_name`

---

## 1.117 Maintenance, Audit, and Other Tools

Schedule maintenance windows, review audit logs, generate error reports, and perform other administrative tasks.

### 1.117.1 List Functions

| Function | Purpose |
|---|---|
| `list_maintenance` | List maintenance windows for a cluster (Required Cluster ID) |
| `list_all_maintenance` | List maintenance windows across all clusters (Required Cluster ID) |
| `add_maintenance_window` | Schedule a maintenance window for a host (Required Cluster ID) |
| `remove_maintenance_window` | Remove a maintenance window by UUID (Required Cluster ID) |
| `remove_maintenance_windows_by` | Bulk remove maintenance windows by cluster or hostname (Required Cluster ID) |
| `list_audit_log` | Who did what, when, and from where (Required Cluster ID) |
| `acknowledge_alarm` | Acknowledge a single alarm (Required Cluster ID) |
| `acknowledge_alarms_by_filter` | Bulk acknowledge alarms by cluster or filter (Required Cluster ID) |
| `list_schemas` | List all database schemas on a cluster (Required Cluster ID) |
| `create_schema` | Create a new database schema (Required Cluster ID) |
| `list_publications` | PostgreSQL logical replication publications (Required Cluster ID) |
| `list_subscriptions` | PostgreSQL logical replication subscriptions (Required Cluster ID) |
| `list_error_reports` | List generated error reports for a cluster (Required Cluster ID) |
| `generate_error_report` | Generate a new error report (Required Cluster ID) |
| `download_error_report` | Download a generated error report archive (Required Cluster ID) |
| `unpack_error_report` | Extract a downloaded error report archive (Required Cluster ID) |

### 1.117.2 Example Prompt

Here are some examples of using clustercontrol-mcp for Maintenance, Audit, and Other Tools :

> Show audit logs across all clusters
>
> List error reports for cluster `cluster_id` or `cluster_name`
>
> Generate an error report for cluster `cluster_id` or `cluster_name`

!!! Tips

```
The `download_error_report` tool will download the error report to the host␣
↪where `clustercontrol-mcp` is installed. When using the AI Assistant, you␣
↪are not restricted to English, feel free to use your local language. You␣
↪can also create your own custom prompts tailored to your specific needs,␣
↪and integrate ClusterControl MCP with any automation tools you already␣
↪have.
```

**Warning**

Attention

Please use ClusterControl MCP with AI Assistants wisely and be cautious about the information you share with them. Avoid sharing sensitive or confidential data such as API keys, or any personal information. ClusterControl MCP is designed to simplify the management of clusters maintained by ClusterControl, with a scope limited to ClusterControl-related operations only.

CHAPTER

TWO

# USER GUIDE

## 2.1 ClusterControl User Guide

The User Guide covers everyday use of ClusterControl: deploying and importing database clusters, observability, cluster and node management, backup and restore, scaling, third-party integrations, and troubleshooting.

### 2.1.1 What's in this guide

- **Overview**: Using the ClusterControl GUI, CLI, RPC API, and available addons. See *Overview*.
- **Deployment**: Creating or importing database clusters and load balancers. See *Deployment*.
- **Observability**: Monitoring cluster and host health, query performance, logs, and audit trails. See *Observability*.
- **Management**: Failover and recovery, cluster and node management, configuration, upgrades, alerts, and certificates. See *Management*.
- **Backup & Restore**: Creating, scheduling, restoring, and verifying backups, including to and from cloud storage. See *Backup & Restore*.
- **Scaling**: Adding or removing database and replica nodes, and creating replica clusters. See *Add Database Node*.
- **Integration**: Connecting ClusterControl to Kubernetes, cloud providers, LDAP, Terraform, and more. See *ClusterControl Ops-C*.
- **Troubleshooting**: Diagnosing issues with ClusterControl and database clusters, and generating error reports. See *Common Issues*.

### 2.1.2 Who should use this guide

This guide is for DBAs, DevOps engineers, and SREs using ClusterControl to deploy, monitor, and manage database clusters day to day. If you're instead responsible for operating ClusterControl itself, such as upgrades and security hardening, see the *Administration Guide*.

For step-by-step deployment walkthroughs, see the *Tutorials* section.

## 2.2 Overview

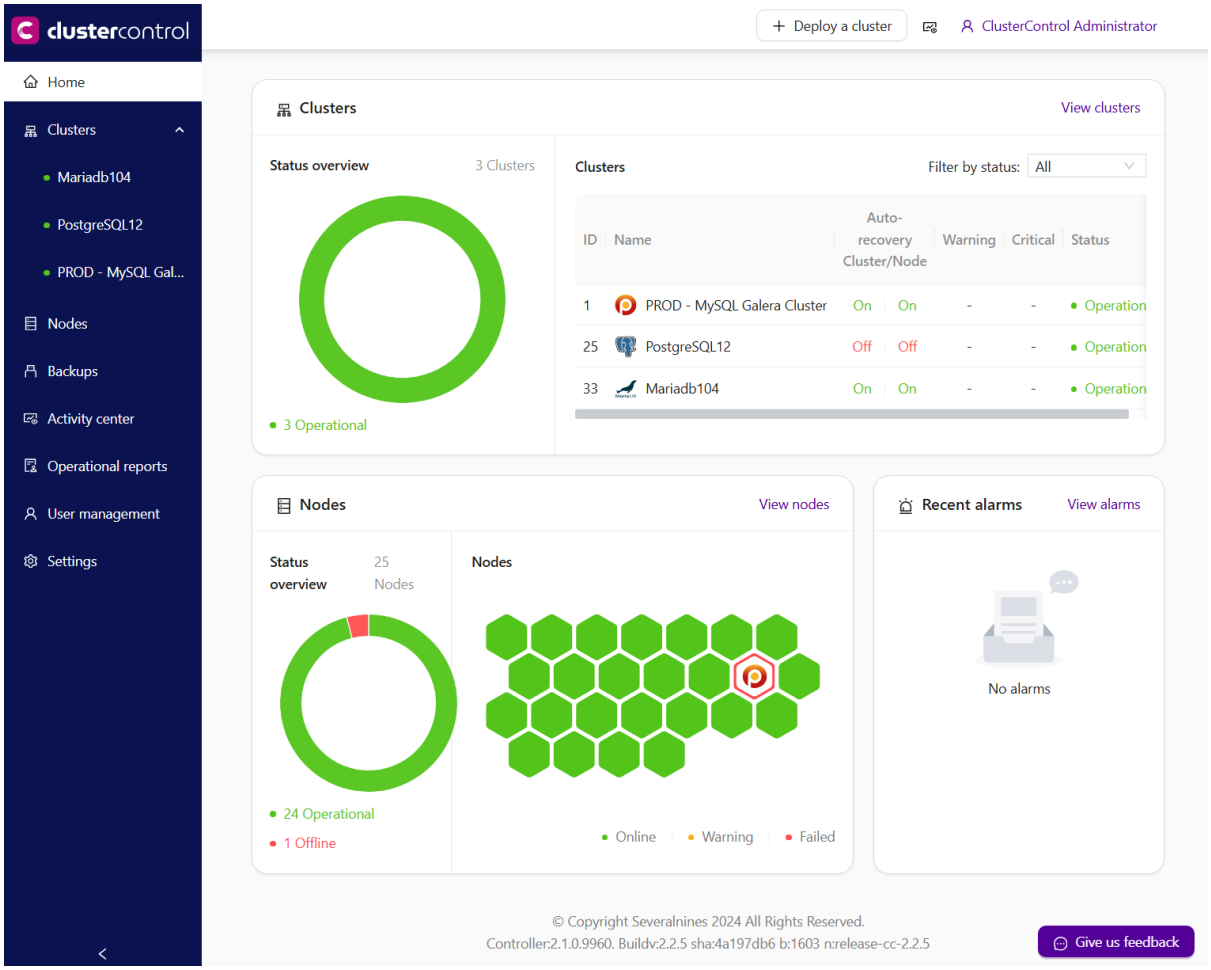
ClusterControl provides several interfaces to interact with the ClusterControl Controller service:

| Interface | Description | Package name |
|---|---|---|
| ClusterControl GUI v1 (deprecated) | The legacy graphical user interface runs as a web application, also known as ClusterControl GUI v1. This is deprecated since ClusterControl 1.9.8 and is no longer supported. | `clustercontrol` |
| ClusterControl GUI | A modern web user interface to visualize and manage the cluster. It also supports for centralized management of multiple ClusterControl Controller instances, also known as ClusterControl Operations Center (Ops-C). It interacts with the ClusterControl controller via the `cmon-proxy` service and remote procedure call (RPC). | `clustercontrol-mcc` |
| ClusterControl CLI | The command-line user interface, also known as `s9s`. | `s9s-tools` |
| ClusterControl RPC API | A programmatic remote procedure call (RPC) interface exposed by the ClusterControl Controller, used by the GUI, CLI, and external integrations to automate and integrate cluster management operations. | `clustercontrol-contro` |
| Terraform provider for ClusterControl | A Terraform provider to manage database clusters through Infrastructure as Code (IaC). | `terraform-provider-cl` |

This user guide provides steps to perform management, monitoring and automation actions via ClusterControl GUI, ClusterControl CLI and Terraform Provider for ClusterControl (where applicable).

### 2.2.1 ClusterControl GUI

The ClusterControl GUI is the graphical user interface to manage and monitor your database clusters. Installation using the installer script will include ClusterControl GUI installation as well, in a package called `clustercontrol-mcc`. It is accessible via `https://{ClusterControl_host}/` and interacts with the ClusterControl Controller (CMON) via remote procedure calls (RPC) on ports 9500 and 9501 for TLS (default).




To use ClusterControl GUI to manage your cluster, see *User Guide → Overview → ClusterControl GUI*. To learn more about ClusterControl GUI, see *Reference Manuals → Components → ClusterControl GUI*.

### 2.2.2 ClusterControl CLI

Also known as `s9s`, ClusterControl CLI is a command-line tool to interact, control, and manage database clusters using the ClusterControl Database Platform. Installation using the installer script will include ClusterControl CLI installation as well, in a package called `s9s-tools`. You can also install it on another host or workstation to manage the database cluster remotely. The commands are basically JSON messages being sent over to the ClusterControl Controller (CMON) RPC interface. Communication between the s9s (the command-line tool) and the controller process (CMON) is encrypted using TLS and requires port 9501 to be opened on the controller to be reached by the client host.

The following screenshot shows an example of ClusterControl CLI output from the terminal:

```
[root@docs-cc clustercontrol]# s9s node --list --long --cluster-id=1
STAT VERSION     CID CLUSTER                 HOST           PORT COMMENT
coC- 2.2.0.10416   1 PROD - MySQL Galera Cluster 164.90.192.155 9500 Up and running.
Po-- 2.29.2        1 PROD - MySQL Galera Cluster 164.90.192.155 9090 Process 'prometheus' is running.
yo-- 2.6.5         1 PROD - MySQL Galera Cluster 192.168.99.13  6032 Process 'proxysql' is running.
koM- 2.2           1 PROD - MySQL Galera Cluster 192.168.99.13   112 Process 'keepalived' is running.
slV- 8.0.36        1 PROD - MySQL Galera Cluster 192.168.99.15  3306 Server is unreachable (read-only).
goS- 8.0.36        1 PROD - MySQL Galera Cluster 192.168.99.3   3306 Up and running (read-write).
Ao-- 1.0.0         1 PROD - MySQL Galera Cluster 192.168.99.3   4433 Process 'cmnd' is running.
goM- 8.0.36        1 PROD - MySQL Galera Cluster 192.168.99.4   3306 Up and running (read-write).
Ao-- 1.0.0         1 PROD - MySQL Galera Cluster 192.168.99.4   4433 Process 'cmnd' is running.
goM- 8.0.36        1 PROD - MySQL Galera Cluster 192.168.99.5   3306 Up and running (read-write).
Ao-- 1.0.0         1 PROD - MySQL Galera Cluster 192.168.99.5   4433 Process 'cmnd' is running.
yo-- 2.4.4         1 PROD - MySQL Galera Cluster 192.168.99.6   6032 Process 'proxysql' is running.
koM- 2.2           1 PROD - MySQL Galera Cluster 192.168.99.6    112 Process 'keepalived' is running.
ho-- 2.4.22        1 PROD - MySQL Galera Cluster 192.168.99.7   9600 Process 'haproxy' is running.
Total: 14
```

Not all functionalities are available on both user interfaces. For instance, there is no practicality for the command-line client (ClusterControl CLI) to have all the advanced monitoring features offered by the web application client (ClusterControl GUI). The command-line client is focused on automation and adoption in management, deployment, and scaling operations while the web UI client focuses more on structural visualization with a guided approach. Occasionally, new management features will be introduced in the CLI before being incorporated into ClusterControl UI to turn it into a polished, finalized feature.

Installation using the installer script will include ClusterControl CLI installation as well on the ClusterControl server. The default "admin" user will be created automatically and is configured only for the root user.

To begin managing your clusters using the ClusterControl CLI, see *User Guide → Overview → ClusterControl CLI*. For a comprehensive list of all supported commands, options, and examples, see *Reference Manuals → Components → ClusterControl CLI*.

### 2.2.3 ClusterControl RPC API

All ClusterControl user interfaces (GUI, CLI, and Terraform provider) communicate with the ClusterControl Controller through a Remote Procedure Call (RPC) API. The RPC layer acts as a single, authoritative interface for executing management operations such as provisioning, monitoring, backups, failover, and upgrades.

The ClusterControl GUI (Ops-C) uses the `cmon-proxy` service to issue RPC calls, while the CLI and Terraform provider invoke the same underlying RPC methods programmatically. The Controller (cmon) executes these requests and applies changes to the managed database clusters. From an operational standpoint, this architecture centralizes control logic, simplifies auditing and automation, and makes it possible to integrate ClusterControl cleanly into GitOps, CI/CD, and Infrastructure-as-Code workflows.

To begin managing your clusters using the ClusterControl RPC API, see *User Guide → Overview → ClusterControl RPC API*. For a comprehensive list of all supported remote procedure calls and endpoints, see *Reference Manuals → API/SDK → ClusterControl RPC API*.

### 2.2.4 Terraform provider for ClusterControl

The Terraform Provider for Severalnines ClusterControl offers a seamless integration between the powerful infrastructure management tool, Terraform, and the robust database management capabilities of ClusterControl. This provider empowers users to effortlessly provision, configure, and manage database clusters through Infrastructure as Code (IaC), streamlining the deployment process and enhancing the efficiency of managing complex database environments.

The following is an example of the Terraform resource definition for a PostgreSQL streaming replication deployment:

```
resource "clustercontrol_db_cluster" "test-postgresql" {

    db_cluster_create = true
    db_cluster_name = "PostgreSQL Cluster"
    db_cluster_type = "pg-replication"
    db_version = "15"
    db_vendor = "postgresql"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "test-primary"
    }
```

(continues on next page)

(continued from previous page)

```
        db_host {
            hostname = "test-replica"
        }
        ssh_key_file = "/root/.ssh/id_rsa"
        ssh_user = "root"
        db_deploy_agents = true
        disable_firewall = true
        disable_selinux = true
        db_install_software = true

    }
```

This project is hosted in Github, terraform-provider-clustercontrol. To learn more about this provider, see *Reference Manuals → Tools → Terraform provider for ClusterConttrol*.

## 2.3 ClusterControl GUI

The ClusterControl GUI is the central interface for deploying, managing, and monitoring database clusters. It gives you a single place to operate MySQL, MariaDB, PostgreSQL, MongoDB, and other supported databases without switching between command-line tools or separate monitoring systems.

From the GUI, you can:

- Deploy new clusters using guided setup wizards.
- Monitor cluster health, performance, and availability in real time.
- Configure backups, upgrades, and recovery operations.
- Add database nodes and load balancers with a few clicks.
- Manage users, access control, and audit settings.
- Review alerts, logs, and operational events from one dashboard.

The ClusterControl GUI interacts with the ClusterControl controller via a proxy service called `ccmgr`, managed by a systemd service called `cmon-proxy` using remote procedure call (RPC), and integrates with other ClusterControl components, including web-based SSH, third-party notification services, cloud management platforms, and the ClusterControl CLI.

By default, the GUI operates in single-controller mode and connects to one ClusterControl controller, typically its localhost. To manage multiple controllers from the same interface, toggle on **Ops-Center** under the **Addons** tab (see *Addons*). This switches the GUI to multi-controller mode.

**Note**

Note Starting from ClusterControl 2.5.0, this toggle moved from the License page to the new Addons tab and was relabeled from "Multi-controller" to "Ops-Center". On earlier versions, look for **Enable multiple controller** under the License page instead.

### 2.3.1 Accessing the GUI

After ClusterControl is successfully installed, you may access the ClusterControl GUI at `https://<ClusterControl_IP_address_or_hostname>/` using a web browser.

If ClusterControl is installed using the *Helm chart*, you need to connect to the FQDN value defined in the `values.yaml` (or by using the `--set` parameter). For example, in the following ClusterControl installation:

```
helm install clustercontrol s9s/clustercontrol \
  --set fqdn=clustercontrol.example.com
```

After the deployment succeeds, you can access the ClusterControl GUI by going to `https://clustercontrol.example.com`, provided this FQDN is resolvable to Kubernetes load balancer EXTERNAL-IP value of `clustercontrol-ingress-nginx-controller` service. If `fqdn` flag is undefined, it defaults to `localhost`. Hence, you have to do DNS mapping of host `localhost` to resolve to the Kubernetes load balancer IP address and access via browser using `https://localhost`.

**Warning**

Attention

For Helm chart deployment, the ClusterControl GUI must be accessed via domain name, hostname or FQDN (configurable by `--set fqdn=<domain_name/fqdn>`). Accessing directly using load balancer IP address or Kubernetes `port-forward` option will result with a "page not found" error, since the GUI access is proxied by an ingress controller service and not a web server.

### 2.3.2 User registration page

For a new installation, you will be redirected to the user registration page. This is where you set up the first admin user, with a valid email address to receive alerts via email. In the next section, provide personal information and consent with the product's terms and condition and privacy policy. Click **Complete** to complete the user registration.

Once registered, you will be automatically logged in as the newly created admin user into ClusterControl.

### 2.3.3 Login page

If there is no active user session (or the session has expired), ClusterControl will default on viewing the login page. Enter a valid username and password to login.

If this is a new installation, you will be redirected to the user registration page instead. See *User registration page*.

### Forgot password

If you forget your password, you can use the ClusterControl CLI to reset the user's password. This action must be performed by any user under the ClusterControl admin group.

Connect to the ClusterControl server via SSH, and run the following command:

```
s9s user --change-password --new-password=<new_password> <username>
```

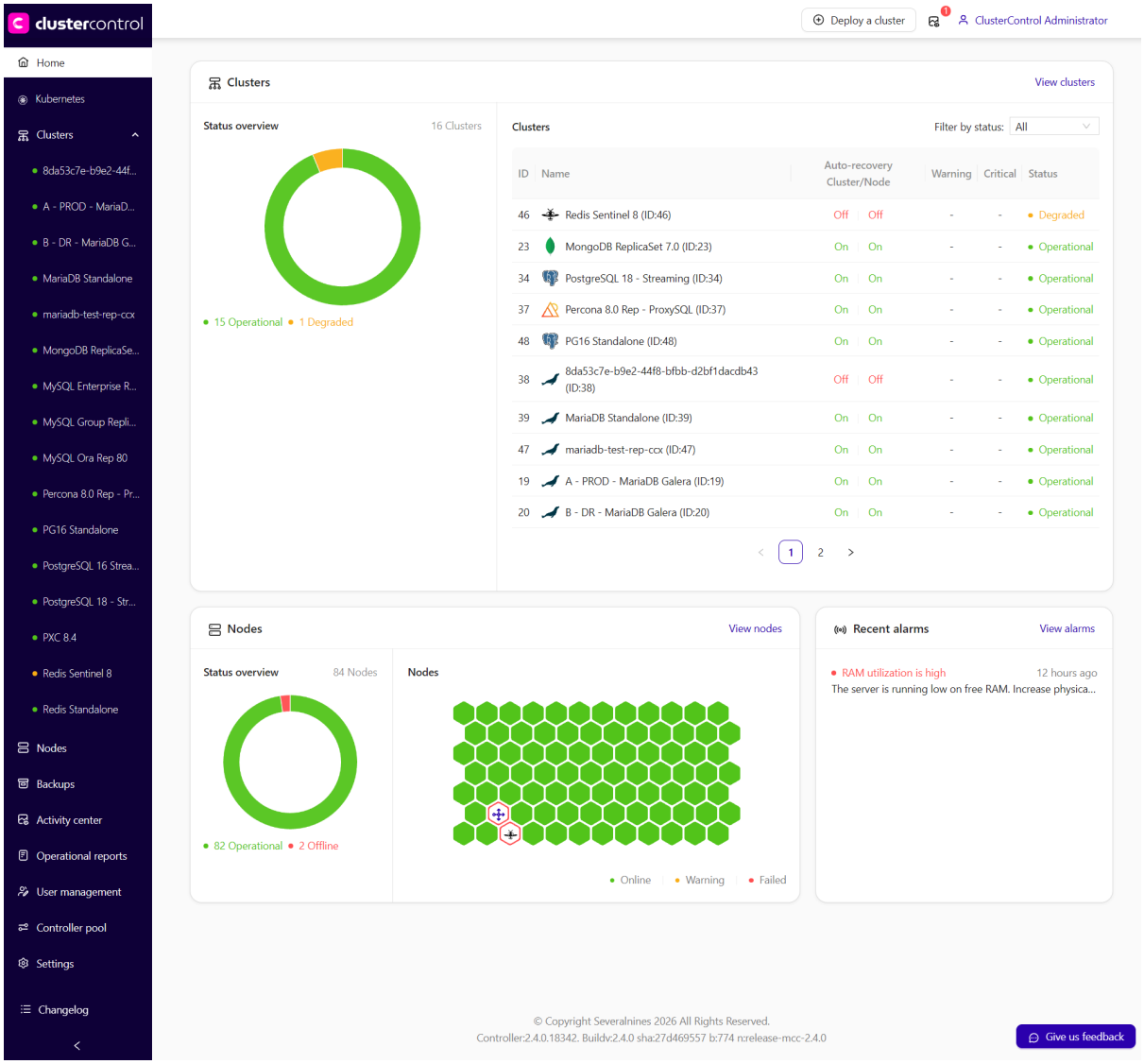
**Example output**

```
$ s9s user --change-password --new-password='K7hY24$2WeC#' dbadmin
Ok.
```

Once done, you may login again using the new password set above.

## 2.3.4 Home

This is the default start-up page after a user has successfully logged in. It presents a summary of database clusters and nodes managed by ClusterControl, bounded by the user's group permission. In the main panel, you will see 3 main sections for clusters, nodes and alarms.




In the **Home** page, here are its key sections:

## Clusters Status (top-left card)

- The donut chart summarizes the clusters' state.
- A compact table listing clusters by ID, name, and status:
  - Auto-recovery: Whether ClusterControl can automatically recover the cluster or node if a failure occurs. See *Failover and Recovery*.
  - Warnings/Critical: Number of issues requiring attention.
  - Status: Cluster status.

## Nodes Status (bottom-left card)

- The donut chart shows a summary of total nodes and how many are online or have warnings/failures.
- A honeycomb grid representing each node's current state (green for online, red for failed and yellow for warning).
- Rollover on a honeycomb cell will pop up a card displaying more information about the node.

#### Recent Alarms (right card)

- Lists active alerts or alarms.
- If everything is running smoothly, you might see "No alarms".

Each card often has a **View..** or **Details..** link, taking you to a more detailed page. For example, **View clusters**, **View nodes**, or **View alarms**.

!!! Tips

```
* Hover and Click: Many charts or icons are interactive - hover to see node␣
↪details, click to drill down.
* Filtering and Sorting: In lists (e.g., the Clusters list), you can often␣
↪filter by status or sort by name/ID.
* Always Check the Events/Alarms: They're a quick way to spot issues before␣
↪they become critical.
```

### 2.3.5 Navigation bar

The top navigation bar provides quick access to ClusterControl's common functionalities. It also displays the breadcrumbs for cluster list and notification banners. If you click sidebar's **Clusters** and click on a cluster, the breadcrumbs will appear on the navigation bar visualizing the current path you are at.

#### Subscribe Now

Clicking on the **Subscribe Now** button at the top-right corner redirects you to the Severalnines' Advanced Self-Serve page, where you can choose from the available subscription packages.

Self-serve is a licensing option offered by Severalnines for ClusterControl customers who want access to advanced features without a long-term support contract. It provides functionality similar to the Advanced license, but with a flexible monthly subscription model. Customers can renew or cancel at any time and manage their subscription directly through the self-serve portal.

This option is ideal for organizations that prefer to avoid annual commitments, have limited budgets, or want to evaluate ClusterControl's capabilities before entering a long-term agreement. Whether for production, QA, staging, or development environments, the self-serve model offers a practical way to explore and utilize ClusterControl with greater flexibility and lower commitment. For more information, visit the Self-serve page.

#### Deploy a cluster

Clicking on the **Deploy a Cluster** button at the top-right area will give users two more options, as shown in the following screenshot:

- **Create a database cluster** – Use this deployment wizard to deploy a new database cluster or server. See *Create Database Cluster*.
- **Import a database cluster** – Use this deployment wizard to import your existing database cluster or server. See *Import Database Cluster*.

#### Activity center

Next to the **Deploy a Cluster** button is an icon (commonly appeared with a small red counter) called **Activity center**. Clicking on it will expand the quick view of activity center where you can see the latest alarms, jobs and audit logs. There are also **Actions** button associated with every entry where you can perform certain actions according to the entry's type.

#### User profile

The top-right corner provides quick access to the current user's profile. It provides the timezone information, email address and the full name (in first name, last name format) with quick access to profile management, license management and the logout page.

### 2.3.6 Sidebar

The left sidebar provides all ClusterControl global menus. It is collapsible and expandable by clicking the < or > icon at the bottom.

#### Kubernetes

Clicking on **Kubernetes** will expand the list of all database clusters managed by ClusterControl which running on Kubernetes platform. See *Integration → Kubernetes*.

**Note**

Note Starting from ClusterControl 2.5.0, this feature can be disabled by going to ***ClusterControl GUI → Settings → Addons*** and toggling off **Kubernetes**. On earlier versions, use ***ClusterControl GUI → Settings → License → Disable Kubernetes*** instead.

#### Clusters

Clicking on **Clusters** will expand the list of all database clusters managed by ClusterControl. Click on one of the clusters will open the individual cluster section on the main panel.

#### Nodes

Provides global overview of all nodes for all clusters managed by ClusterControl.

You can filter the list by cluster and node's status, by clicking on the filter icon in the respective columns. You can also perform node's management tasks by clicking on the **Actions** button.

**Read more**

- *User Guide → Management → Nodes Management*
- *User Guide → Observability → Status and Healthchecks*

#### Backups

The Backups page in ClusterControl provides a global view of all backup tasks, schedules, and storage repositories in your environment. It allows you to create new backups, manage existing ones, and review completed jobs. This page is essential for ensuring data safety and for streamlining backup operations.

Here are its key sections:

- **Create backup** – Clicking this opens a wizard or form to initiate a new backup job, schedule a backup or create a new Elasticsearch snapshot repository (specific to Elasticsearch backups).
- **All Backups** – Lists all completed or in-progress backups for all clusters. You can customize the display columns by clicking on the **View** button.
- **Schedules** – Shows the schedules configured for automatic or recurring backups for all clusters.
- **Elastic Repositories** – (Specific to Elasticsearch backups) Lists all Elasticsearch snapshot repositories.

**Read more**

- *User Guide → Backup and Restore → Create Backup*
- *User Guide → Backup and Restore → Schedule Backup*

### Activity center

Provides global view of all alarms, jobs and audit logs for all clusters managed by ClusterControl.

### Operational reports

The operational reports feature provides an overview of your database cluster's performance, health, and activity over a selected time period. The reports help administrators review system status, resource usage, and operational events in a single place.

With operational reports, you can:

- Review resource usage such as CPU, memory, disk, and network utilization.
- Monitor cluster metrics including query throughput, replication lag, and system load.
- Track backup activity and identify failed or missed backups.
- View database status information such as uptime, availability, and recorded incidents.
- Generate reports for specific time periods or selected metrics.

Reports can be generated on demand or scheduled to run automatically, making it easier to monitor database operations and review cluster activity over time.

**Read more**

- *User Guide → Observability → Operational Reports*

### User management

Manage ClusterControl users, teams and authentication integration with directory services (LDAP or Active Directory).

**Read more**

- *Administration Guide → Users and Teams*

### Settings

Global settings for the ClusterControl instance.

### Profile

Manage the current user's profile and time zone settings.

### Updating user's profile

Click **Settings** to open the editing panel, where you can update your email address, first name, last name, and time zone. The selected time zone affects how time-related information is displayed in the ClusterControl GUI. Click **Save** to save the updated information.

### Change password

Click on the **Change password** and specify the current password and the new password accordingly. Click **Change** to proceed.

If you want to reset the user's password instead, login as a user belongs to "admins" group and go to ***ClusterControl GUI → User management → {user} → Actions → Change password***.

### Cloud storage credentials

The Cloud storage credentials tab provides a centralized location for managing credentials used to store and retrieve backups from supported cloud providers. By configuring valid credentials here, you can automate backup uploads to, and downloads from, various cloud platforms.

The supported object storage platforms are:

- AWS Simple Storage Service (S3)
- Azure Blob Storage
- Google Cloud Storage
- S3-compatible object storage like MinIO, DigitalOcean Spaces, Openstack Swift and SeaweedFS

Cloud storage integration relies on other ClusterControl modules called `cmon-cloud` and `clud`. These binaries are provided by `clustercontrol-cloud` and `clustercontrol-clud` packages.

**Read more**

- *User Guide → Integration → Cloud Providers*
- *User Guide → Backup and Restore → Upload Backup to Cloud*
- *User Guide → Backup and Restore → Create Backup*
- *User Guide → Backup and Restore → Schedule Backup*

### Notification services

Manage the integration with third-party notification and messaging platform on events triggered by ClusterControl. This feature allows you to integrate ClusterControl alerts into your organization's communication channels, incident response systems, workflow management systems or event-driven applications via webhook calls. You can configure the notifications to be sent out to all or specific clusters, or to all or specific alert severity like error or warning alerts.

Notification service relies on another ClusterControl module called `cmon-events`. This binary is provided by package `clustercontrol-notifications`.

**Read more**

- *User Guide → Integration → Notification Services*
- *Reference Manuals → Components → ClusterControl Notifications*

### Certificate management

Manage all keys and certificates of all clusters managed by ClusterControl. In this section, you can generate a new certificate authority (CA), generate a new self-signed certificate or import existing certificates into the certificate repository. The certificates imported or generated here can then be assigned to any cluster managed by ClusterControl.

**Read more**

- *User Guide → Management → Certificates*

### License

View and manage the details of your ClusterControl license, as well as activating enterprise and experimental features. From this section, you can check if your license is active, identify how many nodes you can manage under the current license, and update the license key as needed.

**Note**

Note Starting from ClusterControl 2.5.0, feature toggles are no longer shown on this page. See *Addons* to enable or disable Ops-Center and Kubernetes.

The following table explains the fields shown in this section:

| Field | Description |
|---|---|
| License | Displays the edition of ClusterControl (e.g., ClusterControl Enterprise). |
| Status | Indicates whether the license is currently Active or Expired. |
| License owner | Shows the organization or individual that owns the license. |
| Expiration date | The date and time when the license will expire. Once this date passes, ClusterControl may revert to a restricted mode unless renewed or updated. |
| License owner's email | Contact email for the entity that owns the license. Usually used for support or renewal discussions. |
| Nodes limit | Displays how many database server nodes are currently managed compared to the total limit available under the license (e.g., 12/unlimited nodes). See *Licensed nodes* |

**Info**

ClusterControl offers the following license types:

- **Community** – Free license offering with basic features.
- **Advanced (Self-Serve)** – Monthly subscription license managed through the self-serve portal, without a support contract.
- **Advanced** – Commercial license with support options.
- **Enterprise** – Commercial license with extended support and enterprise-focused features.

See *Key features* and Pricing for features comparison.

### Updating license key

For users with a valid subscription - namely Advanced (Self-Serve), Advanced and Enterprise - enter your license key by clicking on the **Enter a license key** button to activate features based on the subscription. The license string contains information about the license type, company/affiliation, email, expiration date, and the total number of licensed nodes.

The following example is license information that one would get from us:

```
Email: test@severalnines.com
Company: Severalnines AB
Expiration Date: 2019-04-23
Licensed Nodes: 1
License Key:␣
↪ewogICAgImNvbXBhbnkiOiAiU2V2ZXJhbG5pbmPkIEFCIiwKICAgICJlbWFpbF9hZGRyZXNzIjog
```

(continues on next page)

(continued from previous page)

```
InRlc3RAc2VRZXJhbG5pbmVzLmNvbSIsCiAgICAiZXhwaXJhdGlvbl9kYXRlIjogIjIwMTktMDQt
MjNUMDA6MDA6MDAuMDAxWiIsCiAgICAibGljZW5zZWRfbm9kZXMiOiAtMSwKICAgICJ0eXBlIjog
IkVudGVycHJpc2UiCn0Ke3M5cy1zaWduYXR1cmUtc2VwYXJhdG9yfQp0RUew5OZKV8SqmmwiQEzT
P+qTNnmCphirIVm7MriA/aCdlJYQcr1NJr4nvTNcSmgu4uFVf3Ufv4olHr4wrBq0/Js9Rm8bJWZo
BO8svHzQhCmIVEWcTYub362krjRyREnOGXaqWwUnvkZ0uUCT+WDaM1P9qn/HawoNd0e8E0+7WiZK
CpwjH+ESqSEppeu/Ewzf3p0C0e8WZbwHtZ9UmX2qJNQq9NDlByrO8FtbVjOOL4zTbc8jV0W2DWzY
1swzOgeyk+7N2eGVRWfdUSzudQSXkT3LA4cdV2HAsU5QLnmKxSCgg+jq+RQJiPwdPXEr3gzjzJzV
+qhmOZ5tTN+WABPy9l3kpztlCbkfzO84/4lM7Z3c4rQ8snMTu6RvD2M+oh/4lhvR8M9RrQQcl8JF
RX2Ak1ZAKxAXkJ97Z5U7nIzuyUGuMTCXdKGEtQkBXzpIcYFvXDeWu0MUks+EULpqG+OFnl+rSZa0
nNTSW3mR/f9B+4e2mK4y2OpJhh4rWPXR1DLpLVLk/2p0o64aEizA+IPe0TP+ox7bFzEfAXirVWfC
/Ol7m1k6arRbl8PSV1DRRcefM+UsABa6jypoiit+JXNPOajdjY1WBgEekCn/jeXBBoPM2k26274u
br0BuHULLkxGSpC8I2/nW6s84E653FO1Kpbvyx+2SKJxwUxLiuEZ2g==
```

Only paste the license key string (starting after the `License Key:` part) into the license text field. Click **Apply license key** to activate the license into runtime. No service restart required.

### Licensed nodes

The following nodes' type will be counted as licensed nodes by ClusterControl:

- MySQL/MariaDB/Galera Cluster database nodes
- PostgreSQL/TimescaleDB/EDB database nodes(1) { .annotate }
    1. EDB requires appropriate licensing from the vendor. Kindly get the database license ready prior to the deployment.
- ProxySQL nodes
- Redis database (or shard) nodes
- Valkey database (or shard) nodes
- MongoDB Community/Enterprise database (or shard) nodes(1) { .annotate }
    1. MongoDB Enterprise requires appropriate licensing from the vendor. Kindly get the database license ready prior to the deployment.
- SQL Server database nodes(1) { .annotate }
    1. SQL Server requires appropriate licensing from the vendor. Kindly get the database license ready prior to the deployment.
- Elasticsearch master-eligible and data nodes

A "node" is an instance of database server or database proxy process. For PostgreSQL, ClusterControl supports managing multiple instances per host. Each PostgreSQL instance (`host:port` pair) is considered as one node.

The following nodes' type are NOT counted as licensed nodes:

- Load balancers/Connection pooler - HAProxy, PgBouncer, MariaDB MaxScale and Keepalived nodes
- MongoDB sharded cluster - MongoDB mongos, config server and arbiter nodes
- Monitoring - Prometheus, VictoriaMetrics nodes

- Backup-related nodes - Backup verification server, Elasticsearch snapshot repository nodes, pgBackRest repository nodes, Percona MongoDB Backup shared storage nodes
- ClusterControl nodes and ClusterControl multi-controller nodes

### Best Practices and Notes

1. When the license expires, ClusterControl defaults back to the Community edition. For features comparison, refer to the ClusterControl product page. Your database cluster will remain operational after a license is expired.
2. If you want to revert an existing Enterprise edition installation to the Community edition (for example, to compare features during a trial), you can manually remove the license information from the ClusterControl host. On the ClusterControl server, run:

```
$ mysql -uroot -p cmon -e "TRUNCATE TABLE cmon.license"
$ systemctl restart cmon
```

   !!! Danger “Warning”

```
 Once a trial license is truncated and the `cmon` service is␣
↪restarted, there is no way to re-activate the same trial license␣
↪anymore.
```

3. ClusterControl will raise an alarm if a license is about to expire within 30 days (as well as a reminder banner when you logged in ClusterControl). Configure the mail server accordingly to receive the alarm via email. Ensure you renew your license before expiration to avoid interruptions in production management.
4. If you exceed your node limit, you may not be able to add additional nodes. Contact your license provider or Severalnines account manager to upgrade if you anticipate growth beyond the current limit. Severalnines also offers volume license and custom licensing options depending on the use cases and requirements.
5. If you experience any issues related to licensing, you may contact Severalnines support directly and provide the license’s owner email address.
6. After updating your license key, consider refreshing your browser to confirm that all license-related features are enabled and loaded correctly.

### Addons

Turn optional ClusterControl features on or off from a single place. Starting from ClusterControl 2.5.0, the feature toggles that used to live on the License page, **Ops-Center** and **Kubernetes**, moved here, and the License page now shows license and subscription details only.

This tab, along with every other Settings tab besides Profile, is only visible to users on a team with the **Change controller configuration** permission.

**Read more**

- *User Guide → Overview → Addons*
- *Administration Guide → Users and Teams*

#### Diagnostics

Toggle on **UI audit data collection** to capture user interface activity and related diagnostic data while reproducing an issue in the ClusterControl UI. This feature is intended for UI troubleshooting only. For issues related to the ClusterControl controller or managed database clusters, use the *Error Report* feature instead.

After reproducing the issue, download the collected audit data (in .json format) to share it with our support team for further analysis. Disable the setting afterward to stop data collection.

**Read more**

- *User Guide → Troubleshooting → Error Report*

#### Changelog

Open the ClusterControl Release Notes page in the Severalnines Documentation Center. It provides release highlights, version-specific changes, release notes, and recent updates.

### 2.3.7 Footer

The footer displays information about the components' version that you are running on. For example:

```
© Copyright Severalnines 2026 All Rights Reserved.
Controller:2.4.0.18342. Buildv:2.4.0 sha:27d469557 b:774 n:release-mcc-2.4.0
```

The above means in this particular setup, the running components are:

- ClusterControl Controller v2.4.0 and build number is 18342
- ClusterControl GUI (with multi-controller support) v2.4.0, commit hash is 27d469557 and build number is 774

### 2.3.8 Give Us Feedback

This form provides quick access to let us know your feedbacks of the product. If submitted, the form will be sent over via a HTTP call back to our feedback collection system, where it will reach the ClusterControl product manager inbox anonymously.

It is a one-way submission system and no futher information or follow-up is required. The submission requires an Internet connection and without it, the submission will fail.

## 2.4 ClusterControl CLI

The ClusterControl CLI is a command-line tool to interact, control, and manage database clusters using the ClusterControl Database Platform. By default, the installer script will automatically install and configure this package on the ClusterControl node. You can also install it on another computer or workstation to manage the database cluster remotely. This command-line project is open source and publicly available at GitHub.

ClusterControl CLI opens a new door for cluster automation where you can easily integrate it with existing deployment automation tools like Ansible, Puppet, Chef, or Salt. Not all functionalities

### 2.4.1 Overview

The command-line tool is invoked by executing a binary called `s9s`. The commands are basically JSON messages being sent over to the ClusterControl Controller (CMON) RPC interface. Communication between the ClusterControl CLI (`s9s`) and the `cmon` process (ClusterControl Controller) is encrypted using TLS and requires port 9501 to be accessible on the controller node.

### 2.4.2 Installation

The ClusterControl CLI is automatically installed and configured if ClusterControl was installed via one of the methods described under *Online Installation*. You usually want to install the ClusterControl CLI manually if you want to use the CLI outside of the ClusterControl host, or if you installed ClusterControl using the manual or offline installation method.

#### Installer script

We have built an installer script for `s9s-tools` package available at `http://repo.severalnines.com/s9s-tools/install-s9s-tools.sh`. On the ClusterControl host (or any client host):

```
$ wget http://repo.severalnines.com/s9s-tools/install-s9s-tools.sh
$ chmod 755 install-s9s-tools.sh
$ sudo ./install-s9s-tools.sh
```

If you would like to install manually, see *Package manager* or *Compile from source*.

#### Package manager

The package list is available on the s9s-tools repository page.

#### Red Hat/CentOS/Rocky Linux/AlmaLinux

The repository definition file for each distribution can be downloaded directly from:

- RHEL 7: http://repo.severalnines.com/s9s-tools/RHEL_7/s9s-tools.repo
- RHEL 8: http://repo.severalnines.com/s9s-tools/RHEL_8/s9s-tools.repo
- RHEL 9: http://repo.severalnines.com/s9s-tools/RHEL_9/s9s-tools.repo

Installation steps are straight-forward:

```
# Rocky Linux 9
wget http://repo.severalnines.com/s9s-tools/RHEL_9/s9s-tools.repo -P /
↪etc/yum.repos.d
dnf install s9s-tools
```

#### Debian/Ubuntu

To install, simply do:

```
# Ubuntu 24.04
wget -qO - http://repo.severalnines.com/s9s-tools/$(lsb_release -sc)/
↪Release.key | sudo apt-key add -
echo "deb http://repo.severalnines.com/s9s-tools/$(lsb_release -sc)/ ./" |␣
↪sudo tee /etc/apt/sources.list.d/s9s-tools.list
```

(continues on next page)

(continued from previous page)

```
sudo apt-get update
sudo apt-get install s9s-tools
```

#### Compile from source

It is possible to build the ClusterControl CLI on Linux and Mac OS/X. To build from source, you may require additional packages and tools to be installed:

1. Get the source code from Github:

```
git clone https://github.com/severalnines/s9s-tools.git
```

2. Navigate to the source code directory:

```
cd s9s-tools
```

3. Install development packages such as C/C++ compiler, `autotools` and `openssl-devel`:

#### Red Hat/CentOS/Rocky Linux/AlmaLinux

```
yum groupinstall "Development Tools"
yum install automake git openssl-devel
```

#### Ubuntu/Debian

```
sudo apt-get install build-essential automake git libssl-dev byacc flex␣
↪bison
```

4. Compile the source code:

```
./autogen.sh
./configure
make
make install
```

5. ClusterControl CLI should be installed. Verify by running the following command:

```
s9s --version
```

### 2.4.3 Configuring CLI user

The first thing that must be done is to create a user that is allowed to connect to and use the controller. Communication between the `s9s` command-line client and the controller (the `cmon` process) is encrypted using TLS on port 9501. Public and private RSA key pairs associated with a username are used to encrypt the communication. The `s9s` command-line client is responsible to set up the user and the required private and public keys.

### Local access

During a ClusterControl installation, a user called "admin" is automatically created and configured inside `/etc/s9s.conf` (that is the reason "admin" username is reserved) for ClusterControl CLI usage. To verify this, invoke the `--whoami` flag in the `user` command:

```
s9s user --whoami
```

**Example output**

```
$ s9s user --whoami
admin
```

If you don't want to use the default `admin` user for ClusterControl CLI, you can create another user as below:

```
s9s user --create \
  --generate-key \
  --controller="https://localhost:9501" \
  --group=admins <username>
```

**Example output**

Create a user called "dba" to access CLI on the same host as the ClusterControl server:

```
$ s9s user --create \
  --generate-key \
  --controller="https://localhost:9501" \
  --group=admins dba
Grant user 'dba' succeeded.
```

**Warning**

Attention

Group "admins" has the right to perform all operations on the managed clusters. For a fine-grained access level, please create a different group and assign it with proper ACLs. See s9s-tree.

You will notice the following files will be generated under `~/.s9s/` directory:

```
-rw-------   1 root root 1675 Jan 31 13:29 dba.key
-rw-------   1 root root  426 Jan 31 13:29 dba.pub
-rw-r--r--.  1 root root  286 Jan 31 13:29 s9s.conf
```

Commonly, after running the `--whoami` command, you will become the new user "dba" because `~/.s9s/s9s.conf` (if exists) will take precendence over the global `/etc/s9s.conf`:

```
$ s9s user --whoami
dba
```

To force ClusterControl CLI to use the "admin" user (or to use other user's configuration file located elsewhere), you can prefix the command with `S9S_USER_CONFIG` environment variable:

```
$ S9S_USER_CONFIG=/etc/s9s.conf s9s user --whoami
admin
```

### Remote access

The steps to set up the ClusterControl CLI for remote access is similar as for *local access*, except:

- The `s9s` command-line client must exist on the remote server.
- The ClusterControl controller (cmon) must be accepting TLS connections from the remote server.
- The remote server can connect to the controller with key-based authentication (no password). This is required only during the user creation of a private/public key setup for encryption purposes.

The following shows how to prepare ClusterControl to accept remote user access and for a remote user called `remote_dba`.

1. Setup the bind address for ClusterControl controller (cmon) process as follow:

   ```
   vi /etc/default/cmon
   ```

   Add the line:

   ```
   RPC_BIND_ADDRESSES="127.0.0.1,192.168.99.12"
   ```

   Here we assume the public IP address of the ClusterControl host (controller) is 192.168.99.12.

   **Warning**

   Attention

   ```
   Ideally, you should lock down this IP address with firewall rules,␣
   ↪only allowing access from the remote servers you wish to access␣
   ↪the controller from.
   ```

2. Restart the controller and check the log:

   ```
   systemctl restart cmon
   ```

3. Verify the ClusterControl Controller is listening to the configured IP address on port 9501:

   ```
   $ cat /var/log/cmon.log | grep -i tls
   2016-11-29 12:34:04 : (INFO) Server started at tls://127.0.0.1:9501
   2016-11-29 12:34:04 : (INFO) Server started at tls://192.168.99.12:9501
   ```

4. On the remote server/computer, we have to enable key-based authentication and create a user called `remote_dba`:

   ```
   useradd -m remote_dba
   ```

5. As the current user (root or sudoer for example) in the remote server, set up a passwordless SSH to the ClusterControl host. Generate one SSH key if you don't have one:

```
ssh-keygen -t rsa   # press 'Enter' for all prompts
```

6. Copy the SSH public key to the ClusterControl controller host, for example, 192.168.99.12:

```
ssh-copy-id root@192.168.99.12
```

7. Create the ClusterControl CLI client user:

```
s9s user --generate-key --create --group=admins --controller=
↪"https://192.168.99.12:9501" remote_dba
```

**Example output**

```
$ s9s user --generate-key --create --group=admins --controller=
↪"https://192.168.99.12:9501" remote_dba
Warning: Permanently added '192.168.99.12' (ECDSA) to the list of known␣
↪hosts.
Connection to 192.168.99.12 closed.
Grant user 'remote_dba' succeeded.
```

8. Ensure the config file located at `~/.s9s/s9s.conf` looks like this (take note the IP address of the controller may be different):

```
[global]
cmon_user = remote_dba
controller_host_name = 192.168.99.12
controller_port = 9501
rpc_tls = true
```

9. Finally, test the connection:

```
s9s cluster --cluster-id=0 --ping  # you should see 'Ping Ok ..'
s9s cluster --list  # you should see a list of cluster name
```

The configuration for ClusterControl CLI remote user is complete.

### 2.4.4 Using the CLI

In most sections of the user guide, there are many examples on how to perform actions using ClusterControl CLI (alongside ClusterControl GUI and Terraform Provider for ClusterControl). Find the **ClusterControl CLI** tab to understand how the commands are constructred and manipulated by looking at the examples.

**Example**

The following is an example tabbed content in the user guide section which highlights the CLI examples:

### ClusterControl GUI

**.. Some step-by-step guidance ..**

### ClusterControl CLI

**.. Some example commands using ClusterControl CLI ..** (1) { .annotate }

1. Look at this tab for ClusterControl CLI examples.

### Terraform Provider for ClusterControl

**.. Some example definitions using Terraform HCL ..**

For full list of all commands and options, see *ClusterControl CLI Reference Manuals*.

### Getting started

To check the CLI version, use the `-V` or `--version` flag:

```
s9s --version
```

**Example output**

```
$ s9s --version
      ___              _                 _
 ___ / _ \ ___       | |_ ___    ___ | |___
/ __| (_) / __|_____| __/ _ \ / _ \| / __|
\__ \\__, \__ \_____| || (_) | (_) | \__ \
|___/  /_/|___/       \__\___/ \___/|_|___/

s9s version 1.9.2024112020 (Sweden)
BUILD (1.9.2024112020-release) 2024-11-21 18:34:39+00:00
Copyright (C) 2016-2022 Severalnines AB
```

To list out all available commands and options, use the `--help` flag:

```
s9s --help
```

**Example output**

```
$ s9s --help
Usage:
s9s COMMAND [OPTION...]

Where COMMAND is:
```

```
    account - to manage accounts on clusters.
      alarm - to manage alarms.
     backup - to view, create and restore database backups.
    cluster - to list and manipulate clusters.
 controller - to manage Cmon controllers.
        job - to view jobs.
maintenance - to view and manipulate maintenance periods.
   metatype - to print metatype information.
       node - to handle nodes.
    process - to view processes running on nodes.
replication - to monitor and control data replication.
   dbschema - to view database schemas.
     report - to manage reports.
     script - to manage and execute scripts.
     server - to manage hardware resources.
      sheet - to manage spreadsheets.
       user - to manage users.

Generic options:
    -c, --controller=URL       The URL where the controller is found.
    --config-file=PATH         Specify the configuration file for the␣
↪program.
    --help                     Show help message and exit.
    -P, --controller-port INT  The port of the controller.
    -p, --password=PASSWORD    The password for the Cmon user.
    --private-key-file=FILE    The name of the file for authentication.
    --rpc-tls                  Use TLS encryption to controller.
    -u, --cmon-user=USERNAME   The username on the Cmon system.
    -v, --verbose              Print more messages than normally.
    -V, --version              Print version information and exit.

Formatting:
    --batch                    No colors, no human readable, pure data.
    --color=always|auto|never  Sets if colors should be used in the␣
↪output.
    --date-format=FORMAT       The format of the dates printed.
    -l, --long                 Print the detailed list.
    --log-file=PATH            The path where the s9s client puts its␣
↪logs.
    --no-header                Do not print headers.
    --only-ascii               Do not use UTF8 characters.
    --print-json               Print the sent/received JSon messages.
    --print-request            Print the sent JSon request message.

Job related options:
    --job-tags=LIST            Set job tags when creating a new job.
    --log                      Wait and monitor job messages.
    --recurrence=CRONTABSTRING Timing information for recurring jobs.
    --schedule=DATE&TIME       Run the job at the specified time.
    --timeout=SECONDS          Timeout value for the entire job.
```

```
    --wait                      Wait until the job ends.
```

You can also view the user manual by using the `man` command:

```
man s9s
```

For every command, there are help information available with the `--help` flag:

```
s9s cluster --help
s9s node --help
s9s backup --help
s9s user --help
```

Alternatively, run the user manual command and specify the `s9s-<command>` as the chapter name, for example:

```
man s9s-cluster
man s9s-node
man s9s-backup
```

After a command, specify the arguments (with its respective value) or flags. In the following example, we are listing out all nodes belong to cluster ID 36 (Percona XtraDB Cluster 8.0) in a detailed format (`--long`):

```
s9s node --cluster-id=36 --list --long # (1)!
```

1. Get the cluster ID from ClusterControl GUI or ClusterControl CLI using `s9s cluster --list --long`.

**Tip**

Use the `--cluster-name` argument to identify a cluster with string, instead of `--cluster-id` with integer. For example:

```
s9s node --cluster-name="PROD - PXC 8.0" --list --long
```

**Example output**

```
$ s9s node --cluster-id=36 --list --long
STAT VERSION     CID CLUSTER        HOST            PORT COMMENT
coC- 2.3.0.11678  36 PROD - PXC 8.0 192.168.99.2    9500 Up and running.
Po-- 2.29.2       36 PROD - PXC 8.0 192.168.99.2    9090 Process
 ↪'prometheus' is running.
goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.3    3306 Up and running␣
 ↪(read-write).
Ao-- -            36 PROD - PXC 8.0 192.168.99.3    4433 Process 'cmnd' is␣
 ↪running.
goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.4    3306 Up and running␣
 ↪(read-write).
```

```
Ao-- -            36 PROD - PXC 8.0 192.168.99.4   4433 Process 'cmnd' is␣
↪running.
goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.5   3306 Up and running␣
↪(read-write).
Ao-- 1.0.0        36 PROD - PXC 8.0 192.168.99.5   4433 Process 'cmnd' is␣
↪running.
yo-- 2.6.5        36 PROD - PXC 8.0 192.168.99.6   6032 Process 'proxysql
↪' is running.
koM- 2.2          36 PROD - PXC 8.0 192.168.99.6    112 Process
↪'keepalived' is running.
yl-- 2.6.5        36 PROD - PXC 8.0 192.168.99.7   6032 Processes with␣
↪name 'proxysql' has ended.
k?-- 2.2          36 PROD - PXC 8.0 192.168.99.7    112 Process
↪'keepalived' is running.
```

By default, ClusterControl CLI will authenticate as the user configured inside `~/.s9s/s9s.conf`. If it does not exist, it will use the global configuration located at `/etc/s9s.conf`. To bypass these configuration files, specify the user's configuration as arguments:

```
# using private key to authenticate
s9s node --cluster-id=36 --list --long \
  --cmon-user=dba \
  --private-key-file=~/.s9s/dba.key \
  --controller=https://localhost:9501 \
  --rpc-tls

# using password to authenticate
s9s node --cluster-id=36 --list --long \
  --cmon-user=dba \
  --password='mysUp3rRsecr3tP455' \
  --controller=https://localhost:9501 \
  --rpc-tls
```

To let ClusterControl CLI load another user configuration file, use the `--config-file` option or `S9S_USER_CONFIG` environment variables:

```
# using --config-file
s9s node --cluster-id=36 --list --long --config-file=/
↪home/dba/my-custom-s9s.conf

# using prefixed environment variable
S9S_USER_CONFIG=/home/dba/my-custom-s9s.conf s9s node --cluster-id=36 --
↪list --long

# using exported environment variable
export S9S_USER_CONFIG=/home/dba/my-custom-s9s.conf
s9s node --cluster-id=36 --list --long
```

### Initial check

There are multiple ways to check whether ClusterControl CLI is able to connect to the controller service correctly, as show below:

- Ping cluster ID 0 (a special cluster ID):

```
s9s cluster --cluster-id=0 --ping  # make sure you get 'PING Ok ...'
```

- Get the current active user:

```
s9s user --whoami   # make sure you see your own username
```

- Get the command's exit-code:

```
s9s user --whoami -v | grep exit   # 0 means OK. Not 0 means failed or␣
↪error.
```

### Output and manipulation

For more verbosity on the request being sent to the controller, append `-v` flag to any command:

```
s9s node --cluster-id=36 --list --long -v
```

**Example output**

```
    $ s9s node --cluster-id=36 --list --long -v
    Command line options processed.
    Preparing to send request.
    URI is '/v2/auth'

    +++ Connecting to localhost:9501...
    Connected.
    Initiate TLS...
    TLS handshake finished (version: TLSv1.3, cipher:␣
↪TLS_AES_256_GCM_SHA384).
    Sending:
    POST /v2/auth HTTP/1.0
    Host: localhost:9501
    User-Agent: s9s-tools/1.0
    Connection: close
    Accept: application/json
    Transfer-Encoding: identity
    Content-Type: application/json
    Content-Length: 201

    {
        "operation": "authenticateWithPassword",
        "password": "xxxxxxxxxxxxxxxxxxx",
        "request_created": "2025-01-31T13:06:21.100Z",
        "request_id": 1,
        "user_name": "admin"
```

```
    }

    Controller version: 2.3.0.11678
    Preparing to send request.
    URI is '/v2/clusters/'

    +++ Connecting to localhost:9501...
    Connected.
    Initiate TLS...
    TLS handshake finished (version: TLSv1.3, cipher:␣
↪TLS_AES_256_GCM_SHA384).
    Sending:
    POST /v2/clusters/ HTTP/1.0
    Host: localhost:9501
    User-Agent: s9s-tools/1.0
    Connection: close
    Accept: application/json
    Transfer-Encoding: identity
    Cookie: cmon-sid=8fe6911b1dc84ceca65c90b00fe005b8
    Content-Type: application/json
    Content-Length: 155

    {
        "cluster_id": 36,
        "operation": "getClusterInfo",
        "request_created": "2025-01-31T13:06:21.118Z",
        "request_id": 2,
        "with_hosts": true
    }

    STAT VERSION     CID CLUSTER        HOST           PORT COMMENT
    coC- 2.3.0.11678  36 PROD - PXC 8.0 192.168.99.2   9500 Up and␣
↪running.
    Po-- 2.29.2       36 PROD - PXC 8.0 192.168.99.2   9090 Process
↪'prometheus' is running.
    goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.3   3306 Up and␣
↪running (read-write).
    Ao-- -            36 PROD - PXC 8.0 192.168.99.3   4433 Process 'cmnd
↪' is running.
    goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.4   3306 Up and␣
↪running (read-write).
    Ao-- -            36 PROD - PXC 8.0 192.168.99.4   4433 Process 'cmnd
↪' is running.
    goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.5   3306 Up and␣
↪running (read-write).
    Ao-- 1.0.0        36 PROD - PXC 8.0 192.168.99.5   4433 Process 'cmnd
↪' is running.
    yo-- 2.6.5        36 PROD - PXC 8.0 192.168.99.6   6032 Process
↪'proxysql' is running.
    koM- 2.2          36 PROD - PXC 8.0 192.168.99.6    112 Process
```

```
↪'keepalived' is running.
    yl-- 2.6.5        36 PROD - PXC 8.0 192.168.99.7   6032 Processes␣
↪with name 'proxysql' has ended.
    k?-- 2.2          36 PROD - PXC 8.0 192.168.99.7    112 Process
↪'keepalived' is running.
    Total: 12
    Exiting with exitcode 0.
```

Use the `--print-json` flag to get an extended output in JSON format. For example:

```
s9s node --cluster-name="PROD - PXC 8.0" --list --long --print-json
```

If a command triggers a job, you can choose to wait for the job to finish and a progress bar will be displayed (otherwise, a job will be registered and running in background):

```
s9s node --cluster-id=36 --nodes=192.168.99.4 --restart --wait
```

**Example output**

```
$ s9s node --cluster-id=36 --nodes=192.168.99.4 --restart --wait
Restarting Node
/ Job 1630 RUNNING    [████       ]  40% Restarting node
```

To see and follow the triggered job messages, specify `--log` (it will display similar output as ClusterControl GUI's job messages):

```
s9s node --cluster-id=36 --nodes=192.168.99.4 --restart --log
```

**Example output**

```
$ s9s node --cluster-id=36 --nodes=192.168.99.4 --restart --log
CMON version 2.3.0.11678.
Using SSH credentials from cluster.
Cluster ID is 36.
The creds name is 'ssh_cred_cluster_36_6245'.
The username is 'root'.
The keyfile is '/root/.ssh/id_rsa'.
Preparing to restart host.
192.168.99.4: Checking ssh/sudo with credentials ssh_cred_cluster_36_6245.
Saving cluster_autorecovery: true, node_autorecovery: true settings.
Setting cluster_autorecovery: false, node_autorecovery: false settings.
192.168.99.4:3306: Stopping mysqld (timeout=600, force stop after␣
↪timeout=false).
192.168.99.4: Stopping MySQL service.
192.168.99.4:  mysql.service - Percona XtraDB Cluster
     Loaded: loaded␣
↪(]8;;file://docs-pxc-prod-02/usr/lib/systemd/system/mysql.service/usr/lib/systemd/syst
↪enabled; preset: disabled)
```

```
     Active: active (running) since Tue 2025-02-11 10:05:31 UTC; 55s ago
    Process: 92799 ExecStartPre=/usr/bin/mysql-systemd start-pre␣
 ↪(code=exited, status=0/SUCCESS)
    Process: 92826 ExecStartPre=/bin/sh -c systemctl unset-environment _
 ↪WSREP_START_POSITION (code=exited, status=0/SUCCESS)
    Process: 92827 ExecStartPre=/bin/sh -c VAR=`bash /
 ↪usr/bin/mysql-systemd galera-recovery`; [ $? -eq 0 ] && systemctl␣
 ↪set-environment _WSREP_START_POSITION=$VAR || exit 1 (code=exited,␣
 ↪status=0/SUCCESS)
    Process: 93681 ExecStartPost=/bin/sh -c systemctl unset-environment _
 ↪WSREP_START_POSITION (code=exited, status=0/SUCCESS)
    Process: 93683 ExecStartPost=/usr/bin/mysql-systemd start-post
 ↪$MAINPID (code=exited, status=0/SUCCESS)
Main PID: 92862 (mysqld)
     Status: "Server is operational"
      Tasks: 55 (limit: 10866)
     Memory: 609.5M
        CPU: 4.763s
     CGroup: /system.slice/mysql.service
             92862 /usr/sbin/mysqld --
 ↪wsrep_start_position=8fad3f53-7f17-11ef-a5d5-5f0a8f95055b:4637068

Feb 11 10:05:23 docs-pxc-prod-02 systemd[1]: Starting Percona XtraDB␣
 ↪Cluster...
Feb 11 10:05:25 docs-pxc-prod-02 systemd[1]: mysql.service: Got␣
 ↪notification message from PID 93296, but reception only permitted for␣
 ↪main PID 92862
Feb 11 10:05:31 docs-pxc-prod-02 mysql-systemd[93683]:  SUCCESS!
Feb 11 10:05:31 docs-pxc-prod-02 systemd[1]: Started Percona XtraDB␣
 ↪Cluster.
192.168.99.4: Stopping the service mysqld.
...
```

In an output with a list of entries, the left-most column (`STAT`) displays some letters representation of the object. Here is an example:

```
$ s9s node --cluster-id=36 --list --long
STAT VERSION     CID CLUSTER        HOST           PORT COMMENT
coC- 2.3.0.11678  36 PROD - PXC 8.0 192.168.99.2   9500 Up and running.
Po-- 2.29.2       36 PROD - PXC 8.0 192.168.99.2   9090 Process 'prometheus
 ↪' is running.
goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.3   3306 Up and running␣
 ↪(read-write).
Ao-- -            36 PROD - PXC 8.0 192.168.99.3   4433 Process 'cmnd' is␣
 ↪running.
goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.4   3306 Up and running␣
 ↪(read-write).
Ao-- -            36 PROD - PXC 8.0 192.168.99.4   4433 Process 'cmnd' is␣
 ↪running.
```

(continues on next page)

(continued from previous page)

```
goM- 8.0.37       36 PROD - PXC 8.0 192.168.99.5   3306 Up and running␣
↪(read-write).
Ao-- 1.0.0        36 PROD - PXC 8.0 192.168.99.5   4433 Process 'cmnd' is␣
↪running.
yo-- 2.6.5        36 PROD - PXC 8.0 192.168.99.6   6032 Process 'proxysql'␣
↪is running.
koM- 2.2          36 PROD - PXC 8.0 192.168.99.6    112 Process 'keepalived
↪' is running.
yl-- 2.6.5        36 PROD - PXC 8.0 192.168.99.7   6032 Processes with name
↪'proxysql' has ended.
k?-- 2.2          36 PROD - PXC 8.0 192.168.99.7    112 Process 'keepalived
↪' is running.
Total: 12
```

This information can be demystified by looking at the user manual page of the `node` command, under the `STAT` section:

```
$ man s9s-node | grep -A21 -- 'STAT   '
            STAT   Some  status information represented as individual␣
↪letter. This field contains the following char-
                   acters:

                nodetype
                       This is the type of the node. It can be c for␣
↪controller, g for Galera node, x  for  MaxScale
                       node,  k for Keepalived node, p for PostgreSQL, m␣
↪for Mongo, e for MemCached, y for ProxySql,
                       h for HaProxy, b for PgBouncer, B for PgBackRest, t␣
↪for PBMAgent, a for Garbd,  r  for  group
                       replication  host, A for cmon agent, P for␣
↪Prometheus, s for generic MySQL nodes, S for Redis
                       sentinel, R for Redis, E for Elasticsearch, and ?␣
↪for unknown nodes.

                hoststatus
                       The status of the node. It can be o for on-line, l␣
↪for off-line, f for failed  nodes,  r  for
                       nodes performing recovery, - for nodest that are␣
↪shut down and ? for nodes in unknown state.

                role   This  field shows the role of the node in the␣
↪cluster. This can be M for master, S for Slave,
                       U for multi (master and slave), C for controller, V␣
↪for backup verification node, A  for  ar-
                       biter, R for backup repository host D for␣
↪Elasticsearch data host c for Elasticsearch coordi-
                       nator_only host and - for everything else.

                maintenance
                       This field shows if the node is in maintenance␣
```

(continues on next page)

(continued from previous page)

```
↪mode. The character is M for nodes in  maintenance mode and - for nodes that are not in␣
↪maintenance mode.
```

By default, ClusterControl CLI displays a human-friendly output with colouring, header and some statistics. You can use the `--batch` flag to return a plain data, which is useful for manipulation by other commands or automation script:

```
s9s node --cluster-name="PROD - PXC 8.0" --list --long --batch
```

**Example output**

With `--batch`, it prints data without syntax highlight, header and totals. Only pure table to be processed using filters:

```
$ s9s node --cluster-name="PROD - PXC 8.0" --list --long --batch
coC- 2.3.0.11678 36 PROD - PXC 8.0 192.168.99.2   9500 Up and running.
Po-- 2.29.2      36 PROD - PXC 8.0 192.168.99.2   9090 Process 'prometheus
↪' is running.
goM- 8.0.37      36 PROD - PXC 8.0 192.168.99.3   3306 Up and running␣
↪(read-write).
Ao-- -           36 PROD - PXC 8.0 192.168.99.3   4433 Process 'cmnd' is␣
↪running.
goM- 8.0.37      36 PROD - PXC 8.0 192.168.99.4   3306 Up and running␣
↪(read-write).
Ao-- -           36 PROD - PXC 8.0 192.168.99.4   4433 Process 'cmnd' is␣
↪running.
goM- 8.0.37      36 PROD - PXC 8.0 192.168.99.5   3306 Up and running␣
↪(read-write).
Ao-- 1.0.0       36 PROD - PXC 8.0 192.168.99.5   4433 Process 'cmnd' is␣
↪running.
yo-- 2.6.5       36 PROD - PXC 8.0 192.168.99.6   6032 Process 'proxysql'␣
↪is running.
koM- 2.2         36 PROD - PXC 8.0 192.168.99.6    112 Process 'keepalived
↪' is running.
yl-- 2.6.5       36 PROD - PXC 8.0 192.168.99.7   6032 Processes with␣
↪name 'proxysql' has ended.
k?-- 2.2         36 PROD - PXC 8.0 192.168.99.7    112 Process 'keepalived
↪' is running.
```

To redirect the output to another command, it is recommended to do it in batch mode:

```
$ s9s node --cluster-name="PROD - PXC 8.0" --list --long --batch | grep ^
↪coC | awk {'print $9'}
9500
```

With `--print-json`, we can redirect the output to a JSON processor like `jq` and perform JSON object filtering:

```
$ s9s user --whoami --print-json | jq '.user.groups[].group_name'
"admins"
```

Combine with `diff`, we can compare the current database configurations (from the configuration file) side-by-side between two PostgreSQL database nodes, 192.168.99.7 and 192.168.99.8:

```
diff -y <(s9s node --list-config --nodes=192.168.99.7:5432) <(s9s node --
↪list-config --nodes=192.168.99.8:5432)
```

ClusterControl provides an *RPC API interface* for the ClusterControl controller (CMON).

Please note that the RPCv2 API calls are only available via secure (TLS) connections, where encryption is mandatory. The RPCv2 also uses an authentication call, and very few of the calls are available to be used without authenticating first.

By default, the encrypted communication can be used through port 9501 and the RPC v2 requests are server served on the `/v2` path (e.g. `/v2/clusters`).

**Requirements**

- All RPCv2 API calls require a **TLS (secure) connection** — unencrypted connections are not supported.
- The API listens on **port 9501** by default.
- Authentication is required for most endpoints before any requests can be processed.

## 2.5 Accessing the RPC API

By default, the ClusterControl RPC API interface (port 9501) listens to localhost. This means, only local user is able to access the API interface and additional steps are required to expose this interface publicly. See the **Remote** section below.

### Local

1. Access the ClusterControl IP address on port 9501 as shown in the example below:

```
curl -k https://127.0.0.1:9501/v2
```

2. You should get the following reply:

```
{
    "controller_id": "c469a80f-8ad9-4e7b-8f5d-82346c671609",
    "error_string": "Invalid operation '' on '/v2/info'.",
    "request_operation": "",
    "request_path": [
        "v2"
    ],
    "request_processed": "2025-04-15T17:55:49.409Z",
    "request_status": "InvalidRequest"
}
```

Despite the error, it means that you are able to connect to the interface and get a proper reply. See *Authenticating with the API* to start sending a proper request.

### Remote

By default the RPC API interface only listens to localhost on port 9501. To expose the RPC API remotely, include the IP address of the host in the `RPC_BIND_ADDRESSES` variables inside `/etc/default/cmon`. The following commands should be executed on the ClusterControl server.

1. Add the primary IP address of the ClusterControl node inside `/etc/default/cmon`:

```
RPC_BIND_ADDRESSES="192.168.20.10,127.0.0.1"
```

2. Restart `cmon` service to make sure ClusterControl listens on port 9501 for both IP addresses defined above:

```
systemctl restart cmon
```

3. Make sure you see port 9501 is listening to the configured IP addresses.

```
$ netstat -tulpn | grep 9501
tcp        0      0 127.0.0.1:9501          0.0.0.0:*               ␣
↪LISTEN      1681928/cmon
tcp        0      0 192.168.20.10:9501      0.0.0.0:*               ␣
↪LISTEN      1681928/cmon
```

4. Access the ClusterControl IP address on port 9501 as shown in the example below:

```
curl -k https://192.168.61.210:9501/v2
```

5. You should get the following reply:

```
{
    "controller_id": "c469a80f-8ad9-4e7b-8f5d-82346c671609",
    "error_string": "Invalid operation '' on '/v2/info'.",
    "request_operation": "",
    "request_path": [
        "v2"
    ],
    "request_processed": "2025-04-15T17:55:49.409Z",
    "request_status": "InvalidRequest"
}
```

Despite the error, it means that you are able to connect to the interface and get a proper reply. See *Authenticating with the API* to start sending a proper request.

## 2.6 Authenticating with the API

If you would like to access the cluster object via HTTP, send the request to the RPCv2 API accessible on port 9501 (TLS enabled). Use one of the following two methods:

### Inline credentials

Pass the username and password inside a nested `authenticate` method in an operation:

```
curl -k 'https://192.168.20.10:9501/v2/clusters' \
-XPOST -d \
'{"operation": "getAllClusterInfo",
"authenticate":
{"password": "s3cr3tP455","username": "readeruser"}}'
```

### Session cookie (recommended)

This is the recommended way and is used by all our applications to connect to the ClusterControl controller. Users need to maintain cookies/sessions after the first authentication and send other requests with the corresponding cookies/session. To obtain a session, use the `authenticateWithPassword` operation and connect to the `v2/auth` endpoint:

```
curl -k 'https://192.168.20.10:9501/v2/auth' \
-XPOST -d \
'{"operation":"authenticateWithPassword",
"user_name":"readeruser",
"password":"s3cr3tP455"}' \
-c cookies.jar
```

Then, include cookies.jar with every subsequent read-only request:

```
curl -k 'https://192.168.20.10:9501/v2/clusters' \
-XPOST -d \
'{"operation":"getallclusterinfo"}' \
-b cookies.jar
```

## 2.7 Successful request

If the RPC call is successful, you should receive a response with `"request_status": "Ok"`, similar to the following output:

**Successful response**

```
{
    "controller_id": "683e2b4b-038f-4cf7-93a0-4c3f95b3db70",
    "request_processed": "2024-11-07T19:33:12.701Z",
    "request_status": "Ok",
    "request_user_id": 5,
    "total": 3,
    "clusters":
    [
        ...json content...
    ],
    "debug_messages":
    [
        "RPC V2 authenticated user is 'readeruser'."
```

```
    ]
}
```

## 2.8 Failed request

In this example, the user that we use is only having a read-only permission. If the user is trying to execute a non-read-only request, for example, sending a stop node job that requires an "execute" permission:

```
curl -k 'https://192.168.20.10:9501/v2/jobs' \
    -XPOST -d '{"cluster_id": 21, "job": {"class_name": "CmonJobInstance",
    "job_spec":
    {"command": "stop",
    "job_dat":{"clusterid": 21,
    "node":{"hostname": "192.168.20.53"}}}},
    "operation": "createJobInstance"}' \
    -b cookies.jar
```

The reply should return `"request_status": "AccessDenied"`, similar to the following output:

!!! failure "Access denied response"

```
```json
{
    "controller_id": "9eb8b1d1-f9d1-439e-945e-108982812bee",
    "error_string": "Execute access to cluster 21 for user readeruser is␣
↪denied.",
    "request_processed": "2021-05-04T07:01:38.342Z",
    "request_status": "AccessDenied",
    "request_user_id": 7,
    "debug_messages":
    [
        "RPC V2 authenticated user is 'readeruser'."
    ]
}
```
```

**See also**

For a comprehensive list of all supported remote procedure calls and endpoints, see *Reference Manuals → API/SDK → ClusterControl RPC API*.

## 2.9 Addons

The Addons tab is where you turn optional ClusterControl features on or off. Starting from ClusterControl 2.5.0, the feature toggles that used to sit on the License page, **Ops-Center** (previously labeled **Multi-controller**) and **Kubernetes**, moved here. The License page itself now only shows license and subscription details; see *ClusterControl GUI → License*.

**Info**

Settings, including the Addons tab, is only visible to users on a team with the **Change controller configuration** permission (see *Administration Guide → Users and Teams*). Users without this permission only see **Profile** under Settings.

---

### 2.9.1 Access the Addons tab

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI***.
2. Click **Settings** in the left sidebar.
3. Click **Addons**, next to **License** in the Settings tab bar (Profile, Cloud storage credentials, Notification services, Certificate management, License, **Addons**, Diagnostics). This opens a panel listing the available feature toggles side by side as cards, or "No addons are available" if none apply to your installation.

Each addon card shows:

- A title and a toggle switch to turn it on or off.
- A green ✓ **Enabled** tag next to the title, and a highlighted border around the card, when the addon is on.
- A short description of what the addon does.
- A **When to enable it** bullet list of scenarios where it's useful.
- A status line in the bottom-left corner (for example, `Single-controller mode · local` for Ops-Center when it's off).
- A **Learn more** link in the bottom-right corner, pointing to that addon's documentation.

---

### 2.9.2 Ops-Center

Manage several ClusterControl controllers from one console. Ops-Center gives you a single pane of glass to monitor and operate every controller, cluster and node.

Use it when:

- You run more than one ClusterControl controller, for example separate prod, staging or per-region installations, and want to manage them from one place.
- Your fleet grows beyond ~1000 nodes and you need to scale out horizontally across several CMON controllers.
- You want centralized monitoring with the ability to switch between controllers for targeted operations.

**Read more**

- *User Guide → Integration → ClusterControl Ops-C*

### Enable Ops-Center

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Addons***.
2. Toggle on **Ops-Center**. This opens the **Enable Ops-Center** wizard.
3. **Step 1, About:** review the feature summary and, optionally, follow **Learn more about how Ops-Center works** to read further. Click **Continue**.
4. **Step 2, Set root user:** enter a **Root username** and **Root user password** (email is optional). This root user is used exclusively when managing CMON Controllers, as an extra layer of security. Confirm the password, then click **Enable**.
5. The page reloads in multi-controller mode. You can now register additional controllers.

**Read more**

- *User Guide → Integration → ClusterControl Ops-C → Activating ClusterControl Ops-C*
- *User Guide → Integration → ClusterControl Ops-C → Controller Management*

### Disable Ops-Center

Disabling Ops-Center switches the console back to single-controller mode. Since more than one controller may be registered while Ops-Center is active, ClusterControl needs you to pick which one of the registered controllers stays as the single active controller.

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Addons***.
2. Toggle off **Ops-Center**. This opens the **Disable Ops-Center** dialog: "This switches the console to single-controller mode. The general overview across all controllers will no longer be available, and you'll work only with the controller you select below. The page will reload after the operation is complete. You can re-enable Ops-Center later."
3. A warning notes that **the Ops-Center root user is no longer needed and will be deleted**.
4. Under **Controller to keep**, choose the controller that should remain as the single active controller. This field is required.
5. If more than one controller is registered, an additional checkbox appears: **Remove the other controllers from the configuration**. Leave it unchecked to keep the other controllers on record (so you can re-enable Ops-Center later without re-adding them), or check it to remove them now.
6. Click **Disable**. The page reloads in single-controller mode, pointed at the controller you picked.

---

## 2.9.3 Kubernetes

Deploy and manage Kubernetes-based database clusters using database operators, alongside your traditional clusters.

Use it when:

- Your infrastructure runs on Kubernetes and you want to operate databases as native cluster resources.

- You want database operators to automate deployment, scaling and recovery on K8s.
- You need to manage Kubernetes-based database clusters from the same console as the rest of your fleet.

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Addons***.
2. Toggle **Kubernetes** on or off, then confirm in the dialog that appears. The page reloads after the operation completes.
3. When enabled, the **Kubernetes** entry appears in the left sidebar. When disabled, it is hidden.

!!! info

```
Disabling Kubernetes does not remove any existing Kubernetes clusters; it␣
↪only hides the Kubernetes options in the UI. To stop the underlying␣
↪Kubernetes services, do so manually.
```

**Read more**

- *User Guide → Integration → Kubernetes*
- *User Guide → Overview → ClusterControl GUI → Sidebar → Kubernetes*

## 2.10 Deployment

This section covers key strategies and tools for deploying your database clusters with ClusterControl. Learn how to streamline setup, ensure consistency across environments, and enable rapid scaling as your infrastructure grows.

**See also**

- *Create Database Cluster →*
- *Import Database Cluster →*
- *Create Load Balancer →*
- *Import Load Balancer →*

## 2.11 Create Database Cluster

The Create Database Cluster feature in ClusterControl allows users to easily deploy new database clusters with just a few clicks, eliminating the complexities of manual database setup. This feature provides an intuitive workflow for configuring high-availability setups, replication, load balancing, and more.

With Create Database Cluster, you can:

- Automatically deploy single or multi-node database clusters.
- Configure the topology and replication mechanisms during deployment.
- Fine-tune the database server with custom configuration template.

- Configure encryption to improve security.

This feature ensures a streamlined process, making it simple for administrators to quickly set up robust, highly available database environments that are ready for production use. Whether you are a beginner or an experienced DBA, ClusterControl's guided deployment process ensures consistency, reliability, and best practices in database management.

### 2.11.1 Support Matrix

The following database cluster types, vendors and topologies are supported:

| Database | Vendor | Topology |
|---|---|---|
| MySQL | Percona, Oracle | Standalone, replication |
| MariaDB | MariaDB | Standalone, replication |
| Galera Cluster | MariaDB, Percona | Galera certification-based replication |
| MySQL Group Replication | Oracle | Single-primary, multi-primary |
| PostgreSQL | PostgreSQL, EnterpriseDB | Standalone, streaming replication, logical replication |
| TimescaleDB | TimescaleDB | Standalone, streaming replication |
| MongoDB | MongoDB, Percona, MongoDB Enterprise | Replica set, sharded cluster |
| Redis | Redis, Valkey | Sentinel, cluster |
| Microsoft SQL Server for Linux | Microsoft | Standalone, Always On availability group |
| Elasticsearch | Elastic | Single-node cluster, high-availability cluster |
| ClickHouse | ClickHouse | Standalone, replicated cluster |

### 2.11.2 Multi-threaded Deployment

Since the release of version 2.5.0, deployments for the following clusters have multi-threaded deployments. As of this period, the clusters supported are the following:

- Galera Cluster
- PostgreSQL
- MySQL Replication clusters

This means that for these cluster types, package installations are parallelized during deployment, which can be noticeable in cluster deployment job logs.

**Example**

For example, when deploying a MySQL Replication Cluster, you shall notice the following in the deployment job logs (using ClusterControl CLI to highlight the parallelism):

```
$ s9s job --list --job-id 1111
ID   CID STATE    OWNER  GROUP  CREATED             RDY TITLE
1111   0 FINISHED admin2 admins 2026-08-01 23:44:08 100% Deploy MySQL␣
↪Galera Cluster
```

```
$ s9s job --log --job-id 1111 | grep -iE 'parallel'
...
Setting up remaining 2 node(s) in parallel.
...
All 2 parallel node setup(s) completed successfully.
...
```

### 2.11.3 Prerequisites

There are prerequisites that need to be fulfilled prior to the deployment for all database clusters:

- Make sure the target database nodes are running on a supported architecture platform and operating system. See *Hardware* and *Operating System*.
- Passwordless SSH (SSH using key-based authentication) is configured from the ClusterControl node to all database nodes. See *SSH Key-based Authentication*.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*.

### 2.11.4 Common Wizard Steps

The Create Database Cluster wizard follows the same flow and step order for every database type listed in the *Support Matrix* above. Only the **Node configuration** and **Add nodes** steps have database-specific fields, and a small number of database types insert an extra step (HBA Rules, Extensions, Snapshot storage configuration) or a **License** note. These differences are documented per database type in *Database-Specific Settings* below.

#### ClusterControl GUI

**Opening the wizard**

1. Go to ***ClusterControl GUI → Deploy a cluster → Create a database cluster*** and under the **Database** dropdown, choose your target database type (see the dropdown label listed at the start of each entry in *Database-Specific Settings*).
2. Choose a vendor from the **Vendor** dropdown. This will determine the version and package repository to use in the deployment.
3. Choose a major database version from the **Version** dropdown.
4. Click **Continue**.

**Step 1: Cluster name and tags**

You will be presented with a deployment wizard for your chosen database type. Specify the cluster name and tags. For cluster name, you may leave it blank and ClusterControl will generate a name for the cluster. For cluster tags, you can add multiple tags by pressing ++enter++ or choose from any existing tags. Click **Continue** to proceed to the next step.

**Tip**

A proper tag configuration will help you group your database clusters and is commonly recommended in a large deployment.

!!! note “PostgreSQL and PostgreSQL Logical only”

```
If you choose "PostgreSQL Enterprise DB" as the vendor, this step also asks␣
↪for the EDB repository token, which you can retrieve from␣
↪[here](https://www.enterprisedb.com/accounts/login).
```

**Step 2: SSH configuration**

Under the **SSH configuration** section, specify the SSH credentials that ClusterControl should use to connect to the database nodes:

- **SSH user**: The SSH user that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH user.
- **SSH user key path**: The SSH private key path that ClusterControl will use to perform SSH authentication to the database node. Relative path is not supported. The SSH private key must be physically secured and exist on the ClusterControl node.
- **SSH port**: The SSH port that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH port.
- **SSH sudo password**: The sudo password if the **SSH user** requires a password for privilege escalation.
- **SSH sudo / OS elevation command**: Select the OS command used to elevate user privileges for executing system-level operations. You can choose from `sudo`, `doas`, `pbrun`.
- **Install software**: Toggling on (default) instructs ClusterControl to install the database software from the package repository during deployment. Toggling off skips the database installation part. In some cases, where you have a custom database build or in an offline environment without a proper offline repository set up, you may install the database server packages manually beforehand and set this option to off. ClusterControl will then skip the installation part and proceed to the database configuration part.
- **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the database node during deployment. It is recommended to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have configured the necessary ports prior to this deployment otherwise there is a risk of failure deployment. See *Firewall and Security Groups*.
- **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recommended to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.

Click **Continue** to proceed to the next step.

**Step 3: Node configuration**

Under the **Node configuration** section, specify the database credentials and configurations that ClusterControl shall use when deploying the cluster. The exact fields shown depend on the database type; see *Database-Specific Settings* for the full list for your chosen database.

Click **Continue** to proceed to the next step.

**Step 4: Add nodes**

Under the **Add nodes** section (labeled differently for some database types, e.g. **Configuration Servers and Routers** for MongoDB Sharded Cluster), specify the target database nodes and configure the topology you want to deploy. The exact fields depend on the database type; see *Database-Specific Settings* for the full list for your chosen database.

**Note**

Note You can only proceed to the next step if all of the specified nodes are reachable (shown in green).

**Finishing the wizard**

1. Some database types insert one extra step here before the **Preview** page (an HBA Rules step for PostgreSQL/PostgreSQL Logical, an Extensions step for PostgreSQL/PostgreSQL Logical/TimescaleDB, or a Snapshot storage configuration step for Elasticsearch). See *Database-Specific Settings* if your chosen database type has one.
2. Click **Continue** to proceed to the **Preview** page. In this section, you can see the summary of your deployment and if everything is correct, you may proceed to deploy the cluster by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.
3. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

#### ClusterControl CLI

Every database type is created with the same `s9s cluster --create` command. Only `--cluster-type`, `--nodes`, `--vendor`, and a handful of type-specific flags change between database types. See *Database-Specific Settings* for a full working example per database type.

#### Terraform Provider for ClusterControl

Every database type uses the same `clustercontrol_db_cluster` resource. Only `db_cluster_type` and a handful of type-specific arguments change between database types. See *Database-Specific Settings* for a full working example per database type.

---

### 2.11.5 Database-Specific Settings

#### MySQL Replication

**Database dropdown value:** “MySQL Replication”

A minimum of two nodes is required for MySQL/MariaDB replication. If only one database IP address or hostname is provided, ClusterControl will deploy it as a standalone MySQL/MariaDB server with binary log enabled.

The following vendors and versions are supported for a new deployment:

- Oracle MySQL - 8.0 and 8.4.
- Percona Server for MySQL - 8.0 and 8.4.

- Percona Server for MySQL Pro - 8.0 and 8.4.
- MariaDB Server - 10.5, 10.6, 10.11, 11.4, 11.8.

ClusterControl supports configuring either replication scheme - asynchronous or semisynchronous. It is possible to set up a primary-primary replication (circular replication) under the **Add nodes** section. You can also add more replicas later after the deployment is completed.

### Default configuration

By default, ClusterControl deploys MySQL/MariaDB replication with the following configurations:

- MySQL GTID with `log_slave_updates` enabled (MySQL and Percona only).
- MariaDB GTID with `log_slave_updates` enabled (MariaDB only).
- All database nodes will be configured with `read_only=ON` and `super_read_only=ON` (if supported) for protection against accidental write. The chosen primary will be promoted by disabling the read-only in the runtime.
- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- ClusterControl will create and grant necessary privileges for the following database users:

```
| Database user      | Purpose                                      |
| ------------------ | -------------------------------------------- |
| `cmon`             | Management and automation.                   |
| `cmonexporter`     | Prometheus exporter for database monitoring. |
| `cmonagent`        | ClusterControl query monitoring agent.       |
| `cmon_replication` | MySQL/MariaDB replication.                   |
| `backupuser`       | Backup and restore management.               |
```

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

ClusterControl sets `read_only=ON` on all slaves but a privileged user (SUPER) can still write to a slave (except for MySQL versions that support `super_read_only`).

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically.
- **Admin/Root user**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPER privilege and the GRANT option for localhost only.
- **Admin/Root password**: The password for **Admin/Root user**.

- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Configuration template**: The dropdown loads all MySQL configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.
- **Semi-synchronous replication**: Toggling on (default) instructs ClusterControl to configure semi-synchronous replication. Toggling off instructs ClusterControl to configure asynchronous replication.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for single-primary replication. If you want to configure a multi-primary replication (circular replication), toggle on ***Use multi-primary replication***.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes.
- **Use multi-primary replication**: Toggling off (default) instructs ClusterControl to create the replication cluster as a single-primary replication. Toggling on opens another set of **Primary node** and **Replica nodes** form fields to be filled in for multi-primary replication. ClusterControl will set up the secondary primary node using circular replication but with read-only enabled.

### CLI and Terraform examples

#### ClusterControl CLI

- Create a three-node Oracle MySQL replication 8.0, with operating system user "ubuntu" (the first node is the primary):

```
s9s cluster --create \
    --cluster-type=mysqlreplication \
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=oracle \
    --provider-version=8.0 \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='PROD - MySQL Replication 8.0'
```

- Create a four-node MariaDB 11.4 multi-primary replication, using operating system user "root", with multiple primaries and replicas (note the `?` sign to identify the node's role in the `--nodes`

parameter, where the second primary will be a backup-primary) and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=mysqlreplication \
    --nodes=
↪"192.168.5.11?master;192.168.5.12?slave;192.168.7.11?master;192.168.7.12?slave"␣
↪\
    --vendor=mariadb \
    --provider-version=11.4 \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='System A - MariaDB 11.4' \
    --wait
```

### Terraform Provider for ClusterControl

- Deploy a 3-node Percona Server for MySQL replication 8.0 with semi-synchronous replication, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-replication-cluster" {

    db_cluster_create = true
    db_cluster_name = "PROD - MySQL Replication 8.0"
    db_cluster_type = "replication"
    db_version = "8.0"
    db_vendor = "percona"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
        hostname_internal = "db1.mydb.local"
    }
    db_host {
        hostname = "10.10.10.12"
        hostname_internal = "db2.mydb.local"
    }
    db_host {
        hostname = "10.10.10.13"
        hostname_internal = "db3.mydb.local"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true
    db_semi_sync_replication = true

}
```

### MySQL Galera

**Database dropdown value:** "MySQL Galera"

A minimal setup is comprised of one Galera node (no high availability, but this can later be scaled with more nodes). However, a minimum of 3 nodes is recommended for high availability. Garbd (an arbitrator) can be added later after the deployment completes.

The following vendors and versions are supported for a new deployment:

- Percona XtraDB Cluster - 8.0 and 8.4.
- Percona XtraDB Cluster Pro - 8.4.
- MariaDB Cluster - 10.5, 10.6, 10.11, 11.4 and 11.8.

### Default configuration

By default, ClusterControl deploys Galera Cluster with the following configurations:

- Use `xtrabackup-v2` or `mariabackup` (depending on the vendor chosen) for `wsrep_sst_method`.
- Binary logging is enabled.
- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | MySQL/MariaDB replication. |
| `backupuser` | Galera SST, backup and restore management. |

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically.
- **Admin/Root user**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPER privilege and the GRANT option for localhost only.
- **Admin/Root password**: The password for **Admin/Root user**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always

install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.

- **Version**: Choose a supported minor version from the dropdown. If ignored, ClusterControl will install the latest minor version of the current major version available at the moment.
- **Configuration template**: The dropdown loads all MySQL configuration templates file under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step. This option is ignored if the chosen vendor is Percona. Percona XtraDB Cluster 8.0 enables SSL by default regardless of this setting.

### Add nodes fields

- **Galera nodes**: Specify the IP address or hostname of the database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. A minimum of 3 nodes is recommended, and try to stick with an odd number of nodes in total to avoid a split-brain situation.

### CLI and Terraform examples

### ClusterControl CLI

- Create a three-node Percona XtraDB Cluster 8.0, with operating system user "ubuntu":

```
s9s cluster --create \
    --cluster-type=galera \
    --nodes="10.10.10.101;10.10.10.102;10.10.10.103" \
    --vendor=percona \
    --provider-version=8.0 \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='Percona XtraDB Cluster 8.0'
```

- Create a three-node MariaDB Cluster 11.4 and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=galera \
    --nodes="192.168.5.11;192.168.5.12;192.168.5.13" \
    --vendor=mariadb \
    --provider-version=11.4 \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='System A - MariaDB Cluster 11.4' \
    --wait
```

### Terraform Provider for ClusterControl

- Deploy a 3-node Percona XtraDB Cluster 8.0, with operating system user "ubuntu":

```
resource "clustercontrol_db_cluster" "my-galera-cluster" {

    db_cluster_create = true
    db_cluster_name = "Percona XtraDB Cluster 8.0"
    db_cluster_type = "galera"
    db_version = "8.0"
    db_vendor = "percona"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.101"
    }
    db_host {
        hostname = "10.10.10.102"
    }
    db_host {
        hostname = "10.10.10.103"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

---

### MySQL Group Replication

**Database dropdown value:** "MySQL Group Replication"

A minimal setup is comprised of one node (no high availability - can be scaled later with more nodes). A recommended setup should have a minimum of 3 nodes for high availability.

The following vendors and versions are supported for a new deployment:

- Oracle MySQL Group Replication - 8.0 and 8.4.

### Default configuration

By default, ClusterControl deploys Group Replication Cluster with the following configurations:

- By default, Clustercontrol parameter `auto_manage_readonly` is set to false.
- Set `loose-group_replication_single_primary_mode=OFF`, which means it is configured for multi-primary mode by default.
- Additional `my.cnf` settings:
  - `loose-group_replication_exit_state_action=ABORT_SERVER`
  - `loose-group_replication_ssl_mode=REQUIRED`

    - `loose-group_replication_recovery_use_ssl=ON`
- Binary logging is enabled.
- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `rpl_user` | MySQL group replication. |
| `backupuser` | Backup and restore management. |

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically.
- **Admin/Root user**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPER privilege and the GRANT option for localhost only.
- **Admin/Root password**: The password for **Admin/Root user**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Version**: Choose a supported minor version from the dropdown. If ignored, ClusterControl will install the latest minor version of the current major version available at the moment.
- **Configuration template**: The dropdown loads all MySQL configuration templates file under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type, by default ClusterControl uses `my.cnf.grouprepl` template. Choose one configuration template for the deployment. See *Configuration Template*.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step. This option is ignored if the chosen vendor is Percona. Percona XtraDB Cluster 8.0 enables SSL by default regardless of this setting.

### Add nodes fields

- **Node**: Specify the IP address or hostname of the database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. A minimum of 3 nodes is recommended, and try to stick with an odd number of nodes in total to avoid losing quorum.

### CLI and Terraform examples

### ClusterControl CLI

- Create a three-node Oracle MySQL Group Replication Cluster 8.0, with operating system user "ubuntu":

```
s9s cluster --create \
    --cluster-type=group_replication \
    --nodes="192.168.40.194;192.168.40.195;192.168.40.196" \
    --vendor=oracle \
    --provider-version=8.0 \
    --db-admin='root' \
    --db-admin-passwd='mydbPassw0rd' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='My Group Replication Cluster v8.0'
```

### Terraform Provider for ClusterControl

- Deploy a 3-node MySQL Group Replication Cluster 8.0, with operating system user "ubuntu":

```
resource "clustercontrol_db_cluster" "my-group-replication-cluster" {

    db_cluster_create = true
    db_cluster_name = "MySQL Group Replication Cluster 8.0"
    db_cluster_type = "group_replication"
    db_version = "8.0"
    db_vendor = "oracle"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "192.168.40.194"
    }
    db_host {
        hostname = "192.168.40.195"
    }
    db_host {
        hostname = "192.168.40.196"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
```

(continues on next page)

(continued from previous page)

```
    db_install_software = true

}
```

### PostgreSQL Streaming

**Database dropdown value:** "PostgreSQL Streaming"

A minimum of two nodes is required for PostgreSQL streaming replication. If only one database IP address or hostname is provided, ClusterControl will deploy it as a standalone PostgreSQL server. The following vendors and versions are supported for a new deployment:

- PostgreSQL from postgresql.org repository - 14, 15, 16, 17 and 18.
- PostgreSQL from EDB repository - 14, 15, 16 and 17 (requires a valid EDB repository token).

### Default configuration

By default, ClusterControl deploys PostgreSQL instances with the following configurations:

- Configure and load the `pg_stat_statements` module.
- The WAL level is set to `replica`.
- All replica will be set with `hot_standby`.
- Configure the `cluster_name` value.
- ClusterControl will configure the PostgreSQL instance with SSL encryption for client-server connections.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | PostgreSQL streaming replication. |
| `backupuser` | Backup and restore management. |

**Tip**

ClusterControl supports pgvector and PostGIS extensions which can be enabled with our PostgreSQL deployment wizard through an additional ***Extensions*** step, inserted after **Add nodes** and before **Preview** (see *Common Wizard Steps*). Once the cluster is deployed, you will need to enable the extension by using the `CREATE EXTENSION` statement.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically. If blank, ClusterControl will use the default path.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **User**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Configuration template**: The dropdown loads all PostgreSQL configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.
- **Additional HBA rules**: Defaults to **None**, ClusterControl's built-in rules only; no additional entries are appended to `pg_hba.conf`. Pick a rule set to append a pre-defined group of rules from `/etc/cmon/templates/`. Choose **Custom** to define the rules yourself in the next wizard step. These custom rules can be edited later from *Database Users and Privileges*.

### Extra step: HBA Rules

If you chose **Custom** for **Additional HBA rules** in the **Node configuration** step, an **HBA Rules** section is shown next, before **Add nodes**:

- Click **Add HBA Rule** to define a new rule. The list is empty until you add one, with columns **Type**, **Database**, **User**, **Address**, **Method**, **Info**, and **Action**.
- **Type**: The connection type this rule applies to. Default `host` (matches any TCP/IP connection). Other options are `hostssl` (SSL-only), `hostnossl` (non-SSL only), and `local` (Unix domain socket connections; no address required).
- **Database**: The database name this rule applies to. Default `all` (matches any database). Use `sameuser` to match a database with the same name as the connecting user, or enter a specific database name.
- **User**: The database user this rule applies to. Default `all` (matches any user), or enter a specific username.
- **Address**: The client IP address or CIDR range this rule applies to, for example `192.168.1.0/24` or `0.0.0.0/0` for all. Not applicable for the `local` connection type. Empty by default.
- **Method**: The authentication method. Default `scram-sha-256`. `scram-sha-256` and `md5` require a password, `trust` allows connections without a password, `reject` unconditionally denies the connection, `peer` uses the OS user name (local connections only), and `cert` uses SSL client certificates.

As you configure a rule, a live **Preview** message summarizes it in plain language, for example "Allow TCP connections from any address to any database for any user, using SCRAM-SHA-256 password authentication" or "Allow local Unix-socket connections to database 'db1' for user 'user1', using SCRAM-SHA-256 password authentication." The same text is shown when hovering over the info icon in the **Info** column once the rule is added to the list.

Rules are appended to `pg_hba.conf` in the order shown in the list, after PostgreSQL's built-in default rules and ClusterControl's replication and controller access entries. Use the ⋮ menu in the **Action** column to **Move to top**, **Move to bottom**, **Move up**, **Move down**, **Edit**, or **Delete** a rule.

Check **Save these rules as rule set** to have the controller save the rules above as `/etc/cmon/templates/pg_hba.conf.<name>` during this deployment, so other users see the rule set in the **Additional HBA rules** dropdown on their next deployment. If a rule set with the same name already exists, a numeric suffix (`.1`, `.2`, …) is appended.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for single-primary replication.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes.
    - **Synchronous Replication**: Toggling off (default) instructs ClusterControl to skip synchronous replication. Toggling on instructs ClusterControl to configure synchronous replication to the corresponding replica. You can have a mix of synchronous and asynchronous replication for multiple replicas in a cluster.

### CLI and Terraform examples

### ClusterControl CLI

- Create a three-node PostgreSQL 16 streaming replication, with operating system user "ubuntu" (the first node is the primary) with tags `production` and `postgres`:

```
s9s cluster --create \
    --cluster-type=postgresql \
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=postgresql \
    --provider-version=16 \
    --db-admin='postgres' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --with-tags='production;postgres' \
    --cluster-name='PostgreSQL 16 - Streaming Replication'
```

- Create a four-node PostgreSQL EDB 16 streaming replication, using operating system user "root" (the first node is the primary) and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=postgresql \
    --nodes="192.168.5.11;192.168.5.12;192.168.5.13;192.168.5.14" \
    --vendor=enterprisedb \
    --provider-version=11.4 \
    --db-admin='postgres' \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --enterprise-token='XXXXXXXXXXXXXXXXXXXXXXXXX' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='System A - PostgreSQL EDB 16 - Streaming␣
↪Replication' \
    --wait
```

### Terraform Provider for ClusterControl

- Deploy a 3-node PostgreSQL streaming replication 16.0, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-postgresql-cluster" {

    db_cluster_create = true
    db_cluster_name = "PostgreSQL Streaming Replication 16"
    db_cluster_type = "postgresql"
    db_version = "16"
    db_vendor = "postgresql"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
    }
    db_host {
        hostname = "10.10.10.12"
    }
    db_host {
        hostname = "10.10.10.13"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

### PostgreSQL Logical

**Database dropdown value:** "PostgreSQL Logical"

Deploys a new PostgreSQL logical replication cluster from ClusterControl. A PostgreSQL logical cluster is built up of multiple PostgreSQL streaming clusters. Logical replication occurs between these streaming clusters, and each streaming cluster can have both publications and subscriptions. A minimum of two nodes is required for PostgreSQL logical replication, which is represented in the **Cluster 1** and **Cluster 2** sections under **Add Nodes**. Cluster 1 can be the target publisher and Cluster 2 can be the target subscriber (or vice versa). Multi-primary replication is possible by configuring two-way publication and subscription links of the same supported database objects.

The following vendors and versions are supported for a new deployment:

- PostgreSQL from postgresql.org repository - 16, 17 and 18.
- PostgreSQL from EDB repository - 16, 17 and 18 (requires a valid EDB repository token).

### Default configuration

By default, ClusterControl deploys PostgreSQL instances with the following configurations:

- Configure and load the `pg_stat_statements` module.
- The WAL level is set to `logical`.
- Configure the `cluster_name` value.
- ClusterControl will configure the PostgreSQL instance with SSL encryption for client-server connections.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | PostgreSQL streaming replication. |
| `cmon_logical_replication` | PostgreSQL logical replication. |
| `backupuser` | Backup and restore management. |

**Tip**

ClusterControl supports pgvector and PostGIS extensions which can be enabled with our PostgreSQL deployment wizard through an additional ***Extensions*** step, inserted after **Add nodes** and before **Preview** (see *Common Wizard Steps*). Once the cluster is deployed, you will need to enable the extension by using the `CREATE EXTENSION` statement.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.

- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically. If blank, ClusterControl will use the default path.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **User**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Configuration template**: The dropdown loads all PostgreSQL configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.
- **Additional HBA rules**: Defaults to **None**, ClusterControl's built-in rules only; no additional entries are appended to `pg_hba.conf`. Pick a rule set to append a pre-defined group of rules from `/etc/cmon/templates/`. Choose **Custom** to define the rules yourself in the next wizard step. These custom rules can be edited later from *Database Users and Privileges*.

### Extra step: HBA Rules

Identical to the *HBA Rules step documented for PostgreSQL Streaming* above — shown only if you chose **Custom** for **Additional HBA rules**.

### Add nodes fields

The **Add nodes** section is split into two blocks, **Cluster 1** and **Cluster 2**, each with the same fields:

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for single-primary replication.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes.
    - **Synchronous Replication**: Toggling off (default) instructs ClusterControl to skip synchronous replication. Toggling on instructs ClusterControl to configure synchronous replication to the corresponding replica. You can have a mix of synchronous and asynchronous replication for multiple replicas in a cluster.

### CLI and Terraform examples

#### ClusterControl CLI

- Create a two-node PostgreSQL 16 logical replication, with operating system user "ubuntu" (the first node is the primary) with tags `production` and `postgres`:

```
s9s cluster --create \
    --cluster-type=postgresql_logical \
    --nodes="10.10.10.11;10.10.10.12" \
    --vendor=postgresql \
    --provider-version=16 \
    --db-admin='postgres' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --with-tags='production;postgres' \
    --cluster-name='PostgreSQL 16 - Logical Replication'
```

- Create a two-node PostgreSQL EDB 16 logical replication, using operating system user "root" (the first node is the primary) and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=postgresql_logical \
    --nodes="192.168.5.11;192.168.5.12;" \
    --vendor=enterprisedb \
    --provider-version=11.4 \
    --db-admin='postgres' \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --enterprise-token='XXXXXXXXXXXXXXXXXXXXXXXXX' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='System A - PostgreSQL EDB 16 - Logical Replication
↪' \
    --wait
```

#### Terraform Provider for ClusterControl

- Deploy a 2-node PostgreSQL logical replication 17.0, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-postgresql-cluster" {

    db_cluster_create = true
    db_cluster_name = "PostgreSQL Logical Replication 16"
    db_cluster_type = "postgresql_logical"
    db_version = "17"
    db_vendor = "postgresql"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
    }
```

(continues on next page)

(continued from previous page)

```
    db_host {
        hostname = "10.10.10.12"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

### TimescaleDB

**Database dropdown value:** "TimescaleDB"

Deploys a new TimescaleDB standalone or streaming replication cluster from ClusterControl. A minimum of two nodes is required for TimescaleDB streaming replication. If only one database IP address or hostname is provided, ClusterControl will deploy it as a standalone TimescaleDB server. The following vendors and versions are supported for a new deployment:

- TimescaleDB - 14, 15, 16, 17 and 18.

### Default configuration

By default, ClusterControl deploys TimescaleDB with the following configurations:

- Configure and load the `pg_stat_statements` module and `timescaledb` extension.
- The WAL level is set to `replica`.
- All replica will be set with `hot_standby`.
- Configure the `cluster_name` value.
- ClusterControl will configure the PostgreSQL instance with SSL encryption for client-server connections.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | PostgreSQL streaming replication. |
| `backupuser` | Backup and restore management. |

**Note**

Note You can also deploy a PostgreSQL and convert it to TimescaleDB at a later stage. However, this action will be irreversible and ClusterControl will treat the cluster as TimescaleDB onwards.

**Tip**

Same as PostgreSQL, TimescaleDB deployment also includes an additional ***Extensions*** step (inserted after **Add nodes** and before **Preview**) to toggle on extensions you want installed together with the database. Once the cluster is deployed, you will need to enable the extension by using the `CREATE EXTENSION` statement.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically. If blank, ClusterControl will use the default path.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **User**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Configuration template**: The dropdown loads all TimescaleDB configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for streaming replication.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes.
  - **Synchronous Replication**: Toggling off (default) instructs ClusterControl to skip synchronous replication. Toggling on instructs ClusterControl to configure synchronous replication to the

corresponding replica. You can have a mix of synchronous and asynchronous replication for multiple replicas in a cluster.

### CLI and Terraform examples

#### ClusterControl CLI

- Create a three-node TimescaleDB 16 streaming replication, with operating system user "ubuntu" (the first node is the primary) with tags `production` and `timescaledb`:

```
s9s cluster --create \
    --cluster-type=postgresql \
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=postgresql \
    --provider-version=16 \
    --db-admin='postgres' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --with-tags='production;timescaledb' \
    --cluster-name='TimescaleDB 16 - Streaming Replication' \
    --with-timescaledb
```

- Create a standalone TimescaleDB 15, with operating system user "root" with tags `test` and `timescaledb`, and wait until the job finishes:

```
s9s cluster --create \
    --cluster-type=postgresql \
    --nodes="192.168.99.11" \
    --vendor=postgresql \
    --provider-version=15 \
    --db-admin='postgres' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --with-tags='test;timescaledb' \
    --cluster-name='TimescaleDB 16 - standalone' \
    --with-timescaledb \
    --wait
```

#### Terraform Provider for ClusterControl

- Deploy a 3-node TimescaleDB streaming replication 16.0, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-timescaledb-cluster" {

    db_cluster_create = true
    db_cluster_name = "TimescaleDB Streaming Replication 16"
    db_cluster_type = "postgresql"
    db_version = "16"
    db_vendor = "postgresql"
```

(continues on next page)

(continued from previous page)

```
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
    }
    db_host {
        hostname = "10.10.10.12"
    }
    db_host {
        hostname = "10.10.10.13"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true
    db_enable_timescale = true

}
```

### MongoDB Replica Set

**Database dropdown value:** "MongoDB ReplicaSet"

Deploys a new MongoDB standalone or replica set. The database cluster will be automatically added to ClusterControl once deployed. A minimum of three nodes (including a MongoDB arbiter) is recommended for a replica set. If only one node is specified, ClusterControl will deploy it as a standalone MongoDB node. The following vendors and versions are supported for a new deployment:

- MongoDB Community - 7.0 and 8.0.
- MongoDB Enterprise - 7.0 and 8.0.
- Percona Server for MongoDB - 7.0 and 8.0.

### Default configuration

By default, ClusterControl deploys MongoDB Replica Set members with the following configurations:

- Configure `setParameter.enableLocalhostAuthBypass: true` inside MongoDB configuration file.
- ClusterControl will create and grant necessary roles for additional MongoDB users - `admin.cmon_backup` for backup and restore purposes and `admin.cmonexporter` for query monitoring.

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

It is possible to deploy only 2 MongoDB nodes (without an arbiter). The caveat of this approach is no automatic failover. If the primary node goes down then manual failover is required to make the other server run as primary. Automatic failover works fine with 3 nodes and more.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted the built-in root role.
- **Password**: The password for **User**.
- **ReplicaSet name**: Specify the name of the replica set, similar to `replication.replSetName` option in MongoDB.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Configuration template**: The dropdown loads all MongoDB configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for a replica set configuration.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If this field is left blank, ClusterControl will deploy the primary node as a standalone node.
    - **Act as an arbiter**: Toggling off (default) instructs ClusterControl to configure the node as a standard MongoDB replica. Toggling on instructs ClusterControl to configure the node as a MongoDB arbiter. An arbiter participates in elections for primary but does not have a copy of the data set and cannot become primary.

### CLI and Terraform examples

#### ClusterControl CLI

- Create a three-node MongoDB Enterprise replica set 7.0, with operating system user "ubuntu" (the first node is the primary) and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=mongodbenterprise \
    --nodes="192.168.99.13;192.168.99.15;192.168.99.19" \
    --vendor=mongodbenterprise \
    --provider-version='7.0' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --db-admin='admin' \
    --db-admin-passwd='MyS3cr3tPass' \
    --cluster-name='MongoDB Enterprise ReplicaSet 7.0' \
    --wait
```

- Create a three-node MongoDB Community replica set 7.0, with operating system user "root" (the first node is the primary and the last node is an arbiter) and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=mongodb \
    --nodes="192.168.99.13;192.168.99.15;192.168.99.19?arbiter" \
    --vendor=mongodb \
    --provider-version='7.0' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --db-admin='admin' \
    --db-admin-passwd='MyS3cr3tPass' \
    --cluster-name='MongoDB ReplicaSet 7.0' \
    --wait
```

- Create a standalone MongoDB Community server 6.0, using operating system user "root" and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=mongodb \
    --nodes="192.168.5.11" \
    --vendor=mongodb \
    --provider-version='6.0' \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='MongoDB 6.0 - standalone - devtest' \
    --wait
```

### Terraform Provider for ClusterControl

- Deploy a 3-node replica set of Percona Server for MongoDB 7.0, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-mongodb-replicaset" {

    db_cluster_create = true
    db_cluster_name = "Percona MongoDB Replica Set 7.0"
    db_cluster_type = "mongodb"
    db_version = "7.0"
    db_vendor = "percona"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
    }
    db_host {
        hostname = "10.10.10.12"
    }
    db_host {
        hostname = "10.10.10.13"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

---

### MongoDB Sharded Cluster

**Database dropdown value:** "MongoDB Shards"

Deploys a new MongoDB Sharded Cluster. The database cluster will be automatically added to ClusterControl once deployed. For production deployment, it is recommended to have at least 8 nodes for a high-availability setup:

- 2 nodes for the mongos (router),
- 3 nodes for the config server (replica set),
- 3 nodes per shard (replica set).

For a very minimal setup (suitable for testing and experimenting), you can start with two nodes and deploy it as below:

- 1 node for config server (one-node replica set) and mongos (co-locate together),
- 1 node for shard (one-node replica set).

The following vendors and versions are supported for a new deployment:

- MongoDB Enterprise - 7.0 and 8.0.

- MongoDB Community - 7.0 and 8.0.
- Percona Server for MongoDB - 7.0 and 8.0.

### Default configuration

By default, ClusterControl deploys MongoDB Sharded Cluster with the following configurations:

- Configure `setParameter.enableLocalhostAuthBypass: true` inside MongoDB configuration file.
- ClusterControl will create and grant necessary privileges for an additional MongoDB user - `admin.cmon_backup` for backup and restore purposes.

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

MongoDB Sharded Cluster does not support `mongodump` backup method. Users will be asked to install Percona Backup for MongoDB (PBM) when creating or scheduling a backup for this cluster type after the deployment completes. Note that PBM requires shared remote backup storage (e.g., NFS) accessible on all MongoDB nodes.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted the built-in root role.
- **Password**: The password for **User**.
- **Configuration template**: The dropdown loads all MongoDB configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Router configuration template**: The dropdown loads all MongoDB mongos configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.

- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.

### Add nodes fields (Configuration Servers and Routers, then Shard)

This database type replaces the generic **Add nodes** step with two steps:

**Configuration Servers and Routers**

- **Add configuration server**: Specify the IP address or hostname of the MongoDB config servers. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. For high availability, at least 3 nodes are required.
  - **Port**: MongoDB server port for MongoDB config server (config). The default is 27019.
- **Node router server**: Specify the IP address or hostname of the MongoDB mongos servers. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. It is possible to co-locate mongos and config servers together.
  - **Port**: MongoDB server port for MongoDB router server (mongos). The default is 27017.

**Shard**

- **ReplicaSet name**: Specify the name of the replica set, similar to `replication.replSetName` option in MongoDB.
- **Port**: MongoDB server port for MongoDB shard server (mongod). The default is 27018.
- **Add nodes to the shard**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for a replica set configuration.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If this field is left blank, ClusterControl will configure the shard with a single primary node.
  - **Replication delay** - Specify the amount of time the secondary member should be delayed in seconds.
  - **Priority** - Set the secondary member priority. A bigger integer represents a higher priority. A delayed secondary member should be set to 0 because it is not fit to become primary and should be hidden from the application.
- **Add another shard**: Add another shard section to define another replica set of the shard. The same **Shard** form will be presented and you can add as many as 50 shards per cluster.
- **Remove this shard**: Discard the current shard.

**Note**

Note After deployment completes, it is the database administrator job to enable sharding at the database and collection level.

### CLI and Terraform examples

#### ClusterControl CLI

- Create a MongoDB Community sharded cluster 7.0 which consists of 3 mongos, 3 MongoDB config, and one shard consists of a three-node replica set called 'replset2' (with `priority` and `slave_delay` options), operating system user "root" and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=mongodb \
    --nodes=
↪"mongos://192.168.1.11;mongos://192.168.1.12;mongos://192.168.1.12;mongocfg://192.
↪\
    --vendor=mongodb \
    --provider-version='7.0' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --db-admin='admin' \
    --db-admin-passwd='MyS3cr3tPass' \
    --cluster-name='MongoDB ReplicaSet 7.0' \
    --wait
```

- Create a minimal 2-node MongoDB sharded cluster 6.0, using operating system user "root" and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=mongodb \
    --nodes=
↪"mongos://192.168.5.11:27017;mongocfg://192.168.5.11:27019;192.168.5.12:27018"␣
↪\
    --vendor=mongodb \
    --provider-version='6.0' \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='MongoDB Sharded Cluster 6.0 - minimal - devtest' \
    --wait
```

#### Terraform Provider for ClusterControl

- Deploy a 11-node MongoDB Sharded Cluster 7.0, with a 3-node config server replica set, a 6-node database nodes (2 replica sets, 3 nodes per shard) and 2-node mongos with operating system user "ubuntu" (the first node under `db_replica_set` will be the primary):

```
resource "clustercontrol_db_cluster" "my-mongodb-sharded-cluster" {

    db_cluster_create = true
    db_cluster_name = "MongoDB Sharded Cluster Set 7.0"
    db_cluster_type = "mongodb"
    db_version = "7.0"
    db_vendor = "mongodb"
    db_admin_user_password = var.cc_api_user_password
    db_config_server {
        rs = "rs_config"
        member {
            hostname = "192.168.10.201"
        }
        member {
            hostname = "192.168.10.202"
        }
        member {
            hostname = "192.168.10.203"
        }
    }

    db_mongos_server {
        hostname = "192.168.10.101"
    }

    db_mongos_server {
        hostname = "192.168.10.102"
    }

    db_replica_set {
        rs = "rs1"
        member {
            hostname = "192.168.10.111"
        }
        member {
            hostname = "192.168.10.112"
        }
        member {
            hostname = "192.168.10.113"
        }
    }

    db_replica_set {
        rs = "rs2"
        member {
            hostname = "192.168.10.121"
        }
        member {
            hostname = "192.168.10.122"
        }
```

(continues on next page)

(continued from previous page)

```
        member {
            hostname = "192.168.10.123"
        }
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

### Valkey Sentinel

**Database dropdown value:** "Valkey Sentinel"

Deploys new Valkey replication instances with Valkey Sentinel. A minimum of 3 nodes is recommended for high availability and automatic failover. The following vendors and versions are supported for a new deployment:

- Valkey Sentinel - 7 and 8.
- Canonical - 7.

For a very minimal setup (suitable for testing and experimenting), it is possible to deploy a single-node Valkey instance with Sentinel (both services co-located on the same node), and later can be scaled out with more Valkey database and Sentinel nodes.

### Default configuration

By default, ClusterControl deploys Valkey instances with the following configurations:

- Valkey Sentinel (default port is 26379) will be co-located with the Valkey instances (default port is 6379).
- ClusterControl will configure the Valkey instance with `appendonly` enabled.
- ClusterControl will secure the instance with authentication enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the database nodes, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Valkey being killed by OOM.

!!! Tips

```
Valkey Sentinel requires 3 nodes for automatic primary promotion. Sentinel␣
↪can be co-located on the ClusterControl server if you want to deploy a␣
↪two-node Valkey replication cluster (Sentinel will be co-located on each␣
↪database instance).
```

### Node configuration fields

- **Valkey port**: The database server port that ClusterControl shall configure on all Valkey database nodes.
- **Valkey Sentinel port**: The database server port that ClusterControl shall configure on all Valkey Sentinel nodes.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.
- **Password**: The Valkey admin password that ClusterControl shall configure on all database nodes. The value will be used to configure the `requirepass` and `masterauth` options.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for a replication with Sentinel configuration.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify two or more replica nodes. If this field is left blank, ClusterControl will deploy the primary node as a standalone node.

### CLI and Terraform examples

### ClusterControl CLI

- Create a three-node Valkey Sentinel 8.0, with operating system user "ubuntu" (the first node is the primary), where the admin user password will be auto-generated, let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=redis \
    --nodes=
↪"redis://10.0.0.11;redis://10.0.0.12;redis://10.0.0.13;redis-sentinel://10.0.0.11;
↪\
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --vendor=valkey \
    --provider-version=8  \
```

(continues on next page)

(continued from previous page)

```
    --log \
    --cluster-name="My Valkey Sentinel v8"
```

- Create a standalone Valkey server (with Sentinel) 7.0, using operating system user "root" and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=redis \
    --nodes="redis://10.0.0.11;redis-sentinel://10.0.0.11" \
    --vendor=valkey \
    --provider-version=7 \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='Valkey 7 - standalone - devtest' \
    --wait
```

### Terraform Provider for ClusterControl

- Deploy a three-node Valkey with Sentinel 7.0, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-valkey-sentinel" {

    db_cluster_create = true
    db_cluster_name = "Valkey Sentinel 7"
    db_cluster_type = "redis"
    db_version = "7"
    db_vendor = "valkey"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
    }
    db_host {
        hostname = "10.10.10.12"
    }
    db_host {
        hostname = "10.10.10.13"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

### Redis Sentinel

**Database dropdown value:** "Redis Sentinel"

Deploys new Redis replication instances with Redis. A minimum of 3 nodes is recommended for high availability and automatic failover. The following vendors and versions are supported for a new deployment:

- Redis Sentinel - 6, 7 and 8.

For a very minimal setup (suitable for testing and experimenting), it is possible to deploy a single-node Redis instance with Sentinel (both services co-located on the same node), and later can be scaled out with more Redis database and Sentinel nodes.

**Info**

ClusterControl and Severalnines do not use or reproduce the official Redis® logo, which is a registered trademark of Redis Ltd. To comply with Redis trademark and branding guidelines, we have introduced our own distinct branding for Redis-related features within ClusterControl. This logo and name are original creations of Severalnines and are used solely as internal branding to represent Redis.

### Default configuration

By default, ClusterControl deploys Redis instances with the following configurations:

- Redis Sentinel (default port is 26379) will be co-located with the Redis instances (default port is 6379).
- ClusterControl will configure the Redis instance with `appendonly` enabled.
- ClusterControl will secure the instance with authentication enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the database nodes, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Redis being killed by OOM.

!!! Tips

```
Redis Sentinel requires 3 nodes for automatic primary promotion. Sentinel␣
↪can be co-located on the ClusterControl server if you want to deploy a␣
↪two-node Redis replication cluster (Sentinel will be co-located on each␣
↪database instance).
```

### Node configuration fields

- **Redis port**: The database server port that ClusterControl shall configure on all Redis database nodes.
- **Redis Sentinel port**: The database server port that ClusterControl shall configure on all Redis Sentinel nodes.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases);

ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.

- **Password**: The Redis admin password that ClusterControl shall configure on all database nodes. The value will be used to configure the `requirepass` and `masterauth` options.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for a replication with Sentinel configuration.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify two or more replica nodes. If this field is left blank, ClusterControl will deploy the primary node as a standalone node.

### CLI and Terraform examples

### ClusterControl CLI

- Create a three-node Redis Sentinel 7.0, with operating system user "ubuntu" (the first node is the primary), where the admin user password will be auto-generated, let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=redis \
    --nodes=
↪"redis://10.0.0.11;redis://10.0.0.12;redis://10.0.0.13;redis-sentinel://10.0.0.11;
↪\
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --vendor=redis \
    --provider-version=7  \
    --log \
    --cluster-name="My Redis Sentinel v7"
```

- Create a standalone Redis server (with Sentinel) 6.0, using operating system user "root" and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=redis \
    --nodes="redis://10.0.0.11;redis-sentinel://10.0.0.11" \
    --vendor=redis \
    --provider-version=6 \
    --db-admin-passwd='Str0ngpa$$w0rd' \
```

(continues on next page)

(continued from previous page)

```
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='Redis 6 - standalone - devtest' \
    --wait
```

### Terraform Provider for ClusterControl

- Deploy a three-node Redis Sentinel 7.0, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-redis-sentinel" {

    db_cluster_create = true
    db_cluster_name = "Redis with Sentinel 7"
    db_cluster_type = "redis"
    db_version = "7"
    db_vendor = "redis"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
    }
    db_host {
        hostname = "10.10.10.12"
    }
    db_host {
        hostname = "10.10.10.13"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

---

### Valkey Cluster

**Database dropdown value:** "Valkey Cluster"

Deploy a new Valkey Cluster. A minimum of 3 nodes is recommended for automatic failover. Valkey Cluster requires at least 3 nodes for a standard cluster setup with 3 shards. However, each shard should have its replica, hence 6 nodes are recommended (1 primary + 1 replica per shard) for production deployment:

- 2 nodes (1 primary + 1 replica) for shard 1.
- 2 nodes (1 primary + 1 replica) for shard 2.
- 2 nodes (1 primary + 1 replica) for shard 3.

The following vendors and versions are supported for a new deployment:

- Valkey Cluster - 7 and 8.
- Canonical - 7.

### Default configuration

By default, ClusterControl deploys Valkey cluster instances with the following configurations:

- ClusterControl will configure the Valkey instances with `appendonly` enabled.
- ClusterControl will secure the instance with AUTH enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the cluster, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Valkey being killed by OOM.

### Node configuration fields

- **Valkey port**: The database server port that ClusterControl shall configure on all Valkey database nodes.
- **Cluster bus port**: The database server port that ClusterControl shall configure for Valkey cluster communication.
- **Node timeout (ms)**: The maximum amount of time a Valkey Cluster node can be unavailable, without it being considered as failing. If a master node is not reachable for more than the specified amount of time, it will be failed over by its replicas. This parameter controls other important things in the Valkey Cluster. Notably, every node that can't reach the majority of master nodes for the specified amount of time, will stop accepting queries.
- **Replica validity factor**: Specify the replication validity factor to consider a replica that is disconnected from the primary for more than **Node timeout** multiplied by this value. Set the factor to 0 to always consider a replica valid to failover. If the value is positive, a maximum disconnection time is calculated as the node timeout value multiplied by the factor provided with this option, and if the node is a replica, it will not try to start a failover if the master link was disconnected for more than the specified amount of time.
- **Configuration template**: The dropdown loads all Valkey configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.
- **Username**: The Valkey admin username that ClusterControl shall configure on all database nodes. The value will be used to configure the `requirepass` and `masterauth` options.
- **Password**: The Valkey admin password that ClusterControl shall configure on all database nodes. The value will be used to configure the `requirepass` and `masterauth` options.

### Add nodes fields (Shard 1, Shard 2, ...)

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for a replication with Sentinel configuration.
- **Replica nodes**: Specify the IP address or hostname of the replica database node for this particular shard. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If no replica nodes, ClusterControl will configure the shard as a single primary setup.
- **Add another shard**: Add another shard section to define another replica set of the shard. A minimum of 3 shards is required. A similar **Shard** section will be presented where you can add another set of primary and replica nodes.
- **Remove this shard**: Discard the current shard.

## CLI examples

### ClusterControl CLI

- Create a 6-node Valkey Cluster 7.0, with operating system user "root", let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=redis-sharded \
    --redis-port=6379 \
    --redis-bus-port=16379 \
    --vendor=valkey \
    --provider-version=7 \
    --node-timeout-ms=5000 \
    --replica-validity-factor=10 \
    --db-admin=admin \
    --db-admin-pass='Str0ngpa$$w0rd' \
    --nodes=
↪"redis-primary://192.168.0.101:6479;redis-replica://192.168.0.102:6479;redis-prima
↪\
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name="My Valkey Cluster v7" \
    --log
```

- Create a 6-node Valkey Cluster 7.0, with operating system user "ubuntu", where the admin user password will be auto-generated, let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=redis-sharded \
    --redis-port=6379 \
```

(continues on next page)

(continued from previous page)

```
    --redis-bus-port=16379 \
    --vendor=valkey \
    --provider-version=7 \
    --node-timeout-ms=5000 \
    --replica-validity-factor=10 \
    --db-admin=admin \
    --nodes=
↪"redis-primary://192.168.0.101:6479;redis-replica://192.168.0.102:6479;redis-prima
↪\
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name="My Valkey Cluster v7" \
    --log
```

### Redis Cluster

**Database dropdown value:** "Redis Cluster"

Deploy a new Redis Cluster. A minimum of 3 nodes is recommended for automatic failover. Redis Cluster requires at least 3 nodes for a standard cluster setup with 3 shards. However, each shard should have its replica, hence 6 nodes are recommended (1 primary + 1 replica per shard) for production deployment:

- 2 nodes (1 primary + 1 replica) for shard 1.
- 2 nodes (1 primary + 1 replica) for shard 2.
- 2 nodes (1 primary + 1 replica) for shard 3.

The following vendors and versions are supported for a new deployment:

- Redis Cluster - 6, 7 and 8.

**Info**

ClusterControl and Severalnines do not use or reproduce the official Redis® logo, which is a registered trademark of Redis Ltd. To comply with Redis trademark and branding guidelines, we have introduced our own distinct branding for Redis-related features within ClusterControl. This logo and name are original creations of Severalnines and are used solely as internal branding to represent Redis.

### Default configuration

By default, ClusterControl deploys Redis cluster instances with the following configurations:

- ClusterControl will configure the Redis instances with `appendonly` enabled.
- ClusterControl will secure the instance with AUTH enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the cluster, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Redis being killed by OOM.

### Node configuration fields

- **Redis port**: The database server port that ClusterControl shall configure on all Redis database nodes.
- **Cluster bus port**: The database server port that ClusterControl shall configure for Redis cluster communication.
- **Node timeout (ms)**: The maximum amount of time a Redis Cluster node can be unavailable, without it being considered as failing. If a master node is not reachable for more than the specified amount of time, it will be failed over by its replicas. This parameter controls other important things in the Redis Cluster. Notably, every node that can't reach the majority of master nodes for the specified amount of time, will stop accepting queries.
- **Replica validity factor**: Specify the replication validity factor to consider a replica that is disconnected from the primary for more than **Node timeout** multiplied by this value. Set the factor to 0 to always consider a replica valid to failover. If the value is positive, a maximum disconnection time is calculated as the node timeout value multiplied by the factor provided with this option, and if the node is a replica, it will not try to start a failover if the master link was disconnected for more than the specified amount of time.
- **Configuration template**: The dropdown loads all Redis or Valkey configuration template files under `/etc/cmon/templates` or `/usr/share/cmon/templates` of the ClusterControl node for this cluster type. Choose one configuration template for the deployment. See *Configuration Template*.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.
- **Username**: The Redis admin username that ClusterControl shall configure on all database nodes. The value will be used to configure the `requirepass` and `masterauth` options.
- **Password**: The Redis admin password that ClusterControl shall configure on all database nodes. The value will be used to configure the `requirepass` and `masterauth` options.

### Add nodes fields (Shard 1, Shard 2, ...)

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for a replication with Sentinel configuration.
- **Replica nodes**: Specify the IP address or hostname of the replica database node for this particular shard. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If no replica nodes, ClusterControl will configure the shard as a single primary setup.
- **Add another shard**: Add another shard section to define another replica set of the shard. A minimum of 3 shards is required. A similar **Shard** section will be presented where you can add another set of primary and replica nodes.

- **Remove this shard**: Discard the current shard.

### CLI examples

#### ClusterControl CLI

- Create a 6-node Redis Cluster 6.0, with operating system user "root", let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=redis-sharded \
    --redis-port=6379 \
    --redis-bus-port=16379 \
    --vendor=redis \
    --provider-version=6 \
    --node-timeout-ms=5000 \
    --replica-validity-factor=10 \
    --db-admin=admin \
    --db-admin-pass='Str0ngpa$$w0rd' \
    --nodes=
↪"redis-primary://192.168.0.101:6479;redis-replica://192.168.0.102:6479;redis-prima
↪\
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name="My Redis Cluster v6" \
    --log \
    --print-request
```

- Create a 6-node Redis Cluster 7.0, with operating system user "ubuntu", where the admin user password will be auto-generated, let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=redis-sharded \
    --redis-port=6379 \
    --redis-bus-port=16379 \
    --vendor=redis \
    --provider-version=7 \
    --node-timeout-ms=5000 \
    --replica-validity-factor=10 \
    --db-admin=admin \
    --nodes=
↪"redis-primary://192.168.0.101:6479;redis-replica://192.168.0.102:6479;redis-prima
↪\
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name="My Valkey Cluster v7" \
    --log \
    --print-request
```

### SQL Server

**Database dropdown value:** "SQL Server"

Deploys a new standalone SQL Server instance or a SQL Server AlwaysOn high-availability setup. If only one database node is specified, ClusterControl will deploy it as standalone SQL Server. The following vendors and versions are supported for a new deployment:

- Microsoft SQL Server for Linux - 2019, 2022 and 2025.

**Warning**

Attention

Only hostname or FQDN is supported. IP addresses are not accepted and will result in an error during the node addition process. Ensure proper host naming and mapping is configured beforehand, utilizing either `/etc/hosts` or a DNS resolver.

### License

Deploying SQL Server with ClusterControl does not change how you must license SQL Server. Each node where SQL Server is running requires appropriate licensing, just as it would if you deployed SQL Server manually. The presence of ClusterControl as an orchestration or management layer has no bearing on Microsoft's licensing rules. In a new deployment scenario, ClusterControl configures the SQL Server instance with the Evaluation edition, which is free to use for 180 days.

When deploying SQL Server in a clustered environment, licensing rules still apply to each server running SQL Server. However, the specifics differ depending on whether the node is actively running SQL Server or is a passive failover node:

- Active Nodes: Each active node that runs SQL Server services needs to be fully licensed.
- Passive Failover Nodes: Under current SQL Server licensing rules, you can typically have one passive failover node per licensed instance without needing an additional license for that passive node. The passive node is considered a "standby" server that can take over if the active server fails. It must truly remain passive, running no active workloads except those required for maintaining the standby state.

Obtain your SQL Server license directly from Microsoft, a certified Microsoft Partner, or a cloud marketplace (if applicable).

### Default configuration

By default, ClusterControl deploys SQL Server with the following configurations:

- All database nodes must use fully qualified domain name (FQDN). IP address is not supported.
- Minimum available RAM is 1800 MB per database node.
- Enforces the SQL Server user's password policy as shown here.
- At the moment, ClusterControl only deploys AlwaysOn with asynchronous-commit mode, where it does not wait for any secondary replica to write incoming transaction log records to disk.
- ClusterControl will set up the SQL Server using the Evaluation edition (free, 180-day limit).

### Node configuration fields

- **Admin username**: Specify the SQL Server admin username.
- **Admin password**: The password for **Admin username**. The password must comply with the SQL Server password policy. See Password Policy. Click on the eye icon to view the unmasked password. Copy this value to somewhere safe, or you may generate a new one. It is recommended that you use the generated password as it meets the minimum requirements.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.
- **Enable SSL encryption**: Toggling on (default) instructs ClusterControl to configure database SSL encryption for both client-server and replication. Toggling off skips this step.

### Add nodes fields

- **Primary node**: Specify hostname or FQDN of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for SQL Server AlwaysOn availability group configuration.
- **Replica nodes**: Specify hostname or FQDN of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If no replica nodes, ClusterControl will configure the cluster as a standalone setup. You can add up to 8 replica nodes per availability group.

### CLI and Terraform examples

### ClusterControl CLI

- Create a 3-node SQL Server 2022 with Always On availability group, with operating system user "root", let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=mssql_ao_async \
    --vendor=microsoft \
    --provider-version=2022 \
    --db-admin=SQLServerAdmin \
    --db-admin-pass='NqR$feGR68xli^#v' \
    --nodes=
↪"mssql://dsa-mssql-db1:1433;dsa-mssql-db2:1433;dsa-mssql-db3:1433;" \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name="MSSQL Server 2022 Always On" \
    --log
```

- Create a standalone SQL Server 2019, with operating system user "ubuntu", let the deployment job run in the foreground and show the job logs:

```
s9s cluster --create \
    --cluster-type=mssql_single \
    --vendor=microsoft \
    --provider-version=2019 \
    --db-admin=SQLServerAdmin \
    --db-admin-pass='NqR$feGR68xli^#v' \
    --nodes="mssql://dsa-mssql-db1:1433" \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name="Standalone SQL Server 2019" \
    --log
```

### Terraform Provider for ClusterControl

- Deploy a three-node replica set of SQL Server 2022, with operating system user "ubuntu" (the first node will be the primary):

```
resource "clustercontrol_db_cluster" "my-mssql-2022" {
    db_cluster_create = true
    db_cluster_name = "Microsoft SQL Server 2022"
    db_cluster_type = "mssql_ao_async"
    db_version = "2022"
    db_vendor = "microsoft"
    db_admin_user = "SQLServerAdmin"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "10.10.10.11"
    }
    db_host {
        hostname = "10.10.10.12"
    }
    db_host {
        hostname = "10.10.10.13"
    }
    ssh_key_file = "/home/ubuntu/.ssh/id_rsa"
    ssh_user = "ubuntu"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true
}
```

### Elasticsearch

**Database dropdown value:** "Elasticsearch"

Deploys a new Elasticsearch single-node cluster or high-availability cluster setup. For a high-availability cluster setup, 3 nodes are required with 3 eligible masters and at least 2 data nodes (co-located with the masters). The following vendor and versions are supported for a new deployment:

- Elastic - 7.17, 8.3 and 8.15.

### Default configuration

By default, ClusterControl deploys Elasticsearch with the following configurations:

- For high-availability cluster setup, ClusterControl will configure an NFS server on one of the Elasticsearch nodes and mount the shared filesystem on all data nodes. This is for snapshot backup and restoration.
- For single-node cluster setup, ClusterControl will create a local path for snapshot backup and restoration.

**Warning**

Attention

The minimum memory requirement for an Elasticsearch master node, data node, or master-data node is 1576 MB. ClusterControl will abort the deployment job if this requirement is not met. See Hardware Requirement for Elasticsearch.

### Node configuration fields

- **HTTP port**: Elasticsearch HTTP port. The default is 9200 and not configurable.
- **Transfer port**: Elasticsearch transfer port. The default is 9200 and not configurable.
- **Admin user**: Specify the Elasticsearch admin username.
- **Admin password**: Specify the password for **Admin user**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.
- **Enable SSL encryption**: This is not configurable as Elasticsearch enforces SSL encryption for high-availability cluster setup. If this is a single-node cluster setup, SSL encryption will be disabled automatically.

### Add nodes fields

- **Eligible master**: Specify IP address, hostname, or FQDN of the eligible master node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.

    - **Use as data node** - Toggling on configures the master as a data node. The role will be master-data. If toggled off, the node will be configured as master only. You may add more data nodes under the ***Data nodes*** section.

- **Data nodes**: IP address, hostname, or FQDN of the data node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If no replica nodes, ClusterControl will configure the server as a single-node cluster setup.

### Extra step: Snapshot storage configuration

This step follows **Add nodes**, before **Preview**:

- **Repository name**: Specify the snapshot storage repository name.
- **Storage host**: Host with physical file system storing snapshots and sharing with other cluster's nodes.
- **Default storage location**: Location of the shared filesystem used to store and retrieve snapshots. This location will be registered in the `path.repo` setting on all master and data nodes in the cluster.
- **Configure shared filesystem**: Toggling on lets ClusterControl install and configure the NFS shared filesystem on the chosen ***Storage host*** and ***Default storage location*** as the default cluster's snapshot repository. Toggling off skips this configuration part. Users are responsible for configuring the NFS shared filesystem on the chosen ***Storage host*** and ***Default storage location***. The deployment job will fail if a shared filesystem is not configured.

---

## ClickHouse

**Database dropdown value:** "ClickHouse"

Deploys a new ClickHouse column-oriented OLAP cluster. ClusterControl supports two topologies:

- **Standalone** - a single `clickhouse-server` node with no coordination service. Suitable for testing and development, and can be scaled out later.
- **Replicated cluster** - two or more `clickhouse-server` nodes where data is replicated across all nodes. Coordination is handled by ClickHouse Keeper, either embedded in each data node or running on dedicated keeper nodes. A minimum of 3 keepers (Raft quorum) is recommended for production high availability.

The following vendor and versions are supported for a new deployment:

- ClickHouse - 24.8 (LTS) and 25.3.

### Default configuration

By default, ClusterControl deploys ClickHouse with the following configurations:

- ClickHouse cluster communication is TLS-only. ClusterControl configures secure ports for all client and inter-node traffic: `9440` (native TCP protocol), `8443` (HTTPS API), `9010` (interserver replication), `9281` (Keeper client) and `9234` (Keeper Raft).
- For replicated topologies, ClusterControl configures ClickHouse Keeper (embedded in the data nodes by default) to coordinate replication metadata using the Raft consensus algorithm.

- Prometheus metrics are exposed on port `9363` for monitoring.
- The following packages are installed from the official ClickHouse repository: `clickhouse-common-static`, `clickhouse-server` and `clickhouse-client`.
- An admin user (default `admin`) is created and granted full privileges. For replicated clusters, users are propagated to all nodes via `CREATE USER ... ON CLUSTER`.

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Note**

Note ClusterControl only supports managing x86-64 architecture hosts for ClickHouse. All ClickHouse communication is TLS-only; there is no non-SSL deployment option.

### Node configuration fields

- **Native TCP port**: The secure native TCP port used by `clickhouse-client`. The default is `9440`.
- **HTTPS API port**: The secure HTTPS API port used for health checks and queries. The default is `8443`.
- **Admin user**: Specify the ClickHouse admin username. The default is `admin`.
- **Admin password**: Specify the password for **Admin user**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Enable SSL encryption**: This is not configurable as ClickHouse enforces TLS encryption for all client and inter-node communication.

### Add nodes fields

- **Data nodes**: Specify the IP address or hostname of the ClickHouse data node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Specifying a single data node deploys a standalone cluster; specifying two or more data nodes deploys a replicated cluster with embedded Keeper.
- **Keeper nodes**: (Optional) Specify the IP address or hostname of dedicated ClickHouse Keeper nodes. Press ++enter++ to add the node. If no keeper nodes are specified for a replicated cluster, ClusterControl configures Keeper embedded in each data node. A minimum of 3 keepers is recommended to maintain a Raft quorum.

### CLI examples

### ClusterControl CLI

- Create a standalone ClickHouse 24.8 node, with operating system user “ubuntu”:

```
s9s cluster --create \
    --cluster-type=clickhouse \
    --nodes="clickhouse://10.10.16.4" \
    --vendor=clickhouse \
    --provider-version=24.8 \
    --db-admin='admin' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='ClickHouse 24.8 - Standalone'
```

- Create a three-node ClickHouse 24.8 replicated cluster with embedded Keeper, with operating system user “ubuntu”:

```
s9s cluster --create \
    --cluster-type=clickhouse \
    --nodes=
↪"clickhouse://10.10.16.4;clickhouse://10.10.16.5;clickhouse://10.10.16.6"␣
↪\
    --vendor=clickhouse \
    --provider-version=24.8 \
    --db-admin='admin' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='ClickHouse 24.8 - Replicated Cluster'
```

- Create a ClickHouse 24.8 replicated cluster with dedicated Keeper nodes (2 data nodes + 3 keeper nodes), using operating system user “root” and let the deployment job run in the foreground:

```
s9s cluster --create \
    --cluster-type=clickhouse \
    --nodes=
↪"clickhouse://10.10.16.5;clickhouse://10.10.16.6;clickhouse-keeper://10.10.16.8;cl
↪\
    --vendor=clickhouse \
    --provider-version=24.8 \
    --db-admin='admin' \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='ClickHouse 24.8 - Dedicated Keeper' \
    --wait
```

**Recommended Next Steps**

- *Create Load Balancer*
- *Database Status and Healthchecks*
- *Schedule Backup*
- *Restore External Backup*
- *Alerts and Notifications*

## 2.12 Import Database Cluster

If you already have a running database node or cluster, you can bring it under ClusterControl management using **Import Database Cluster** feature. You may import one database node

### 2.12.1 Support Matrix

The following database cluster types, vendors and topology are supported:

| Database | Vendor | Topology |
|---|---|---|
| MySQL | Percona, Oracle | Standalone, replication |
| MariaDB | MariaDB | Standalone, replication |
| Galera Cluster | MariaDB, Percona | Galera certification-based replication |
| MySQL Group Replication | Oracle | Single-primary, multi-primary |
| PostgreSQL | PostgreSQL, EnterpriseDB | Standalone, streaming replication, logical replication |
| TimescaleDB | TimescaleDB | Standalone, streaming replication |
| MongoDB | MongoDB, Percona, MongoDB Enterprise | Replica set, sharded cluster |
| Redis | Redis, Valkey | Sentinel, cluster |
| Microsoft SQL Server for Linux | Microsoft | Standalone, Always On availability group |
| Elasticsearch | Elastic | Single-node cluster, high availability cluster |
| ClickHouse | ClickHouse | Standalone, replicated cluster |

### 2.12.2 Prerequisites

There are prerequisites that need to be fulfilled prior to the deployment for all database clusters:

- Make sure the target database nodes are running on a supported architecture platform and operating system. See *Hardware* and *Operating System*.
- Passwordless SSH (SSH using key-based authentication) is configured from the ClusterControl node to all database nodes. See *SSH Key-based Authentication*.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*.
- The target cluster must be in healthy state and not in a degraded state. For example, if you have a three-node Galera cluster, all nodes must be alive, accessible, and in sync.

**Note**

Note For more details, refer to the *Requirements* section. Each time you import an existing cluster or server, ClusterControl will trigger a job under ***ClusterControl GUI → Activity Center → Jobs***. You can see the progress and status on this page. Click on **…** of the importing job and click **Details**. A window will also appear with messages showing the progress.

### 2.12.3 Common Wizard Steps

The Import Database Cluster wizard follows the same flow and step order for every database type listed in the *Support Matrix* above. Only the **Node configuration** and **Add nodes** steps have database-specific fields. These differences are documented per database type in *Database-Specific Settings* below.

#### ClusterControl GUI

**Opening the wizard**

1. Go to ***ClusterControl GUI → Deploy a cluster → Import a database cluster*** and under the **Database** dropdown, choose your target database type (see the dropdown label listed at the start of each entry in *Database-Specific Settings*).

---

**Step 1: Cluster details**

2. Under **Cluster details**, specify the cluster details you want to assign:

   - **Name**: Optional. This is the name of your cluster, labeled **Name your cluster**. Once the import is done, ClusterControl will use this as the cluster's registry name.
   - **Tags**: Add tags to search or group your database clusters.

   !!! note "MongoDB Replica Set and MongoDB Sharded Cluster only"

   ```
   This step also has a **Vendor** field, under **Vendor and version**.␣
   ↪This is required and identifies which vendor's binaries the existing␣
   ↪cluster is running: choose from **Percona**, **MongoDB**, or␣
   ↪**MongoDB Enterprise**.
   ```

   Click **Continue**.

---

**Step 2: SSH configuration**

3. Under the **SSH configuration** section, specify the SSH credentials that ClusterControl should use to connect to the database nodes:

   - **SSH user**: The SSH user that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH user.
   - **SSH user key path**: The SSH private key path that ClusterControl will use to perform SSH authentication to the database node. Relative path is not supported. The SSH private key must be physically secured and exist on the ClusterControl node.
   - **SSH port**: The SSH port that ClusterControl will use to perform SSH to the database node. ClusterControl assumes that all database nodes are using the same SSH port.

- **SSH sudo password**: The sudo password if the **SSH user** requires a password for privilege escalation.
- **SSH sudo / OS elevation command**: The OS elevation command (`sudo`, `doas`, `pbrun`) that ClusterControl will use for a non-root account.

Click **Continue** to proceed to the next step.

---

**Step 3: Node configuration**

4. Under the **Node configuration** section, specify the database credentials and configurations that ClusterControl shall use when importing the cluster. The exact fields shown depend on the database type; see *Database-Specific Settings* for the full list for your chosen database.

   Click **Continue** to proceed to the next step.

---

**Step 4: Add nodes**

5. Under the **Add nodes** section, specify your existing target database nodes. Since this is an existing database cluster that is already set up, ClusterControl will identify and scan the topology of your database cluster automatically. The exact fields depend on the database type; see *Database-Specific Settings* for the full list for your chosen database.

   **Note**

   Note

### You can only proceed to the next step if all of the specified nodes are reachable (shown in green). :::

**Finishing the wizard**

6. Click **Continue** to proceed to the **Preview** page. In this section, you can see the summary of your import and if everything is correct, you may proceed by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The settings will be kept until you exit the wizard.
7. ClusterControl will trigger an import job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

Every database type is imported with the same `s9s cluster --register` command used for *creating* a cluster, just replacing `--create` with `--register`. Only `--cluster-type`, `--nodes`, `--vendor`, and a handful of type-specific flags change between database types. See *Database-Specific Settings* for a full working example per database type.

Importing via CLI is not currently implemented for **SQL Server** or **Elasticsearch** — use the GUI for those database types.

### 2.12.4 Database-Specific Settings

#### MySQL Replication

**Database dropdown value:** "MySQL Replication"

ClusterControl can manage and monitor existing MySQL and MariaDB servers, whether standalone or in a replication setup. A minimum of two nodes is required for MySQL/MariaDB replication. If only one database IP address or hostname is provided, ClusterControl will deploy it as a standalone MySQL/MariaDB server with binary log enabled.

When adding individual hosts, listing them together will place them in the same server group. For this to work, ClusterControl expects all instances within a group to use the same MySQL root password. It will also automatically try to identify each server's role (primary, replica, multi, or standalone).

When importing an existing MySQL Replication, ClusterControl will do the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, replica, multi, standalone).
- Pull the configuration files.
- Generate the authentication key and register the nodes into ClusterControl.

#### Default configuration

By default, ClusterControl deploys MySQL/MariaDB replication with the following configurations:

- MySQL GTID with `log_slave_updates` enabled (MySQL and Percona only).
- MariaDB GTID with `log_slave_updates` enabled (MariaDB only).
- All database nodes will be configured with `read_only=ON` and `super_read_only=ON` (if supported) for protection against accidental write. The chosen primary will be promoted by disabling the read-only in the runtime.
- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- ClusterControl will create and grant necessary privileges for the following database users:

```
| Database user      | Purpose                                          |
| ------------------ | ------------------------------------------------ |
| `cmon`             | Management and automation.                       |
| `cmonexporter`     | Prometheus exporter for database monitoring.     |
| `cmonagent`        | ClusterControl query monitoring agent.           |
| `cmon_replication` | MySQL/MariaDB replication.                       |
| `backupuser`       | Backup and restore management.                   |
```

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

ClusterControl sets `read_only=ON` on all slaves but a privileged user (SUPER) can still write to a slave (except for MySQL versions that support `super_read_only`).

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically.
- **Admin/Root user**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPER privilege and the GRANT option for localhost only.
- **Admin/Root password**: The password for **Admin/Root user**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **information_schema queries**: Off by default. When enabled, this allows ClusterControl to query the `information_schema` schema to collect metrics from your MySQL database nodes.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node, for example one whose database process was unexpectedly terminated.

### Add nodes fields

- **Node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.

### CLI example

```
s9s cluster --register \
    --cluster-type=mysqlreplication \
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=oracle \
    --provider-version=8.0 \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='PROD - MySQL Replication 8.0' \
    --wait \
    --log
```

Regardless of node order, ClusterControl identifies the primary node(s) through the `read_only` and `super_read_only` variables.

### MySQL Galera

**Database dropdown value:** "MySQL Galera"

ClusterControl can manage and monitor existing Galera Clusters, assuming all instances in the group use the same MySQL root password. Adding *asynchronous slaves* and *adding load balancers* connected to the Galera Cluster should be specified and imported separately after the cluster is imported into ClusterControl.

When importing an existing Galera Cluster, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Detect the remaining nodes in the cluster, only if Automatic node discovery is ON.
- Pull the configuration files.
- Generate the authentication key and register the nodes into ClusterControl.

A minimal setup is comprised of one Galera node (no high availability, but this can later be scaled with more nodes). However, a minimum of 3 nodes is recommended for high availability. Garbd (an arbitrator) can be added later after the deployment completes.

The following vendors and versions are supported for importing your existing cluster:

- Percona XtraDB Cluster - 8.0 and 8.4.
- Percona XtraDB Cluster Pro - 8.4.
- MariaDB Cluster - 10.4, 10.5, 10.6, 10.11 (LTS) and 11.4 (LTS).

### Default configuration

By default, ClusterControl deploys Galera Cluster with the following configurations:

- Use `xtrabackup-v2` or `mariabackup` (depending on the vendor chosen) for `wsrep_sst_method`.
- Binary logging is enabled.
- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | MySQL/MariaDB replication. |
| `backupuser` | Galera SST, backup and restore management. |

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes. If the path does not exist, ClusterControl will create and configure it automatically.
- **Admin/Root user**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPER privilege and the GRANT option for localhost only.
- **Admin/Root password**: The password for **Admin/Root user**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **information_schema queries**: Off by default. When enabled, this allows ClusterControl to query the `information_schema` schema to collect metrics from your MySQL database nodes.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node, for example one whose database process was unexpectedly terminated.
- **Automatic node discovery**: On by default. Set this to off if `wsrep_node_incoming_address=AUTO` for all of your primary Galera nodes, or if your ClusterControl node uses different IPs/hostnames to connect to Galera nodes for inter-node connectivity than the ones you pass here.

### Add nodes fields

- **Node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.

### CLI examples

- Import a three-node Percona XtraDB Cluster 8.0, with operating system user "ubuntu":

```
s9s cluster --register \
    --cluster-type=galera \
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=percona \
    --provider-version=8.0 \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='PROD - MySQL Replication 8.0' \
    --wait \
    --log
```

- Import a three-node MariaDB Cluster 11.4:

```
s9s cluster --register \
    --cluster-type=galera \
    --nodes="192.168.5.11;192.168.5.12;192.168.5.13" \
    --vendor=mariadb \
    --provider-version=11.4 \
    --db-admin-passwd='Str0ngpa$$w0rd' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='System A - MariaDB Cluster 11.4' \
    --wait \
    --log
```

---

### PostgreSQL Streaming

**Database dropdown value:** "PostgreSQL Streaming"

ClusterControl can manage and monitor existing PostgreSQL or TimescaleDB clusters running streaming replication version 12 or later. When importing hosts, those listed together will form a server group in the user interface. It is expected that all instances within the same group use the same database administrator password.

A minimum of two nodes is required for PostgreSQL streaming replication. If only one database IP address or hostname is provided, ClusterControl will deploy it as a standalone PostgreSQL server. The following vendors and versions are supported for importing your existing cluster:

- PostgreSQL from postgresql.org repository - 14, 15, 16, 17 and 18.
- PostgreSQL from EDB repository - 14, 15, 16 and 17 (requires a valid EDB repository token).

When importing an existing PostgreSQL or TimescaleDB streaming replication cluster, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, replica, TimescaleDB extension).
- Pull the configuration files.
- Generate the authentication key and register the nodes into ClusterControl.

### Default configuration

By default, ClusterControl deploys PostgreSQL instances with the following configurations:

- Configure and load the `pg_stat_statements` module.
- The WAL level is set to `replica`.
- All replica will be set with `hot_standby`.
- Configure the `cluster_name` value.
- ClusterControl will configure the PostgreSQL instance with SSL encryption for client-server connections.

- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | PostgreSQL streaming replication. |
| `backupuser` | Backup and restore management. |

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **User**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for single-primary replication.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If you have more than one replica node, make sure to add each of them under this field.

### CLI examples

- Import a three-node PostgreSQL 16 streaming replication, with operating system user "ubuntu" (the first node is the primary), with tags `production` and `postgres`, waiting for and logging the job:

```
s9s cluster --register \
    --cluster-type=postgresql \
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=postgresql \
    --provider-version=16 \
    --db-admin='postgres' \
    --db-admin-passwd='mYpa$$word' \
```

(continues on next page)

(continued from previous page)

```
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --with-tags='production;postgres' \
    --cluster-name='PostgreSQL 16 - Streaming Replication' \
    --wait --log
```

- Import a four-node PostgreSQL EDB 16 streaming replication, using operating system user "root" (the first node is the primary) and let the deployment job run in the foreground:

```
s9s cluster --register \
    --cluster-type=postgresql \
    --nodes="192.168.5.11;192.168.5.12;192.168.5.13;192.168.5.14" \
    --vendor=enterprisedb \
    --provider-version=11.4 \
    --db-admin='mydbuser' \
    --db-admin-passwd='mydbPassw0rd' \
    --enterprise-token='XXXXXXXXXXXXXXXXXXXXXXXXX' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --cluster-name='System A - PostgreSQL EDB 16 - Streaming␣
↪Replication' \
    --wait
```

---

### TimescaleDB

**Database dropdown value:** "TimescaleDB"

Imports an existing TimescaleDB standalone or streaming replication cluster. A minimum of two nodes is required for TimescaleDB streaming replication. If only one database IP address or hostname is provided, ClusterControl will deploy it as a standalone TimescaleDB server. The following vendors and versions are supported for importing your existing cluster:

- TimescaleDB - 14, 15, 16, 17 and 18.

### Default configuration

By default, ClusterControl deploys TimescaleDB with the following configurations:

- Configure and load the `pg_stat_statements` module and `timescaledb` extension.
- The WAL level is set to `replica`.
- All replica will be set with `hot_standby`.
- Configure the `cluster_name` value.
- ClusterControl will configure the TimescaleDB instance with SSL encryption for client-server connections.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | PostgreSQL streaming replication. |
| `backupuser` | Backup and restore management. |

**Note**

Note You can also import an existing PostgreSQL cluster and convert it to TimescaleDB at a later stage. However, this action is irreversible and ClusterControl will treat the cluster as TimescaleDB from that point onward.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **User**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Only one primary node is allowed for single-primary replication.
- **Replica nodes**: Specify the IP address or hostname of the replica database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If you have more than one replica node, make sure to add each of them under this field.

### CLI examples

- Import a three-node TimescaleDB 16 streaming replication, with operating system user "ubuntu" (the first node is the primary) with tags `production` and `timescaledb`:

```
s9s cluster --register \
    --cluster-type=postgresql \
```

(continues on next page)

(continued from previous page)

```
    --nodes="10.10.10.11;10.10.10.12;10.10.10.13" \
    --vendor=postgresql \
    --provider-version=16 \
    --db-admin='mydbuser' \
    --db-admin-passwd='mydbPassw0rd' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --with-tags='production;timescaledb' \
    --cluster-name='TimescaleDB 16 - Streaming Replication' \
    --with-timescaledb
```

- Import a standalone TimescaleDB 15, with operating system user "root", tags `test` and `timescaledb`, waiting until the job finishes:

```
s9s cluster --register \
    --cluster-type=postgresql \
    --nodes="192.168.99.11" \
    --vendor=postgresql \
    --provider-version=15 \
    --db-admin='postgres' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --with-tags='test;timescaledb' \
    --cluster-name='TimescaleDB 16 - standalone' \
    --with-timescaledb \
    --wait
```

---

### MongoDB Replica Set

**Database dropdown value:** "MongoDB ReplicaSet"

ClusterControl is able to manage and monitor an existing MongoDB or Percona Server for MongoDB replica set.

When importing an existing MongoDB replica set, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, secondary, arbiter).
- Pull the configuration files.
- Generate the authentication key and register the nodes into ClusterControl.

The database cluster will be automatically added to ClusterControl once the import completes successfully. If only one node is specified, ClusterControl will import it as a standalone MongoDB node. The following vendors and versions are supported for importing your existing cluster:

- MongoDB Community - 7.0 and 8.0
- MongoDB Enterprise - 7.0 and 8.0

- Percona Server for MongoDB - 7.0 and 8.0

**Warning**

Attention

If SSL/TLS is required, ClusterControl only supports a proper CAFile configuration, as shown in the MongoDB documentation, and does not support the `--allowInvalidCertificates` flag.

### Default configuration

By default, ClusterControl deploys MongoDB Replica Set members with the following configurations:

- Configure `setParameter.enableLocalhostAuthBypass: true` inside MongoDB configuration file.
- ClusterControl will create and grant necessary roles for additional MongoDB users - `admin.cmon_backup` for backup and restore purposes and `admin.cmonexporter` for query monitoring.

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

It is possible to deploy only 2 MongoDB nodes (without an arbiter). The caveat of this approach is no automatic failover. If the primary node goes down then manual failover is required to make the other server run as primary. Automatic failover works fine with 3 nodes and more.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **User**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.

### Add nodes fields

- **Node**: Specify the IP address or hostname of the primary database node only. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. The secondary/replica nodes and arbiters are auto-detected and imported along with the primary.

### CLI examples

- Import a 2-node MongoDB replica set using Percona binaries. Only the primary node needs to be specified — the replica (in this example, at 192.168.40.191) is auto-detected and imported alongside it. This example uses `ubuntu` as the OS username and `ed25519` as the SSH key algorithm, and waits for and prints the logs of the job:

```
s9s cluster --register \
    --cluster-type=mongodb \
    --nodes="192.168.40.190" \
    --vendor=percona \
    --provider-version='7.0' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
    --db-admin='mydbuser' \
    --db-admin-passwd='mydbPassw0rd' \
    --cluster-name='MongoDB Percona Server ReplicaSet 7.0' \
    --wait --log
```

- Import a three-node MongoDB Enterprise replica set 7.0, with operating system user "root", running in the foreground:

```
s9s cluster --register \
    --cluster-type=mongodbenterprise \
    --nodes="192.168.40.155" \
    --vendor=mongodbenterprise \
    --provider-version='7.0' \
    --os-user=root \
    --os-key-file=/root/.ssh/id_rsa \
    --db-admin='mydbuser' \
    --db-admin-passwd='mydbPassw0rd' \
    --cluster-name='MongoDB Enterprise ReplicaSet 7.0'
```

---

### MongoDB Sharded Cluster

**Database dropdown value:** "MongoDB Shards"

ClusterControl is able to manage and monitor an existing MongoDB, Percona Server for MongoDB, or MongoDB Enterprise 4.x, 5.x, 6.x, 7.x, or 8.x sharded cluster.

When importing an existing MongoDB sharded cluster, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (mongos, config server, shard server - primary, secondary, arbiter).
- Pull the configuration files.
- Generate the authentication key and register the nodes into ClusterControl.

The following vendors and versions are supported for importing your existing cluster:

- MongoDB Enterprise - 7.0 and 8.0
- MongoDB Community - 7.0 and 8.0
- Percona Server for MongoDB - 7.0 and 8.0

### Default configuration

By default, ClusterControl deploys MongoDB Sharded Cluster with the following configurations:

- Configure `setParameter.enableLocalhostAuthBypass: true` inside MongoDB configuration file.
- ClusterControl will create and grant necessary privileges for an additional MongoDB user - `admin.cmon_backup` for backup and restore purposes.

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

MongoDB Sharded Cluster does not support the `mongodump` backup method. Users will be asked to install Percona Backup for MongoDB (PBM) when creating or scheduling a backup for this cluster type after the deployment completes. Note that PBM requires shared remote backup storage (e.g., NFS) accessible on all MongoDB nodes.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **User**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **User**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.

### Add nodes fields

- **Node router server**: Specify the IP address or hostname of a mongos (router) node only. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Specify only the MongoDB router servers (mongos); ClusterControl discovers the config servers and shard members automatically. If you have multiple router servers, add each of them here.

### CLI example

- Import a MongoDB sharded cluster using Percona binaries, consisting of 3 router (mongos) nodes, using `ubuntu` as the OS username and `ed25519` as the SSH key algorithm, waiting for and printing the logs of the job:

```
s9s cluster --register \
    --cluster-type=mongodb \
    --nodes=
↪"mongos://192.168.1.11;mongos://192.168.1.12;mongos://192.168.1.12;"␣
↪\
    --vendor=percona \
    --provider-version='7.0' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
    --db-admin='mydbuser' \
    --db-admin-passwd='mydbPassw0rd' \
    --cluster-name='MongoDB Percona Server Shards 7.0' \
    --wait --log
```

---

### Valkey Sentinel

**Database dropdown value:** "Valkey Sentinel"

Imports existing Valkey instances with Sentinel, for v7 and v8. When importing existing Valkey instances, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, replica, sentinel).

**Warning**

Attention

ClusterControl does not support importing Valkey instances without authentication. Authentication must be enabled and the provided admin username and password must have `~* +@all` (all keys, all commands) privilege.

The following vendors and versions are supported for importing your existing cluster:

- Valkey Sentinel - 7 and 8
- Canonical - 7

For a very minimal setup (suitable for testing and experimenting), it is possible to import a single-node Valkey instance with Sentinel (both services co-located on the same node), and later scale out with more Valkey database and Sentinel nodes.

### Default configuration

By default, ClusterControl deploys Valkey instances with the following configurations:

- Valkey Sentinel (default port is 26379) will be co-located with the Valkey instances (default port is 6379).
- ClusterControl will configure the Valkey instance with `appendonly` enabled.
- ClusterControl will secure the instance with authentication enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the database nodes, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Valkey being killed by OOM.

!!! Tips

```
Valkey Sentinel requires 3 nodes for automatic primary promotion. Sentinel␣
↪can be co-located on the ClusterControl server if you want to import a␣
↪two-node Valkey replication cluster (Sentinel will be co-located on each␣
↪database instance).
```

### Node configuration fields

- **Valkey port**: The database server port that ClusterControl shall configure on all Valkey database nodes.
- **Valkey Sentinel port**: The database server port that ClusterControl shall configure on all Valkey Sentinel nodes.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.
- **Password**: Under the **Authentication** pane, the primary node password used for client/replica authentication.
- **Replication user**: Under the **Authentication** pane, the replication user of your Valkey replication cluster.
- **Replication password**: Under the **Authentication** pane, the replication password of your Valkey replication cluster.
- **Sentinel password**: The password for your Sentinel user `sentinel-user`.
- **Use existing certificates**: Specify cmon's certificates directory to use on the imported cluster. If SSL configuration is not completed, scale-up operations may fail.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.
- **Replica nodes**: Specify the IP address or hostname of a replica node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. If you have more than one replica node, make sure to add each of them under this field.

### CLI example

- Import a three-node Valkey Sentinel cluster, using `ubuntu` as the OS username, `sjl2l5wi` as the Sentinel password, and `ed25519` as the SSH key algorithm, waiting for and printing the logs of the job:

```
s9s cluster --register --cluster-type=valkey \
    --nodes="valkey://192.168.40.190:6379;␣
↪valkey://192.168.40.191:6379;valkey://192.168.40.192:6379; \
            ␣
↪valkey-sentinel://192.168.40.190:26379;valkey-sentinel://192.168.40.191:26379;␣
↪\
             valkey-sentinel://192.168.40.192:26379;" \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
    --sentinel-passwd sjl2l5wi \
    --vendor=valkey \
    --provider-version=8 \
    --cluster-name="My Valkey Sentinel v8" \
    --log --wait
```

---

### Valkey Cluster

**Database dropdown value:** “Valkey Cluster”

Imports an existing Valkey Cluster, for v7 and v8. When importing existing Valkey instances, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, replica).

**Warning**

Attention

ClusterControl does not support importing Valkey instances without authentication. Authentication must be enabled and the provided admin username and password must have `~* +@all` (all keys, all commands) privilege.

The following vendors and versions are supported for importing your existing cluster:

- Valkey Cluster - 7 and 8
- Canonical - 7

### Default configuration

By default, ClusterControl deploys Valkey cluster instances with the following configurations:

- ClusterControl will configure the Valkey instances with `appendonly` enabled.
- ClusterControl will secure the instance with AUTH enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the cluster, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Valkey being killed by OOM.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Username**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **Username**.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node.

### Add nodes fields

- **Cluster node**: Specify the IP address or hostname of any node in the cluster. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Specify one node and ClusterControl will discover the rest of the cluster members.

### CLI example

- Import a Valkey Cluster by specifying a single endpoint node. Only the primary needs to be specified — replicas (in this example, at 192.168.40.199) are auto-detected and imported alongside it. This example uses `ubuntu` as the OS username and `ed25519` as the SSH key algorithm, and waits for and prints the logs of the job:

```
s9s cluster --register \
    --cluster-type=valkey_sharded \
    --nodes="valkey://192.168.40.198;" \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
    --vendor=valkey \
    --provider-version=8 \
    --cluster-name="My Valkey Sharded Cluster" \
    --log --wait
```

---

### Redis Sentinel

**Database dropdown value:** "Redis Sentinel"

Imports existing Redis instances with Sentinel, for v7 and v8. When importing existing Redis instances, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, replica, sentinel).

**Warning**

Attention

ClusterControl does not support importing Redis instances without authentication. Authentication must be enabled and the provided admin username and password must have `~* +@all` (all keys, all commands) privilege.

The following vendors and versions are supported for importing your existing cluster:

- Redis Sentinel - 6, 7 and 8

**Info**

ClusterControl and Severalnines do not use or reproduce the official Redis® logo, which is a registered trademark of Redis Ltd. To comply with Redis trademark and branding guidelines, we have introduced our own distinct branding for Redis-related features within ClusterControl. This logo and name are original creations of Severalnines and are used solely as internal branding to represent Redis.

### Default configuration

By default, ClusterControl deploys Redis instances with the following configurations:

- Redis Sentinel (default port is 26379) will be co-located with the Redis instances (default port is 6379).
- ClusterControl will configure the Redis instance with `appendonly` enabled.
- ClusterControl will secure the instance with authentication enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the database nodes, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Redis being killed by OOM.

!!! Tips

```
Redis Sentinel requires 3 nodes for automatic primary promotion. Sentinel␣
↪can be co-located on the ClusterControl server if you want to import a␣
↪two-node Redis replication cluster (Sentinel will be co-located on each␣
↪database instance).
```

### Node configuration fields

- **Redis port**: The database server port that ClusterControl shall configure on all Redis database nodes.
- **Redis Sentinel port**: The database server port that ClusterControl shall configure on all Redis Sentinel nodes.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part. Choose "Create new repository" to let ClusterControl create a local repository for the chosen vendor on the ClusterControl node.
- **Password**: Under the **Authentication** pane, the primary node password used for client/replica authentication.
- **Replication user**: Under the **Authentication** pane, the replication user of your Redis replication cluster.
- **Replication password**: Under the **Authentication** pane, the replication password of your Redis replication cluster.
- **Sentinel password**: The password for your Sentinel user `sentinel-user`.
- **Use existing certificates**: Specify cmon's certificates directory to use on the imported cluster. If SSL configuration is not completed, scale-up operations may fail.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.
- **Replica nodes**: Specify the IP address or hostname of a replica node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. If you have more than one replica node, make sure to add each of them under this field.

### CLI example

- Import a three-node Redis Sentinel cluster, using `ubuntu` as the OS username, `sjl2l5wi` as the Sentinel password, and `ed25519` as the SSH key algorithm, waiting for and printing the logs of the job:

```
s9s cluster --register --cluster-type=redis \
    --nodes="redis://192.168.40.190:6379;␣
↪redis://192.168.40.191:6379;redis://192.168.40.192:6379; \
            ␣
↪redis-sentinel://192.168.40.190:26379;redis-sentinel://192.168.40.191:26379;␣
↪\
             redis-sentinel://192.168.40.192:26379;" \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
    --sentinel-passwd sjl2l5wi \
    --vendor=redis \
    --provider-version=8 \
    --cluster-name="My Redis Sentinel Cluster" \
    --log --wait
```

---

### Redis Cluster

**Database dropdown value:** “Redis Cluster”

Imports an existing Redis Cluster, for v7 and v8. When importing existing Redis instances, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, replica).

**Warning**

Attention

ClusterControl does not support importing Redis instances without authentication. Authentication must be enabled and the provided admin username and password must have `~* +@all` (all keys, all commands) privilege.

The following vendors and versions are supported for importing your existing cluster:

- Redis Cluster - 6, 7 and 8
- Canonical - 7

**Info**

ClusterControl and Severalnines do not use or reproduce the official Redis® logo, which is a registered trademark of Redis Ltd. To comply with Redis trademark and branding guidelines, we have introduced our own distinct branding for Redis-related features within ClusterControl. This logo and name are original creations of Severalnines and are used solely as internal branding to represent Redis.

### Default configuration

By default, ClusterControl deploys Redis cluster instances with the following configurations:

- ClusterControl will configure the Redis instances with `appendonly` enabled.
- ClusterControl will secure the instance with AUTH enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the cluster, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Redis being killed by OOM.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Username**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPERUSER role and allowed for localhost only.
- **Password**: The password for **Username**.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node.

### Add nodes fields

- **Cluster node**: Specify the IP address or hostname of any node in the cluster. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Specify one node and ClusterControl will discover the rest of the cluster members.

### CLI example

- Import a Redis Cluster by specifying a single endpoint node. Only the primary needs to be specified — replicas (in this example, at 192.168.40.189) are auto-detected and imported alongside it. This example uses `ubuntu` as the OS username and `ed25519` as the SSH key algorithm, and waits for and prints the logs of the job:

```
s9s cluster --register \
    --cluster-type=redis_sharded \
    --nodes="redis://192.168.40.188;" \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_ed25519 \
    --vendor=redis \
    --provider-version=8 \
    --cluster-name="My Redis Sharded Cluster" \
    --log --wait
```

---

## Microsoft SQL Server

**Database dropdown value:** “SQL Server”

Imports an existing Microsoft SQL Server for Linux instance or Always On availability group - 2019, 2022 and 2025. When importing existing SQL Server instances, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (primary, replica).

The following vendors and versions are supported:

- Microsoft SQL Server for Linux - 2019, 2022 and 2025

**Warning**

Attention

A Microsoft SQL Server database node must have at least 2000 MB of memory. ClusterControl will abort the import job if this requirement is not met.

### Default configuration

By default, ClusterControl manages SQL Server with the following configurations:

- All database nodes must use fully qualified domain name (FQDN). IP address is not supported.
- Minimum available RAM is 1800 MB per database node.
- Enforces the SQL Server user's password policy as shown here.
- At the moment, ClusterControl only supports AlwaysOn with asynchronous-commit mode, where it does not wait for any secondary replica to write incoming transaction log records to disk.

### Node configuration fields

- **Admin username**: The username of your existing admin user.
- **Admin password**: The password for the admin user.
- **Server port**: This field is disabled during import; it is shown only as a reference for the port already in use.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node.

### Add nodes fields

- **Primary node**: Specify the IP address or hostname of the primary database node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic.
- **Replica nodes**: Specify the IP address or hostname of a replica node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify zero or more replica nodes. If you have more than one replica node, make sure to add each of them under this field.

### CLI example

Importing via CLI is not currently implemented for SQL Server; use the GUI instead.

### Elasticsearch

**Database dropdown value:** "Elasticsearch"

Imports an existing Elasticsearch cluster, for 7.x and 8.x. When importing existing Elasticsearch instances, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the database role of each node (master, data).

> **Warning**
>
> Attention
>
> ClusterControl does not support importing Elasticsearch instances without authentication. Authentication must be enabled, and a valid sysadmin username and password must be provided.

The following vendor and versions are supported:

- Elastic - 7.17, 8.3 and 8.15

### Default configuration

By default, ClusterControl manages Elasticsearch with the following configurations:

- For a high-availability cluster setup, ClusterControl will configure an NFS server on one of the Elasticsearch nodes and mount the shared filesystem on all data nodes. This is for snapshot backup and restoration.
- For a single-node cluster setup, ClusterControl will create a local path for snapshot backup and restoration.

> **Warning**
>
> Attention
>
> The minimum memory requirement for an Elasticsearch master node, data node, or master-data node is 1576 MB. ClusterControl will abort the import job if this requirement is not met. See Hardware Requirement for Elasticsearch.

### Node configuration fields

- **HTTP port**: This field is disabled during import; it is shown only as a reference for the port already in use.
- **Certificates password**: If specified, the password for a preexisting CA private key.
- **Admin username**: The sysadmin username used by your Elasticsearch cluster.
- **Admin password**: The sysadmin password used by your Elasticsearch cluster.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always

install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.

- **Use existing certificates**: Specify cmon's certificates directory to use on the imported cluster. If left blank, ClusterControl searches the configuration folder `/etc/elasticsearch` or the default generation folder `/usr/share/elasticsearch` for certificate files (such as `elastic-certificates.p12`, `elastic-stack-ca.p12`, and `elasticsearch-ssl-http.zip`) on the node, to configure certificates on any new nodes added to the cluster. If SSL configuration is not completed, scale-up operations may fail.
- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node.

### Add nodes fields

- **Cluster node**: Specify the IP address or hostname of the primary or master-data node. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. Specify one node and ClusterControl will discover the rest of the cluster members; the specified node is also used to search for certificate files.

### CLI example

Importing via CLI is not currently implemented for Elasticsearch; use the GUI instead.

---

## MySQL Group Replication

**Database dropdown value:** "MySQL Group Replication"

ClusterControl can manage and monitor existing MySQL Group Replication clusters. A minimum of three nodes is required for MySQL Group Replication to achieve high availability and automatic failover. The cluster can be configured in either single-primary mode (one primary and multiple secondaries) or multi-primary mode (all nodes accept writes).

When importing an existing MySQL Group Replication cluster, ClusterControl will do the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Pull the configuration files.
- Detect the remaining nodes in the cluster, only if Automatic node discovery is ON.
- Register the nodes into ClusterControl.

The following vendors and versions are supported for importing your existing cluster:

- Oracle MySQL 8.0 and 8.4

### Default configuration

By default, ClusterControl deploys MySQL Group Replication with the following configurations:

- MySQL GTID is enabled with `log_slave_updates` enabled.
- Group Replication plugin is loaded and configured.
- Binary logging is enabled.
- All database nodes in multi-primary mode will be configured with `read_only=OFF` to allow writes on all nodes.
- The generated backup credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- ClusterControl will create and grant necessary privileges for the following database users:

```
| Database user      | Purpose                                          |
| ------------------ | ------------------------------------------------ |
| `cmon`             | Management and automation.                       |
| `cmonexporter`     | Prometheus exporter for database monitoring.     |
| `rpl_user`         | MySQL Group Replication.                         |
| `backupuser`       | Backup and restore management.                   |
```

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

MySQL Group Replication requires all nodes to be reachable by each other. Ensure that network connectivity between all nodes is properly configured before importing the cluster.

### Node configuration fields

- **Server port**: The database server port that ClusterControl shall configure on all database nodes.
- **Server data directory**: The database server data directory path that ClusterControl shall configure on all database nodes.
- **Admin/Root user**: The database admin username that ClusterControl shall configure on all database nodes. This user will be granted global SUPER privilege and the GRANT option for localhost only.
- **Admin/Root password**: The password for **Admin/Root user**.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.
- **Replication user/Replication password**: The credentials configured for replication.
- **information_schema queries**: Off by default. When enabled, this allows ClusterControl to query the `information_schema` schema to collect metrics from your MySQL database nodes.

- **Cluster auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a degraded or failing cluster.
- **Node auto-recovery**: Off by default. When enabled, ClusterControl will automatically attempt to recover a failing node, for example one whose database process was unexpectedly terminated.
- **Automatic node discovery**: On by default. Set it to off if you want to specify each of the IPs/hostnames that ClusterControl should use to connect to the database nodes yourself.

### Add nodes fields

- **Node**: Specify the IP address or hostname of any database node in the group. Press ++enter++ to add the node, where ClusterControl will perform a check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify just one node; ClusterControl automatically discovers the remaining nodes in the Group Replication cluster.

### CLI example

- Import an Oracle MySQL Group Replication 8.4 cluster, with operating system user `ubuntu`:

```
s9s cluster --register \
    --cluster-type=group_replication \
    --nodes="10.10.10.10,10.10.10.11,10.10.10.12" \
    --vendor=oracle \
    --provider-version=8.4 \
    --db-admin-passwd="mYpa$$word" \
    --repl-user="rpl_user" \
    --repl-passwd="mYrepLpa$$word" \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name="MySQL Group Replication 8.4" \
    --wait \
    --log
```

---

### ClickHouse

**Database dropdown value:** “ClickHouse”

ClusterControl can manage and monitor existing ClickHouse clusters, whether standalone or replicated. When importing an existing ClickHouse cluster, ClusterControl will perform the following:

- Verify SSH connectivity to all nodes.
- Detect the host environment and operating system.
- Discover the cluster topology and the role of each node (data, keeper) from `system.clusters`.
- Register the nodes into ClusterControl without modifying any existing configuration files.

The import is non-invasive: ClusterControl does not change the configuration files of the existing ClickHouse cluster. You may specify a single node as an endpoint and ClusterControl will automatically discover the rest of the cluster topology.

The following vendor and versions are supported for importing your existing cluster:

- ClickHouse - 24.8 (LTS) and 25.3

**Warning**

Attention

ClusterControl does not support importing ClickHouse instances without authentication. Authentication must be enabled and a valid admin username and password provided. All ClickHouse communication is TLS-only.

### Default configuration

By default, ClusterControl manages ClickHouse with the following configurations:

- ClickHouse cluster communication is TLS-only, using the secure ports: `9440` (native TCP protocol), `8443` (HTTPS API), `9010` (interserver replication), `9281` (Keeper client) and `9234` (Keeper Raft).
- Prometheus metrics are collected from port `9363`.
- An admin user (default `admin`) is used to connect to and monitor the cluster.

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the import. See *Configuration Template* for details.

### Node configuration fields

- **Native TCP port**: The secure native TCP port used by `clickhouse-client`. The default is `9440`.
- **HTTPS API port**: The secure HTTPS API port used for health checks and queries. The default is `8443`.
- **Admin user**: The admin username used by your ClickHouse cluster. The default is `admin`.
- **Admin password**: The admin password used by your ClickHouse cluster.
- **Repository**: Choosing "Use vendor repositories" (default) lets ClusterControl provision software by setting up and using the database vendor's preferred software repository. ClusterControl will always install the latest version of what is provided at that moment. Choose "Do not setup vendor repositories" if you have a special configuration from the vendor (commonly for enterprise databases); ClusterControl will then skip the repository configuration part.

### Add nodes fields

- **Node**: Specify the IP address or hostname of any ClickHouse data node in the cluster. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication. If the target node has more than one network interface, you will be able to select or enter a separate IP address to be used only for database traffic. You can specify just one node; ClusterControl automatically discovers the remaining data and keeper nodes in the ClickHouse cluster.

### CLI examples

- Import a standalone ClickHouse 24.8 node, with operating system user “ubuntu”:

```
s9s cluster --register \
    --cluster-type=clickhouse \
    --nodes="clickhouse://10.10.16.4" \
    --vendor=clickhouse \
    --provider-version=24.8 \
    --db-admin='admin' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='ClickHouse 24.8 - Standalone' \
    --wait \
    --log
```

- Import an existing ClickHouse 24.8 replicated cluster by specifying a single endpoint node. ClusterControl will discover the rest of the topology (data and keeper nodes) automatically:

```
s9s cluster --register \
    --cluster-type=clickhouse \
    --nodes="clickhouse://10.10.16.4" \
    --vendor=clickhouse \
    --provider-version=24.8 \
    --db-admin='admin' \
    --db-admin-passwd='mYpa$$word' \
    --os-user=ubuntu \
    --os-key-file=/home/ubuntu/.ssh/id_rsa \
    --cluster-name='ClickHouse 24.8 - Replicated Cluster' \
    --wait \
    --log
```

**Recommended Next Steps**

- *Create Load Balancer*
- *Database Status and Healthchecks*
- *Schedule Backup*
- *Restore External Backup*
- *Alerts and Notifications*

## 2.13 Create Load Balancer

ClusterControl simplifies the deployment of load balancers for database clusters by providing a guided workflow via its web UI and CLI. Load balancers are essential in distributing traffic across multiple database nodes, improving availability, fault tolerance, and performance.

With Create Load Balancer, you can:

- Deploy supported load balancers in front of your database clusters.
- Automatically configure health checks and failover mechanisms.
- Integrate with Keepalived to provide VIP-based failover.
- Manage multiple load balancer instances for redundancy.

### 2.13.1 Support Matrix

ClusterControl supports deploying the following load balancer types:

| Application | Purpose | Supported Databases |
|---|---|---|
| ProxySQL | Load balancer, query router | MySQL, MariaDB, Galera Cluster |
| HAProxy | TCP load balancer | MySQL, MariaDB, Galera Cluster, PostgreSQL, TimescaleDB |
| MariaDB MaxScale | SQL load balancer, query router | MySQL, MariaDB, Galera Cluster |
| garbd | Galera arbitrator (vote-only) | Galera Cluster |
| PgBouncer | Lightweight connection pooler for PostgreSQL | PostgreSQL, TimescaleDB |
| Keepalived | Provides a floating virtual IP address (VIP) service | ProxySQL, HAProxy, MariaDB MaxScale |

**Info**

A single set of load balancers should be dedicated to a single database cluster. ClusterControl does not support managing multiple database clusters sharing one set of load balancers.

### 2.13.2 Prerequisites

Before deploying a load balancer, ensure the following:

- You have a database cluster deployed and managed by ClusterControl.
- Passwordless SSH (SSH using key-based authentication) is configured from the ClusterControl node to load balancer nodes. See *SSH Key-based Authentication*.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*.
- The target node(s) are reachable from the ClusterControl server.

### 2.13.3 ProxySQL

ClusterControl deploys ProxySQL in read/write split mode using ProxySQL query rules where the read-only traffic will be sent to replicas while your writes will be sent to a writable primary by creating two host groups.

In case of primary failure, ProxySQL will detect the new writable primary and route writes to it automatically without any user intervention. ProxySQL offers a wide range of features and capabilities that make it suitable for various scenarios, including improving the scalability, availability, performance, and security of MySQL database infrastructures. You can use an existing database server or another new host by specifying the hostname or IP address.

With two or more ProxySQL nodes, you can then configure a virtual IP address service using Keepalived. See *Keepalived*.

**See also**

Database Load Balancing for MySQL and MariaDB with ProxySQL – Tutorial

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Create balancer**.
2. Choose the load balancer service **ProxySQL**.
3. Click **Continue**.
4. You will be presented with a **Create ProxySQL service** deployment wizard. Specify:
   - **Version**: Choose a supported version.
   - **Server address**: Specify the hostname or IP address of the host. This host must be accessible via passwordless SSH from the ClusterControl node.
   - **Admin port**: ProxySQL administration port. The default is 6032.
   - **Listening port**: ProxySQL for MySQL load balancing. The default is 6033. This is where the applications and clients will connect after ProxySQL is activated.
5. For **ProxySQL Configuration**, you can choose:
   - **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the node during deployment. It is recomemnded to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have configured the necessary ports prior to this deployment otherwise there is a risk of failure deployment. See *Firewall and Security Groups*.
   - **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recomemnded to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.
   - **Import configuration**: Deploys a new ProxySQL based on an existing ProxySQL instance. The source instance must be added first into ClusterControl. Once added, you can choose the source ProxySQL instance from a dropdown list.
   - **Use native clustering**: The ProxySQL Server will be created using ProxySQL native clustering and an entry will be created in the `proxysql_server` tab. It is recommended to enable this if you would like to have more than one ProxySQL node. Port 6032 must be reachable between all ProxySQL nodes.

   Then, click **Continue** to the next step.
6. Under the **Configuration** section, specify the credentials that ClusterControl should use to create the load balancer:
   - **Administration user**: ProxySQL administration user name.
   - **Administration password**: Password for the administration user.

- **Monitor user**: ProxySQL monitor user.
- **Monitor password**: Password for monitor user.
- **Database user**: Create or use an existing database user.
  - For existing user:
    - **Database username**: Choose the existing user name from the dropdown.
    - **Database password**: Password for DB User.
  - For new user:
    - **Database username**: The database user name.
    - **Database password**: Password for DB Users.
    - **Database name**: Database name in "database.table" format.
    - **MySQL privilege(s)**: ClusterControl will load the privilege name along the keypress. Multiple privileges are possible.

**Note**

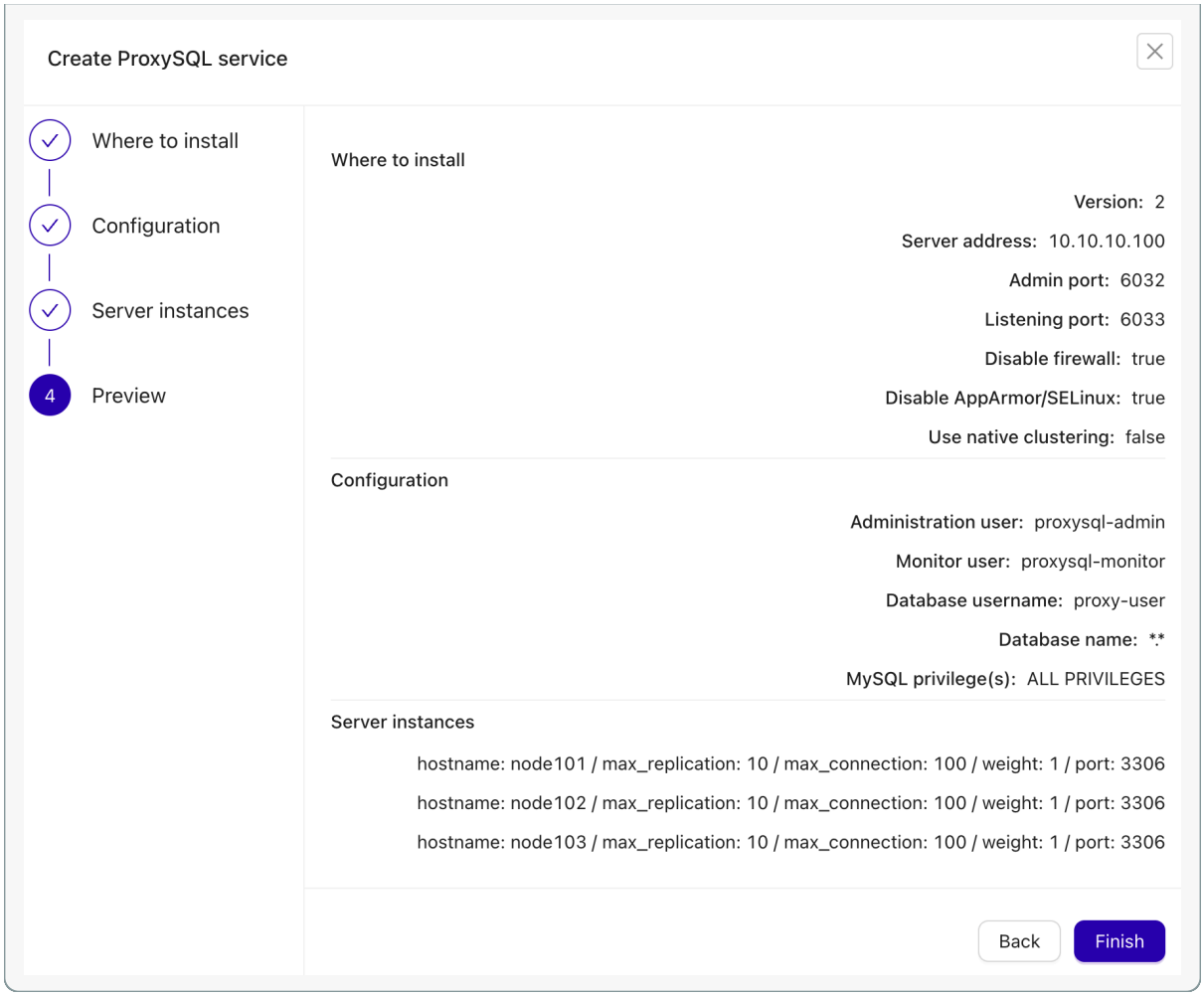
Note

Administration requires port 6032 to be open in the firewall/security group. ::: 7. In the **Server instances** section, specify what nodes are going to be added to the load balancer, its Max replication lag, Max connection, and Weight. Also, choose if you want to use implicit transactions - default value is Off.

```
- **Max replication lag**: How many seconds of replication lags should be␣
↪allowed before marking the node as unhealthy. The default value is 10.
- **Max connection**: Maximum connections are to be sent to the backend␣
↪servers. It's recommended to match or lower than the `max_connections`␣
↪value of the backend servers.
- **Weight**: This value is used to adjust the server's weight relative to␣
↪other servers. All servers will receive a load proportional to their␣
↪weight relative to the sum of all weights. The higher the weight, the␣
↪higher the priority.
- **Are you using implicit transactions?**: Choose **Off** if you rely on␣
↪`SET AUTOCOMMIT=0` to create a transaction. Choose **On** if you␣
↪explicitly use `BEGIN` or `START TRANSACTION` to create a transaction.␣
↪Choose **Off** if you are unsure of this part.
```

8. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your deployment and if everything is correct, you may proceed to deploy the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

**Example output**




9. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Create a ProxySQL node:

```
s9s cluster \
    --cluster-id=2 \
    --add-node \
    --nodes=
↪"proxysql://10.10.10.103?db_username=proxysqluser&db_password=proxysqlpass&db_data
↪'*.*'&db_privs='ALL PRIVILEGES'" \
    --wait \
    --log
```

## 2.13.4 HAProxy

HAProxy is a TCP load balancer and can be used for MySQL/MariaDB replication, Galera Cluster, PostgreSQL and TimescaleDB.

ClusterControl will automatically install and configure HAProxy, install and configure the healthcheck script (mysqlchk for MySQL/MariaDB-based clusters and postgresqlchk for PostgreSQL-based clusters) on each of the database nodes using systemd service, and start the HAProxy service. If you set up read/write

splitting for primary-replica replication, there will be two listening ports configured (one for read-write and another one for read-only connections). You can use an existing database server or another new host by specifying the hostname or IPv4 address.

With two or more HAProxy nodes, you can then configure a virtual IP address service using Keepalived. See *Keepalived*.

**See also**

MySQL Load Balancing with HAProxy – Tutorial

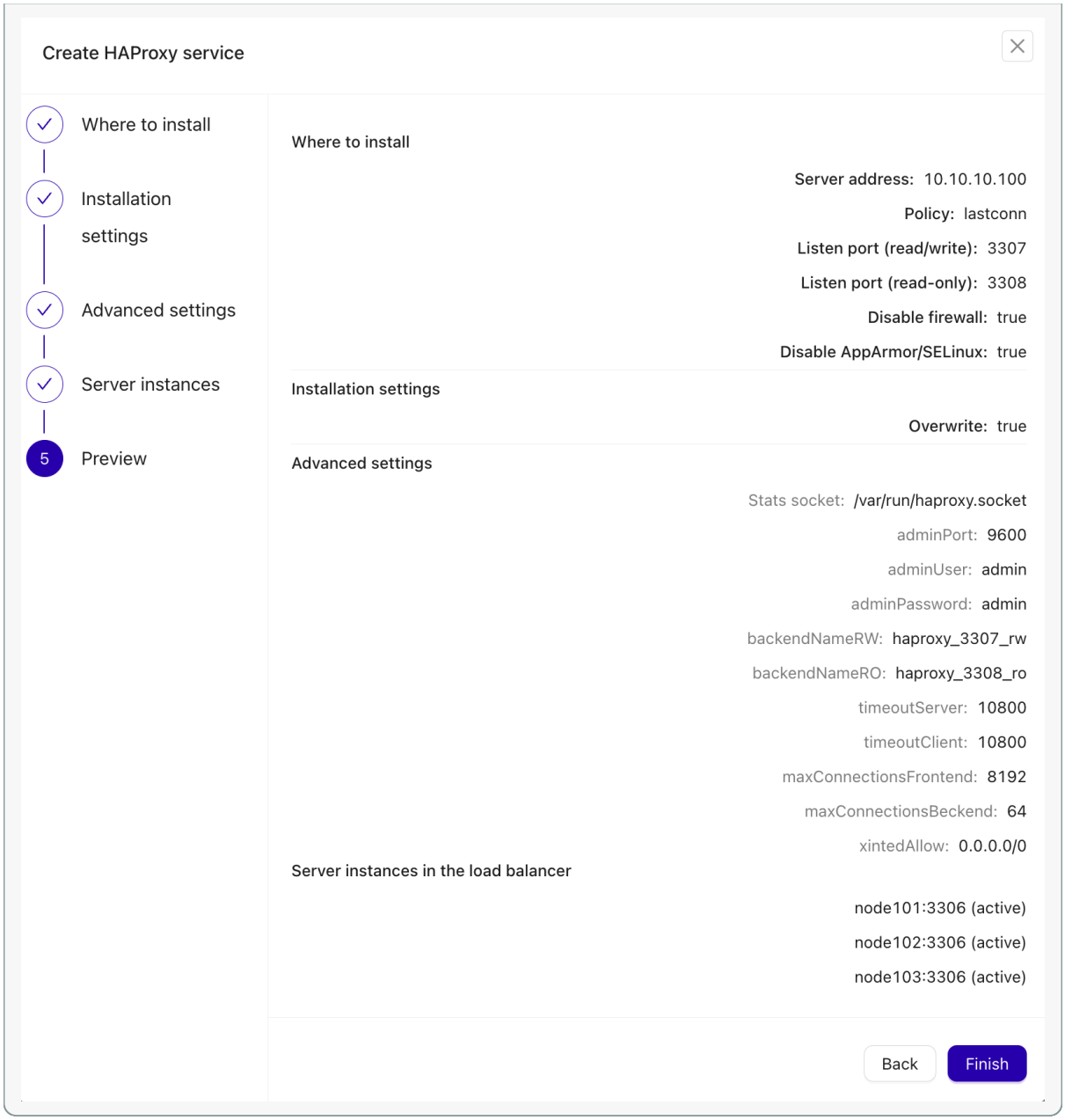
### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Create balancer**.
2. Choose the load balancer service **HAProxy**.
3. Click **Continue**.
4. You will be presented with a **Create HAProxy service** deployment wizard. Specify:
   - **Server address**: Select which host to add the load balancer. If the host is not provisioned by ClusterControl, type in the IP address or hostname. The required files will be installed on the new host. Note that ClusterControl will access the new host using passwordless SSH.
   - **Policy**: Choose one of these load-balancing algorithms:
     - **leastconn:** The server with the lowest number of connections receives the connection.
     - **roundrobin:** Each server is used in turns, according to their weights.
     - **source:** Same client IP address will always reach the same server as long as no server goes down.
   - **Listen port (read/write)**: The listening port that ClusterControl shall configure during the deployment process to receive read/write traffic.
   - **Listen port (read-only)**: The listening port that ClusterControl shall configure during the deployment process to receive read-only traffic.
   - **Install for read/write splitting (master-slave replication)**: Toggled On if you want HAProxy to use another listener port for read-only. A new text box for **Listen Port (Read Only)** will appear right next to the **Listen Port (Read/Write)** text box. You can then specify the port for read-only database connection port.
   - **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the node during deployment. It is recomemnded to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have configured the necessary ports prior to this deployment otherwise there is a risk of failure deployment. See *Firewall and Security Groups*.
   - **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recomemnded to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.

   Then, click **Continue** to the next step.

5. Under the **Installation settings** section, specify if you want to Overwrite existing health check script on the node. Enabled by default.
6. In **Advanced settings** section, specify:
   - **Stats socket**: Specify the path to bind a UNIX socket for HAProxy statistics.
   - **Admin port**: Port to listen to the HAProxy statistic page.
   - **Admin user**: Admin username to access the HAProxy statistic page.
   - **Admin password**: Password for the admin user.
   - **Backend name (RW)**: Name of the read-write backend to be created in HAProxy.
   - **Backend name (RO)**: Name of the read-only backend to be created in HAProxy.
   - **Timeout server (in seconds)**: Sets the maximum inactivity time on the server side.
   - **Timeout client (in seconds)**: Sets the maximum inactivity time on the client side.
   - **Max connections frontend**: Sets the maximum per-process number of concurrent connections to the HAProxy instance.
   - **Max connections backend (per instance)**: Sets the maximum per-process number of concurrent connections per backend instance.
   - **xinetd allow connections from**: The specified subnet will be allowed to access the health check script `mysqlchk` or `mysqlchk_rw_split` for read/write splitting (MySQL-based clusters) or `postgreschk` and `postgreschk_rw_split` (PostgreSQL-based clusters) as xinetd service, which listens on port 9200 on every database node. To allow connections from all IP addresses, use the default value, "0.0.0.0/0".
7. For the **Server instances** section, switch on the instances you want to be included in the load balancer. As **Advanced options** you can set Role (`active` – the server is actively used in load balancing, or `backup` – the server is only used in load balancing when all other non-backup servers are unavailable) and **Connection address** (choose the IP address where HAProxy should be listening on the host) for each database node.
8. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your deployment and if everything is correct, you may proceed to deploy the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

**Example output**




9. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

## ClusterControl CLI

- Create an HAProxy node:

```
s9s cluster \
    --add-node \
    --cluster-id=1 \
    --nodes="haproxy://10.10.10.101" \
    --wait \
    --log
```

### 2.13.5 MariaDB MaxScale

MaxScale is an intelligent proxy that allows the forwarding of database statements to one or more database servers using complex rules, a semantic understanding of the database statements, and the roles of the various servers within the backend cluster of databases.

You can deploy or import the existing MaxScale node as a load balancer and query router for your Galera Cluster and MariaDB replication. For new deployment using ClusterControl, by default, it will create two listeners:

- RW – Implements read-write split access (default port 4008)
- RR – Implements round-robin access (default port 4006)

With two MaxScale nodes, you can then configure a virtual IP address service using Keepalived. See *Keepalived*.

**Warning**

Attention

ClusterControl performs MariaDB MaxScale installation via direct package download without using the MariaDB repository. The package download URL is kept inside `/usr/share/cmon/templates/packages.conf` on the ClusterControl server under the `[maxscale]` section. Occasionally, the provided URL will be outdated as MariaDB releases a new minor version, and removes the older minor version for a specific MaxScale major version. If that is the case, a manual modification is required to update the download link in this file. The updated download URL is available on the MariaDB MaxScale website.

#### ClusterControl GUI

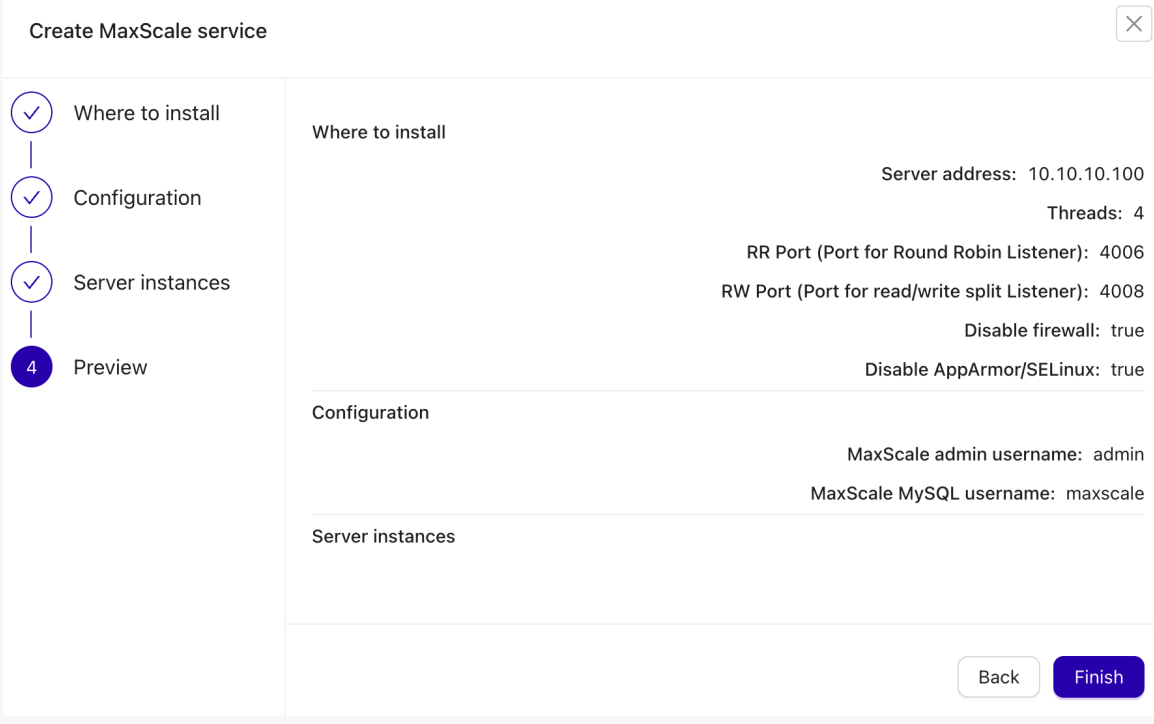
1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Create balancer**.
2. Choose the load balancer service **MariaDB MaxScale**.
3. Click **Continue**.
4. You will be presented with a **Create MaxScale service** deployment wizard. Specify:
   - **Server address**: The IP address of the node where MaxScale will be installed. ClusterControl must be able to perform passwordless SSH to this host.
   - **Threads**: How many threads MaxScale is allowed to use.
   - **RR Port**: Port for round-robin listener. The default is 4006.
   - **RW Post**: Port for the read-write split listener. The default is 4008.
   - **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the node during deployment. It is recomemnded to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have configured the necessary ports prior to this deployment otherwise there is a risk of failure deployment. See *Firewall and Security Groups*.
   - **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recomemnded

to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.

Then, click **Continue** to the next step.

5. Under the **Configuration** section, specify the credentials that ClusterControl should use to create the load balancer:
   - **MaxScale admin username**: MaxScale admin username. The default is 'admin'.
   - **MaxScale admin password**: MaxScale enforces that the admin password for admin user 'admin' is 'mariadb'. If you want to change or use another password you must create another user. The 'admin' user can later be dropped.
   - **MaxScale MySQL username**: MariaDB/MySQL user that will be used by MaxScale to access and monitor the MariaDB/MySQL nodes.
   - **MaxScale MySQL password**: Password of MaxScale MySQL Username.
6. In the **Server instances** section, switch on the instances you want to be included in the load balancer.
7. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your deployment and if everything is correct, you may proceed to deploy the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

**Example output**




8. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Create a MaxScale node:

```
s9s cluster \
    --cluster-id=2 \
    --add-node \
    --nodes="maxscale://10.10.10.104" \
    --maxscale-mysql-user=maxscaleuser \
    --maxscale-mysql-password=maxscalepass \
    --wait \
    --log
```

## 2.13.6 Garbd

Exclusive for Percona XtraDB Cluster or MariaDB Galera Cluster. Galera arbitrator daemon (garbd) can be installed to avoid network partitioning or split-brain scenarios.

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Create balancer**.
2. Choose the load balancer service **Garbd**.
3. Click **Continue**.
4. You will be presented with a **Create garbd service** deployment wizard. Specify:
   - **Server address**: The server address that ClusterControl will use to deploy the load balancer. ClusterControl must be able to perform passwordless SSH to this host.
   - **CmdLine**: Garbd command line to start garbd process on the target node.
   - **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the node during deployment. It is recomemnded to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have configured the necessary ports prior to this deployment otherwise there is a risk of failure deployment. See *Firewall and Security Groups*.
   - **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recomemnded to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.

   Then, click **Continue** to the next step.
5. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your deployment and if everything is correct, you may proceed to deploy the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

**Example output**




8. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

**Warning**

Attention

ClusterControl does not support and allow garbd deployment on a server where ClusterControl is also running and hosted. There is a tendency that the existing MySQL packages will be removed which is managed by the software packaging tools.

### 2.13.7 PgBouncer

This section is for PgBouncer nodes (PostgreSQL, TimescaleDB). PgBouncer is a lightweight connection pooler for PostgreSQL. It reduces PostgreSQL resource consumption (memory, backends, fork) and supports online restart or upgrade without dropping client connections.

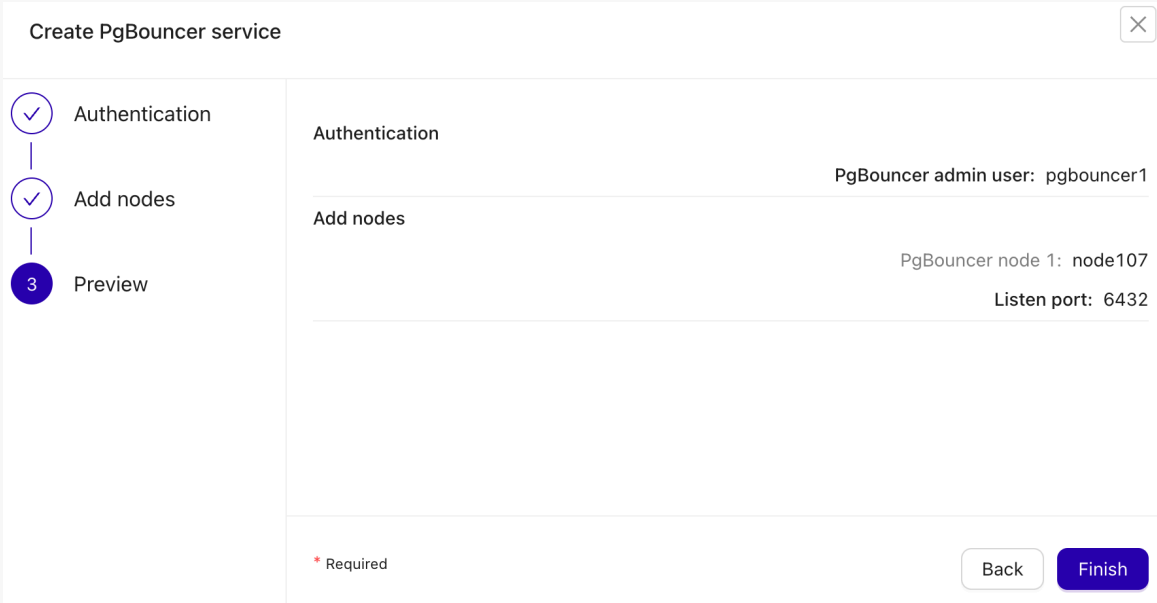
Using ClusterControl, you can manage PgBouncer on one or more nodes, manage multiple pools per node and support 3 pool modes:

- **session (default):** When a client connects, a server connection will be assigned to it for the whole duration the client stays connected. When the client disconnects, the server connection will be put back into the pool.
- **transaction:** A server connection is assigned to a client only during a transaction. When PgBouncer notices that the transaction is over, the server connection will be put back into the pool.
- **statement:** The server connection will be put back into the pool immediately after a query completes. Multi-statement transactions are disallowed in this mode as they would break.

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Create balancer**.
2. Choose the load balancer service **PgBouncer**.
3. Click **Continue**.
4. You will be presented with a **Create PgBouncer service** deployment wizard. Specify:
   - **PgBouncer admin user**: PgBouncer admin username. ClusterControl will create the admin user if specified. If empty, ClusterControl will generate an admin user called "pgbadmin".
   - **PgBouncer admin password**: The password for the PgBouncer admin user.
5. Under the **Add nodes** section, specify the nodes where to deploy PgBouncer and the Listening port to be used (default is 6432).
6. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your deployment and if everything is correct, you may proceed to deploy the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

**Example output**




7. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Create a ProxySQL node:

```
s9s cluster \
    --cluster-id=4 \
    --add-node \
    --nodes="pgbouncer://10.10.10.107" \
```

(continues on next page)

(continued from previous page)

```
    --wait \
    --log
```

### 2.13.8 Keepalived

Keepalived requires two or more HAProxy, ProxySQL, or MariaDB MaxScale instances to provide virtual IP address failover. Keepalived uses the IP Virtual Server (IPVS) kernel module to provide transport layer (Layer 4) load balancing, redirecting requests for network-based services to individual members of a server cluster.

Initially, the virtual IP address is assigned to the instance designated as 'Keepalived 1'. In the event of a failure of this node, the IP address is automatically failed over to 'Keepalived 2'.

**See also**

To understand further how ClusterControl configures Keepalived, see this blog post: How ClusterControl Configures Virtual IP and What to Expect During Failover

#### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Create balancer**.
2. Choose the load balancer service **Keepalived**.
3. Click **Continue**.
4. You will be presented with a **Create Keepalived service** deployment wizard:
   - **Load balancer type**: HAProxy, ProxySQL, MaxScale.
   - **Keepalived**: Choose the servers from the list or type IP address or hostname of the nodes where to install Keepalived.
   - **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the node during deployment. It is recomemnded to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have configured the necessary ports prior to this deployment otherwise there is a risk of failure deployment. See *Firewall and Security Groups*.
   - **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recomemnded to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.
5. Under the **Host configuration** section, specify the virtual IP address and network interface where to add this virtual IP address. Specify a network interface to bind the virtual IP address of the load balancer host. This interface must be able to communicate with other Keepalived instances and support IP protocol 112 (VRRP) and unicast.

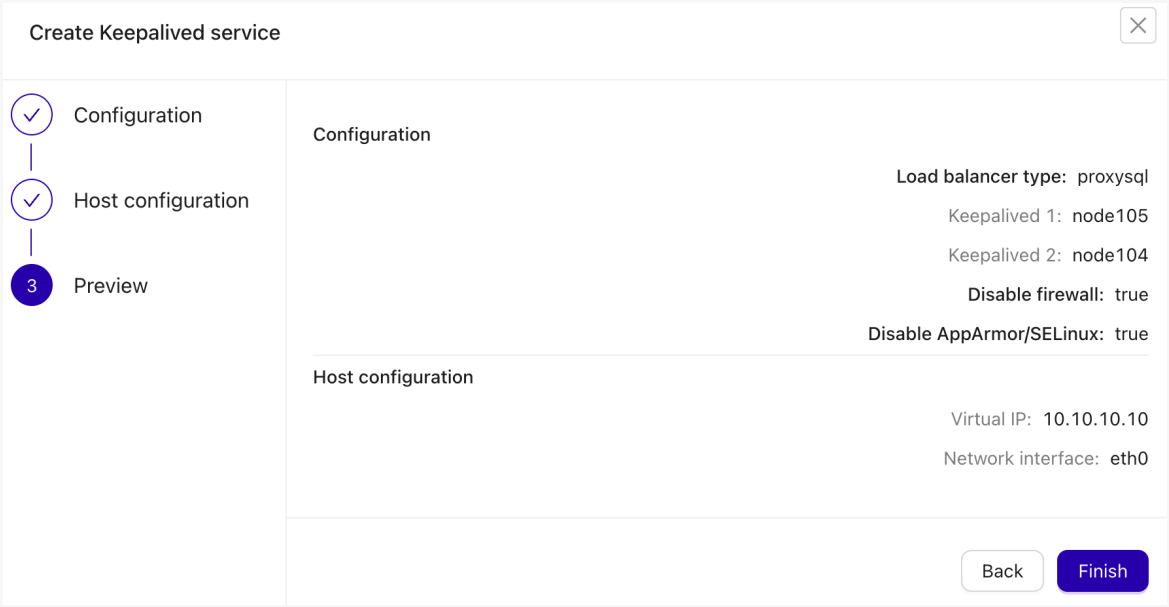
**Warning**

Attention

```
Make sure the specified network interface matches the name of the␣
↪network interface in the node.
```

6. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your deployment and if everything is correct, you may proceed to deploy the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The deployment settings will be kept until you exit the deployment wizard.

**Example output**




7. ClusterControl will trigger a deployment job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

## 2.14 Import Load Balancer

The Import Load Balancer feature in ClusterControl supports importing existing load balancer instances into its management console. It enables users to integrate external load balancers with imported database clusters, allowing centralized visibility, monitoring, and management from within ClusterControl.

With Import Load Balancer, it allows you to:

- Import supported load balancers in front of your database clusters.
- Monitor performance metrics such as throughput, latency, and active connections.
- Automatically configure health checks and failover mechanisms.
- Manage configuration changes.
- Integrate with Keepalived to provide VIP-based failover.
- Manage multiple load balancer instances for redundancy.

ClusterControl supports importing the following load balancer types:

- ProxySQL

- HAProxy
- MariaDB MaxScale
- Garbd
- PgBouncer
- Keepalived

### 2.14.1 Prerequisites

Before importing a load balancer, ensure the following:

- You have a database cluster imported and managed by ClusterControl. Every load balancer must be associated with a database cluster, and can't be shared with another cluster.
- Passwordless SSH (SSH using key-based authentication) is configured from the ClusterControl node to load balancer nodes. See *SSH Key-based Authentication*.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*.
- The target node(s) are reachable from the ClusterControl server.

### 2.14.2 Support Matrix

| Load Balancer | Supported Databases |
|---|---|
| ProxySQL | MySQL, MariaDB, Galera Cluster |
| HAProxy | MySQL, MariaDB, Galera Cluster, PostgreSQL, TimescaleDB |
| MariaDB MaxScale | MySQL, MariaDB, Galera Cluster |
| Garbd | Galera Cluster |
| PgBouncer | PostgreSQL, TimescaleDB |
| Keepalived | For floating virtual IP address (VIP) setup. Support ProxySQL, HAProxy and MariaDB MaxScale |

### 2.14.3 ProxySQL

This section is for ProxySQL nodes (MySQL/MariaDB replication, Galera Cluster). By default, ClusterControl imports ProxySQL in read/write split mode – your read-only traffic will be sent to replicas while your writes will be sent to a writable primary by creating two host groups. In case of primary failure, ProxySQL will detect the new writable primary and route writes to it automatically without any user intervention.

ProxySQL offers a wide range of features and capabilities that make it suitable for various scenarios, including improving the scalability, availability, performance, and security of MySQL database infrastructures. You can use an existing database server or another new host by specifying the hostname or IP address. With two or more ProxySQL nodes, you can then configure a virtual IP address service using Keepalived.

**See also**

Database Load Balancing for MySQL and MariaDB with ProxySQL – Tutorial

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Import balancer**.
2. Choose the load balancer service **ProxySQL**.
3. Click **Continue**.
4. You will be presented with a **Import ProxySQL service**, and on the step 1, it's on the Configuration:
   - **Server address**: Specify the hostname or IP address of the host. This host must be accessible via passwordless SSH from the ClusterControl node.
   - **Admin port**: ProxySQL administration port. The default is 6032.
   - **Listening port**: ProxySQL for MySQL load balancing. The default is 6033. This is where the applications and clients will connect after ProxySQL is activated.
   - **Import configuration**: By default, this is **Off**. If set to **On**, this means its enabled. A text field will show up allowing you to type the hostname of your source ProxySQL instance that contains the configuration.
5. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The import settings will be kept until you exit the import wizard.
6. ClusterControl will trigger an import job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Import a ProxySQL node:

```
s9s cluster \
    --register \
    --cluster-id=2 \
    --nodes=
↪"proxysql://10.10.10.103?db_username=proxysqluser&db_password=proxysqlpass&db_data
↪'*.*'&db_privs='ALL PRIVILEGES'" \
    --wait \
    --log
```

## 2.14.4 HAProxy

This section is for HAProxy nodes (MySQL/MariaDB replication, Galera Cluster, PostgreSQL, TimescaleDB). ClusterControl will automatically install and configure HAProxy, install mysqlcheck script (for MySQL health checks) on each of the database nodes as part of xinetd service, and start the HAProxy service. If you set up read/write splitting for primary-replica replication, there will be two listening ports configured (one for read-write and another one for read-only connections). You can use an existing database server or another new host by specifying the hostname or IPv4 address. With two or more HAProxy nodes, you can then configure a virtual IP address service using Keepalived.

**See also**

MySQL Load Balancing with HAProxy – Tutorial

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Import balancer**.
2. Choose the load balancer service **HAProxy**.
3. Click **Continue**.
4. You will be presented with a **Import HAProxy service**, and on the step 1, it's on the Configuration:
   - **Server address**: Select which host to add the load balancer. If the host is not provisioned by ClusterControl, type in the IP address or hostname. The required files will be installed on the new host. Note that ClusterControl will access the new host using passwordless SSH.
   - **Port**: By default, this is set to 9600. This is the port to access stats for HAProxy. Unless you have different port, then change it to something else other than 9600.

   Then, click **Continue** to the next step.
5. In **Advanced settings** section, specify:
   - **Admin user**: Admin username to access the HAProxy statistic page.
   - **Admin password**: Password for the admin user.
   - **cmdline**: This is a string of command line that will be executed to run the daemon service or server. By default, the value is `/usr/sbin/haproxy -D -f /etc/haproxy/haproxy.cfg -sf $(cat /var/run/haproxy.pid)`
   - **LB name**: This shall be the load balancer name. By default, this is set to `lb_group1`
   - **Haproxy config**: The default value is `/etc/haproxy/haproxy.cfg`. This is the path of the configuration for your HAProxy server.
   - **Stats socket**: The default value is `/var/run/haproxy.socket`. This is the path of the socket file for your haproxy server.
6. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The import settings will be kept until you exit the import wizard.
7. ClusterControl will trigger an import job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Import an existing HAProxy node:

```
s9s cluster \
    --register \
    --cluster-id=1 \
    --nodes="haproxy://10.10.10.101" \
```

(continues on next page)

(continued from previous page)

```
    --wait \
    --log
```

### 2.14.5 MariaDB MaxScale

This section is for MariaDB MaxScale nodes (MySQL/MariaDB replication, Galera Cluster). MaxScale is an intelligent proxy that allows the forwarding of database statements to one or more database servers using complex rules, a semantic understanding of the database statements, and the roles of the various servers within the backend cluster of databases.

You can import the existing MaxScale node as a load balancer and query router for your Galera Cluster and MySQL/MariaDB replication. For new import using ClusterControl, by default, it will create two production services:

- RW – Implements read-write split access.
- RR – Implements round-robin access.

With two MaxScale nodes, you can then configure a virtual IP address service using Keepalived.

**Warning**

Attention

ClusterControl performs MariaDB MaxScale installation via direct package download without using the MariaDB repository. The package download URL is kept inside `/usr/share/cmon/templates/packages.conf` on the ClusterControl server under the `[maxscale]` section. Occasionally, the provided URL will be outdated as MariaDB releases a new minor version, and removes the older minor version for a specific MaxScale major version. If that is the case, a manual modification is required to update the download link in this file. The updated download URL is available on the MariaDB MaxScale website.

#### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Import balancer**.
2. Choose the load balancer service **MariaDB MaxScale**.
3. Click **Continue**.
4. You will be presented with a **Import MaxScale service** import wizard. Specify:
    - **Server address**: The IP address of the node where MaxScale will be installed. ClusterControl must be able to perform passwordless SSH to this host.
    - **Threads**: How many threads MaxScale is allowed to use.
    - **RR Port**: Port for round-robin listener. The default is 4006.
    - **RW Post**: Port for the read-write split listener. The default is 4008.
    - **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the node during import. It is recomemnded to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have

configured the necessary ports prior to this import otherwise there is a risk of failure import. See *Firewall and Security Groups*.

- **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recomemnded to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.

Then, click **Continue** to the next step.

5. Under the **Configuration** section, specify the credentials that ClusterControl should use to create the load balancer:
   - **MaxScale admin username**: MaxScale admin username. The default is 'admin'.
   - **MaxScale admin password**: MaxScale enforces that the admin password for admin user 'admin' is 'mariadb'. If you want to change or use another password you must create another user. The 'admin' user can later be dropped.
   - **MaxScale MySQL username**: MariaDB/MySQL user that will be used by MaxScale to access and monitor the MariaDB/MySQL nodes.
   - **MaxScale MySQL password**: Password of MaxScale MySQL Username.
6. In the **Server instances** section, switch on the instances you want to be included in the load balancer.
7. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The import settings will be kept until you exit the import wizard.
8. ClusterControl will trigger an import job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

#### ClusterControl CLI

- Import an existing MaxScale node:

```
s9s node \
    --register \
    --cluster-id=2 \
    --nodes="maxscale://10.10.10.104" \
    --maxscale-mysql-user=maxscaleuser \
    --maxscale-mysql-password=maxscalepass \
    --wait \
    --log
```

### 2.14.6 Garbd

Exclusive for Galera Cluster. Galera arbitrator daemon (garbd) can be installed to avoid network partitioning or split-brain scenarios. If you already have an existing Galera arbitrator daemon running, you can import it into ClusterControl.

#### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Import balancer**.
2. Choose the load balancer service **Garbd**.
3. Click **Continue**.
4. You will be presented with a **Import garbd service** import wizard. Specify:
   - **Server address**: The server address that ClusterControl will use to import the load balancer. ClusterControl must be able to perform passwordless SSH to this host.
   - **CmdLine**: Garbd command line to start garbd process on the target node.
   - **Disable firewall**: Whether ClusterControl should disable firewall (`iptables`, `ufw`, `firewalld`) on the node during import. It is recomemnded to use the default setting (checked). If unchecked, this configuration task will be skipped. If the firewall is enabled, ensure you have configured the necessary ports prior to this import otherwise there is a risk of failure import. See *Firewall and Security Groups*.
   - **Disable SELinux/AppArmor**: Whether ClusterControl should disable SELinux (Red Hat-based operating system) or AppArmor (Debian-based operating system). It is recomemnded to use the default setting (checked). If unchecked, ensure you have set a proper policy for the database-related processes and all of their dependencies.

   Then, click **Continue** to the next step.
5. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The import settings will be kept until you exit the import wizard.
6. ClusterControl will trigger an import job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

**Warning**

Attention

ClusterControl does not support and allow garbd import on a server where ClusterControl is also running and hosted. There is a tendency that the existing MySQL packages will be removed which is managed by the software packaging tools.

### 2.14.7 PgBouncer

This section is for PgBouncer nodes (PostgreSQL, TimescaleDB). PgBouncer is a lightweight connection pooler for PostgreSQL. It reduces PostgreSQL resource consumption (memory, backends, fork) and supports online restart or upgrade without dropping client connections.

Using ClusterControl, you can manage PgBouncer on one or more nodes, manage multiple pools per node and support 3 pool modes:

- **session (default):** When a client connects, a server connection will be assigned to it for the whole duration the client stays connected. When the client disconnects, the server connection will be put back into the pool.

- **transaction:** A server connection is assigned to a client only during a transaction. When PgBouncer notices that the transaction is over, the server connection will be put back into the pool.
- **statement:** The server connection will be put back into the pool immediately after a query completes. Multi-statement transactions are disallowed in this mode as they would break.

#### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Import balancer**.
2. Choose the load balancer service **PgBouncer**.
3. Click **Continue**.
4. You will be presented with a **Import PgBouncer service** import wizard. In the first step, under Authentication, specify the following fields below:
   - **PgBouncer admin user**: PgBouncer admin username you currently used in your PgBouncer
   - **PgBouncer admin password**: The password for the PgBouncer admin user.
5. Under the **Add nodes** section, specify the nodes where to import PgBouncer and the Listening port to be used (default is 6432).
6. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The import settings will be kept until you exit the import wizard.
7. ClusterControl will trigger an import job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

#### ClusterControl CLI

- Import an existing PgBouncer node:

```
s9s node \
    --register \
    --cluster-id=4 \
    --nodes="pgbouncer://10.10.10.107" \
    --wait \
    --log
```

### 2.14.8 Keepalived

This section is for Keepalived import. Keepalived requires two or more HAProxy, ProxySQL, or MariaDB MaxScale instances to provide virtual IP address failover. By default, the virtual IP address will be assigned to instance 'Keepalived 1'. If the node goes down, the IP address will automatically fail over to 'Keepalived 2' accordingly.

Keepalived uses the IP Virtual Server (IPVS) kernel module to provide transport layer (Layer 4) load balancing, redirecting requests for network-based services to individual members of a server cluster.

**See also**

How ClusterControl Configures Virtual IP and What to Expect During Failover

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Select a cluster → Cluster Actions → Add new → Load balancer*** and choose **Import balancer**.
2. Choose the load balancer service **Keepalived**.
3. Click **Continue**.
4. You will be presented with a **Import Keepalived service** import wizard:
   - **Keepalived**: Choose the servers that contains your Keepalived instance. Take note that importing a Keepalived requires that the host/node shall be existing in the cluster.
5. Click **Continue**.
6. Under the **Host configuration** section, specify the virtual IP address and network interface that is currently used by your existing Keepalived in your database cluster.

**Warning**

Attention

```
Make sure the network interface matches the name of the network␣
↪interface in the nodes.
```

7. Click **Continue** to proceed to the **Preview** step. In this section, you can see the summary of your import and if everything is correct, you may proceed to import the load balancer by clicking **Finish**. You can always go back to any previous section to modify your configurations if you wish. The import settings will be kept until you exit the import wizard.
8. ClusterControl will trigger an import job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

## 2.15 Observability

This section explores the essential strategies and tools for monitoring and analyzing your database clusters with ClusterControl. Learn how to gain full visibility into performance, detect issues proactively, and ensure optimal health of your database environment. You can also scale observability as your clusters grow.

**See also**

- *Database Status and Healthchecks →*
- *Host Monitoring →*
- *Query Monitoring →*
- *Database Performance →*
- *Database Logs →*

- *Database Audit Logs →*
- *Watchlist →*
- *Activity Center→*
- *Operational Reports →*

## 2.16 Database Status & Health Checks

There are several ways to perform database health checks to ensure it runs correctly. ClusterControl provides comprehensive features to monitor your database clusters. Here's how to effectively check database health and status:

### 2.16.1 Cluster Summary

The Cluster Summary page appears after login into the ClusterControl UI. The Cluster Summary describe the status of each cluster database either it is in `Operational`, `Failed`, `Offline`, `Shutdown`, `Recovering` or `Unreachable/Unknown`. There is also information about global alert in Cluster Summary which give information about Critical alerts across all clusters, Warning notifications, Informational messages.

There is a filter on the right side in Cluster Summary which you can filter the status of Cluster either Operational, Unknown State, Failed, or All.

How to access the Cluster Summary:

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. The landing page will go to the Home menu and you will see the status of each database cluster, nodes, and alarms too.

#### ClusterControl CLI

1. Check status of the cluster.

```
s9s cluster --list --long
```

2. Filter the cluster specific by `clusterID`.

```
s9s cluster --cluster-id=<clusterID> --list --long
```

### 2.16.2 Dashboard Metrics

The dashboard metrics provides a comprehensive set of real-time and historical metrics on its dashboard to monitor your database clusters from system-level metrics, database specific, cluster health status, performance advisors.

### Cluster Load

The cluster-wide load metrics to help monitor the overall performance and health of entire database cluster (not just individual nodes). These metrics give a high-level view of how well the cluster is handling workloads and whether it's operating efficiently. It give insight related to `Total Queries Per Second (QPS)`, `Read vs. Write Ratio (%)`, `Slow Queries Count`, `Connection`, `Active Connections`.

Monitor the Cluster Load in ClusterControl:

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the Dashboard.
3. Choose the **Cluster Overview**, and you will see the **Cluster Load** metrics.

### Cluster Health Status

The Cluster Health metrics provide a holistic view of the database cluster's operational state, ensuring high availability and quick issue detection. The node status metrics display real-time condition on each servers, clearly indicating roles (primary/master for write operation or secondary/slave for read scaling), the replication status, and health states (online, offline, recovery).

Utilize the Cluster Health Metrics in ClusterControl using statistics information in the database overview:

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the Dashboard.
3. Choose the `Cluster Overview`, and you will see information about status of Primary & Secondary nodes status of database. Hover the pointer into the each column will describe the ToolTip which describe information about the column.
4. The statistics can be expanded, there is icon in the right side to expand the data. For example in MariaDB/MySQL-based, information available in Primary Node are: Instance, ServerID, ExecutedGTID, Binlog File, Binlog Pos, Binlog do DB and Binlog ignore DB, Semisync status.

**Note**

Note Available in MySQL/MariaDB and PostgreSQL

There is information to check the status database.

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the Dashboard.
3. Choose the `MySQL Server - General` or `MongoDB Server`, and it will see information about database Uptime, queries per second, percentage of connection used, and buffer hit ratio.

**Note**

Note Available in MySQL and MongoDB

Another metric that can be used to check the database status and health is the Load Average Ratio. The Load Average refers to the average number of processes that are either running on the CPU or waiting for CPU time.

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the Dashboard.
3. Choose the `Cluster Overview`, and it will see information Load Average in 1 minute, 5 minutes, and 15 minutes. The default range of time metrics is 15 minutes.

### 2.16.3 Performance Advisors

The Performance Advisor in ClusterControl is an intelligent monitoring feature that automatically analyzes database clusters to detect performance issues, configuration problems, and potential risks, providing actionable recommendations for optimization.

There are a number of metrics monitored by the advisors, such as:

- CPU usage
- Disk Space usage
- Connection used
- Connection ever used
- InnoDB Dirty Pages
- Top Queries
- Numa checks
- Replication check

The thresholds, justifications, and advice are based on a custom configuration, which is flexible to manage. The advisors can be managed through:

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the ***Manage → Advisors***.
3. There is ***Action*** on each type of Advisories and some menu such as **Edit scripts** (modify the advisors script based on JavaScript), `Edit Param` (change the schedule and parameter argument), and **Disable** the advisor.

Monitor the Performance Advisors can be accessed in ClusterControl

### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the ***Performance → Advisors***.
3. There will be information about type of Advisories, Host, Status, Justification, and Advice.

Host Monitoring in ClusterControl is designed to provide comprehensive insights into the health, performance, and availability of the hosts (servers) within the database cluster.

It can monitor status of the nodes and key metrics such as CPU usage, memory consumption, disk I/O, network activity, and many more.

## 2.17 Cluster Node Summary

The Cluster Node Summary is part of the Cluster Summary where is available after we login into ClusterControl. The Cluster Node Summary on the home page provides a quick overview of the current status of each node in database cluster.

Here is what it typically includes:

- Node Name/Hostname – Identifies the server/node.
- Status – Indicates if the node is Operational, Failed, Offline, Shutdown, Recovering or Unreachable/Unknown.
- Role – Shows the role of the node (e.g., Master, Slave, Primary, Secondary, depending on the database type).
- Type – Describe of the database engine running on the host.
- Cluster name – Identifies the cluster name to which the node belongs.

To get the cluster node summary:

### ClusterControl GUI

1. Login into the ClusterControl UI.
2. In the **Home** section and you will see the **Nodes** group which appears in the **Status Overview** and **Nodes** itself.
3. The **Status Overview** inside the **Nodes** group shows the summary of the nodes. The green color indicates that the nodes status in operational while the red indicates the nodes is offline. If you rollover to one of the node, it will show more information about the node such as Hostname, Role, Type, Cluster Name, and Status.

### ClusterControl CLI

1. Check status of the nodes and there will be information about cluster name, hostname, port, and status of the nodes.

```
s9s node --list --long
```

2. Filter the nodes specific by `clusterID`.

```
s9s node --cluster-id=<clusterID> --list --long
```

The ***clusterID (CID)*** is the unique identifier for each cluster.

## 2.18 Node Health Status

The node health status represents the current operational state of each individual server (or node) within a database cluster. It is a key part of monitoring and helps you quickly identify if a node is running as expected or experiencing issues.

### 2.18.1 CPU Utilization

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the Dashboard.
3. Choose the **System Overview**, and you will see information about CPU Usage.
4. The CPU Usage metrics describe each status that is running in the nodes such as : Idle, User, System, Iowait, Irq, Streal.

### 2.18.2 Memory Utilization

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the Dashboard.
3. Choose the **System Overview**, and you will see information about memory Usage.
4. The **Memory Usage** metrics show the **Free Memory** and **Used Memory** statistics.

### 2.18.3 Disk IO Utilization

#### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to specific cluster, and choose the Dashboard.
3. Choose the **System Overview**, and you will see information about disk I/O utilization.
4. The **Disk IO Util** metrics show percentage usage on each storage and mount points.

## 2.19 Node Menu

The Node Menu in ClusterControl is a context-specific menu accessible from the ClusterControl UI, typically by clicking on a specific database node listed in the cluster view or in the Nodes menu on the left side of the interface. The Node Menu serves as a key interface for accessing monitoring capabilities related to the status of services on each host.

It also provides a set of actions and detailed options for managing and monitoring the selected node.

### ClusterControl GUI

1. Login into the ClusterControl UI.
2. Go to the ***Nodes*** menu in the left side of ClusterControl UI.
3. In the header, it appears the total of nodes in several categories (`Operational`, `Failed`, `Offline`, `Shutdown`, `Recovering`, `Unkown State`, `All`).
4. Beside the summary header, it appears list of the host with information such as: Type, Status, Role, version, Cluster, Last seen, and last one is Action menu. The actions menu provide some menu for managing the selected nodes (for example: Restart Nodes, Reboot Host, Stop Node, Remove Nodes, Rebuild Replication, Change Replication)

The Query Monitoring feature is a powerful tool designed to provide real-time visibility into the queries running on your database. It helps administrators identify performance bottlenecks, optimize query execution, and ensure the overall health of the database. This feature includes real-time query monitoring, top queries analysis, and query outlier detection, all of which work together to provide actionable insights into database performance.

Every database cluster managed by ClusterControl has it is own Query Monitoring. This will help monitor queries to the specific database cluster. Ideally, after the database cluster has been provisioned, administrators can set up the Slow Query Log. The purpose of the Slow Query Log is to capture slow queries based on a configured threshold.

## 2.20 Configure Slow Query Log

### ClusterControl GUI

1. In the ClusterControl UI, go to the cluster and choose the tab ***Performance → Query Monitor***, go to the ***ettings***.
2. Enable the Query Monitor toggle and set the thresholds for long query time. The unit for long query time is in seconds. For example, we can set the threshold to 5 seconds.
3. In addition to the slow query threshold, we can enable the option to log queries that do not use indexes. This means any query performing a full table scan will also be logged.

### ClusterControl CLI

1. Change the value of the long query time in s9s CLI is very straightforward. Define the `--cluster-id`, and the value of new slow query threshold. The long query time defined as seconds.

```
s9s cluster --change-config --cluster-id=<clusterID> --opt-
↪name=long_query_time --opt-value=5
```

2. Verify the config by executing the following command:

```
s9s cluster --list-config --cluster-id=<id> | grep -i long_query_time
```

The clusterID is the unique identifier belong to the cluster.

## 2.21 Slow Query Monitor

Queries that exceed the slow query threshold and those that do not use indexes will appear in the Top Queries section.

### ClusterControl GUI

1. In the ClusterControl UI, go to the cluster and choose the tab ***Performance → Query Monitor***, go to the ***Top Queries***.
2. It provides detailed information about each query, including the query text, execution time, user, database, and state (example: `running`, `idle`, `waiting`, or `completed`).

The next step after identifying the slow queries is to manually check the execution plan for each query (example: missing indexes, inefficient joins, and other issues) and make the necessary corrections.

## 2.22 RealTime Query Monitor

Administrators can monitor queries running in the database in real time, which is beneficial in the event of an incident. We can immediately check which queries are causing performance issues. Key aspects to watch include the query execution time, query status (active, idle, or waiting), the schema, and the user executing the queries.

### ClusterControl GUI

1. In the ClusterControl UI, go to the cluster and choose the tab ***Performance → Query Monitor***, go to the DB Connections.
2. It provides detailed information about each query, including the query text, execution time, user, database, and state (example: `running`, `idle`, `waiting`, or `completed`).

## 2.23 ProxySQL for Query Monitor

Administrators can also monitor queries in real time through ProxySQL. It monitors queries from the perspective of the database load balancer. The ***Top Queries*** section in ProxySQL provides insights such as the number of queries executed (count_star), total query time (sum_time), and the timestamps for when queries were first and last seen. The ***Process List*** provides information about the queries being executed, including the host group, session ID, thread ID, user, and execution time.

### ClusterControl GUI

1. In the ClusterControl UI, go to the cluster and choose the tab ***Nodes → Node List***, go to the ProxySQL node.
2. There is ***Actions*** menu in the right side, just click the menu and choose ***Node details***.
3. There are several tabs in the ProxySQL node list, and administrators can monitor the queries in the ***Top Queries***.

**Note**

Note ProxySQL is database load balancer that is currently supported for MySQL (Replication and Percona XtraDB Cluster), MariaDB (Replication & Galera Cluster)

## 2.24 Reset Query Monitoring

After the queries have been fixed, they may still appear in the ***Top Query Monitor***. Administrators can reset the query monitoring to start fresh.

### ClusterControl GUI

1. In the ClusterControl UI, go to the cluster and choose the tab ***Performance → Query Monitor***, go to the Top Queries.
2. There is button ***Clear all queries*** to reset the query monitoring or enable ***Auto Clear Queries*** toggle in the Settings.

## 2.25 DB Growth

Provides a summary of your database and table growth on daily basis for the last 30 days. On the first line of the Top 25 Largest Databases grid, you should notice the actual size of the MySQL data directory (with a folder icon). This is useful to determine whether any other files that exist in the data directory may consume huge spaces e.g binary log, error log or MySQL general log.

| Field | Description |
|---|---|
| DB growth | A summary of your database and table growth daily. |
| Top largest tables | A summary of the largest tables per database daily. |
| Update data in controller | Triggers a job to re-calculate the database size. |
| Actions → View | Shows the details per database on table count, row count, data size, index size, and database size. |

Click on a database listed for further details on growth summary per table.

## 2.26 DB Status

Provides a comprehensive overview of your database's current state. It offers insights into various aspects of the database, such as connection status, data usage, and server performance. This feature is particularly useful for database administrators and developers as it allows them to monitor the health and performance of the database, identify potential issues, and make informed decisions to ensure optimal database operation.

By default, ClusterControl performs status collection every 30 seconds. This is configurable using the `db_stats_collection_interval` parameter inside the controller's configuration file.

| Field | Description |
|---|---|
| Show differences | Shows a list of status and values that are not identical among each node (highlighted in red). |
| Filter Server | Specify the database server you want to view. Multiple values accepted. |
| Search | Filter the results. |

## 2.27 DB Variables

Provides detailed information about the configuration of your database. It allows you to view and manage various parameters that control the behavior of your database system. These variables can include settings related to memory allocation, data storage, query optimization, and more. This feature is particularly beneficial for database administrators and developers, enabling them to customize the database environment to best suit their specific needs and application requirements.

By default, ClusterControl performs variable collection every 30 seconds. This is configurable using the `db_stats_collection_interval` parameter inside the controller's configuration file.

| Field | Description |
|---|---|
| Show differences | Shows a list of variables and values that are not identical among each node (highlighted in red). |
| Filter Server | Specify the database server you want to view. Multiple values accepted. |
| Search | Filter the results. |

## 2.28 Advisors

Lists of scheduled advisors' results created in ***Manage → Advisors*** and ***Manage → Scripts*** using ClusterControl DSL. You can think of it as a 'scheduled mini-program' that executes a script created inside ClusterControl and produces a result containing status, advice, and justification. You can get detailed schedules by rolling over to the Advisor's name.

| Field | Description |
|---|---|
| Manage advisors | Opens the ***Manage → Advisors*** for advisor's management. |

## 2.29 Schema Analyzer

Analyze your database schemas for missing primary keys, redundant indexes, and tables using the MyISAM storage engine. Galera Cluster needs an explicitly defined primary key on each table (unique key does not count). MyISAM tables are not recommended in Galera. ClusterControl will periodically check the schemas for these (default every 8 hours or every CMON restart), and raise an alert if necessary.

| Field | Description |
|---|---|
| Tables without Primary Keys | List of tables without primary keys. A primary key is important in Galera. The `DELETE` operations are unsupported on tables without a primary key. Also, rows in tables without a primary key may appear in a different order on different nodes. |
| My ISAM Tables | MyISAM does not support transactions. However, the DMLs for MyISAM should also work but it's still experimental in Galera. |
| Redundant Indexes | Having duplicate keys in schemas can hurt the performance of the database: |
| | * They make the optimizer phase slower because MySQL needs to examine more query plans. |
| | * The storage engine needs to maintain, calculate and update more index statistics. |
| | * DML and even read queries can be slower because MySQL needs to update and fetch more data to Buffer Pool for the same load. |
| | * Data needs more disk space so the backups will be bigger and slower. |

## 2.30 Transaction Deadlocks

Available for: MySQL-based clusters and MariaDB-based clusters

Lists out long-running transactions and deadlocks across database clusters where you can easily view what transactions are causing the deadlocks. The default query time threshold is 30 seconds. This is configurable in the CMON configuration file under `db_long_query_time_alarm` configuration option.

Click on the listed query to see the output of InnoDB status for detailed debugging.

| Field | Description |
|---|---|
| Deadlock check interval | How often to check for deadlocks in seconds. |
| | 0 means disabled (default). |
| | Deadlock detection will affect CPU usage on database nodes. |

## 2.31 Running Operations

Available for: MongoDB-based clusters

Provides a view of current running operations similar to `db.currentOp()` command.

## 2.32 System Logs

Tree view of corresponding database-related logs across a cluster. Clicking on a log file will appear in a new tab in the viewing panel on the right-hand side. Browsing through the appropriate error logs is the key element of the troubleshooting process.

### ClusterControl GUI

1. In the ClusterControl UI, go to the cluster and choose the tab ***Logs → System Log***.
2. It provides detailed log from the operating system, database logs, and load balancer logs for the database.

Click on ***Refresh logs*** will trigger a job for ClusterControl to perform log collection.

## 2.33 Audit Logs

Shows records of ClusterControl's events and changes:

- User authentication, cloud credentials, database configuration, and user changes.
- CMON configuration changes and jobs.
- ClusterControl user management changes (Users and ACLs), operational reports.

### ClusterControl GUI

1. In the ClusterControl UI, go to the cluster and choose the tab ***Logs → Audit Log***.
2. Each log entry has the activity, type of activity, user, hostname or IP address (where the user comes from), cluster name, and timestamp. Rollover on the ***Cluster*** entry will pop up a cluster information card summarizing the cluster status. Rollover on the ***When*** entry will pop up a tooltip containing the exact timestamp with timezone.

## 2.34 Audit Logging

Enable audit logging on the cluster using policy-based monitoring and logging of connection and query activity for security and compliance purposes. It enables database administrators to track and monitor database operations, such as logins, queries, data modifications, and schema changes. This plugin records these activities in a structured format, making it easier to analyze and audit database usage. This feature will enable audit logging on all nodes in the cluster.

!!! Notes

```
Available in MariaDB Replication, MariaDB (Galera Cluster), PostgreSQL and␣
↪TimescaleDB.
```

### 2.34.1 MariaDB

For MariaDB audit logging, the following fields are presented:

| Field | Description |
|---|---|
| Log Path | The filename to store the audit log in the log directory `/var/log/mysql/`. Alternatively, it can contain a path relative to the data directory or an absolute path. The default is `server_audit.log`. |
| Rotation Size in MB | Log file size in MB before log rotation happens. Changing this default value will require a cluster restart. |
| Rotation | Number of log files to keep after rotation. |
| Events | Specify MariaDB audit events that you would like to capture. ClusterControl preloads the audit events as you type. Multiple values are allowed. |
| Exclude Users | Exclude the specified MariaDB user(s) from the auditing. ClusterControl preloads all database users in a dropdown. Multiple values are allowed. |

To enable audit logging for MariaDB-based clusters:

#### ClusterControl GUI

1. Go to the MariaDB cluster, Click on ***Actions Menu → Audit Logging***.
2. Just enable the toggle ***ON*** in the ***Audit Logging*** option.
3. There will be a dialogue box to fill related to the Log Path, Rotation, Events type, and Exclude the Users.

#### ClusterControl CLI

1. Enable audit logging through CLI as easy as the command below:

```
s9s cluster --setup-audit-logging --cluster-id=<clusterId>
```

   The `clusterId` indicate identifier of your MariaDB cluster.

### 2.34.2 PostgreSQL

For PostgreSQL/TimescaleDB audit logging, the following fields are presented:

| Field | Description |
|---|---|
| Event | Specify PostgreSQL audit events that you would like to capture. ClusterControl preloads the audit events as you type. Multiple values are allowed. |

To enable audit logging in PostgreSQL-based environment:

#### ClusterControl GUI

1. Go to the PostgreSQL/TimescaleDB cluster, Click on ***Actions Menu → Audit Logging***.
2. Just enable the toggle ***ON*** in the ***Audit Logging*** option.
3. There will be a dialogue box to fill the ***Events*** (eg. `ROLE`, `DDL`, `READ`, `WRITE`, `ALL`).

### ClusterControl CLI

1. Enable audit logging through CLI as easily as the command below:

```
s9s cluster --setup-audit-logging --cluster-id=<clusterId>
```

2. Enable audit logging for specific roles or events through CLI:

```
s9s cluster --setup-audit-logging --cluster-id=<clusterId> --
↪audit-log-events-data=role,ddl
```

The `clusterId` indicates the identifier of your PostgreSQL cluster.

After enabling the audit log in the database, the output of the audit activity will be stored in the audit log file.

**Warning**

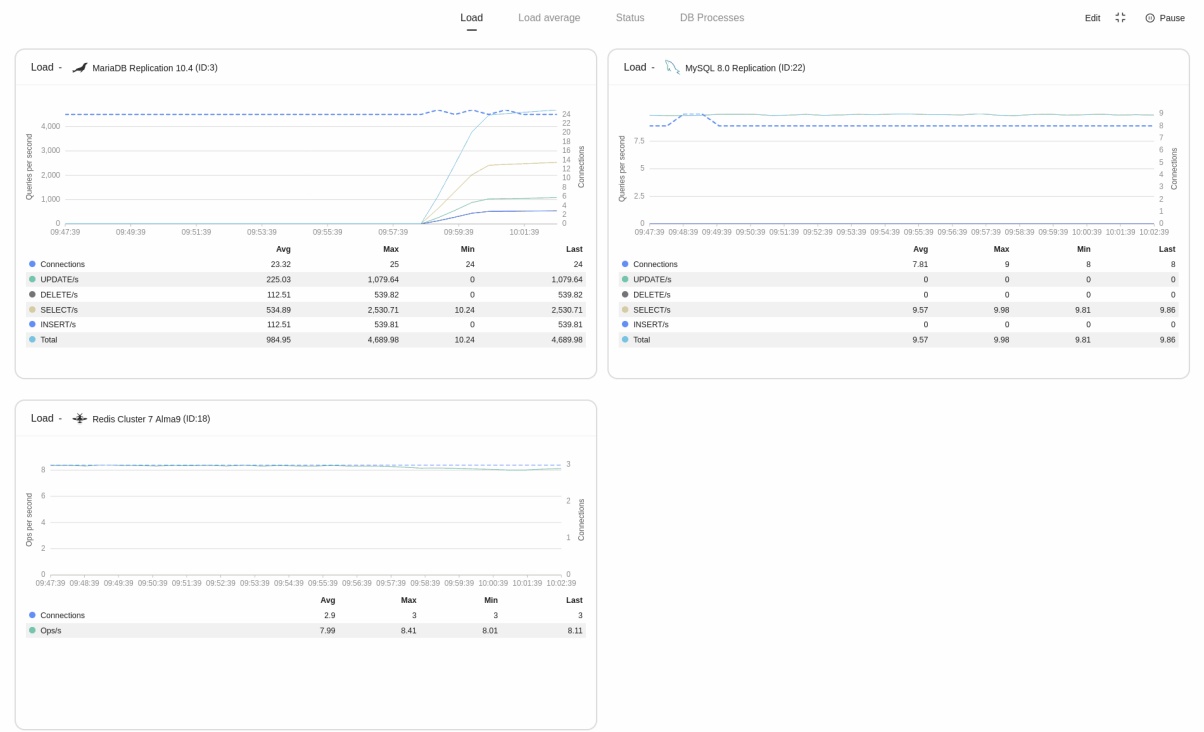
Attention

This feature will be enabled on all nodes of this cluster, cluster will be restarted to install the new plugin service.

## 2.35 Watchlist - Monitoring dashboards in kiosk-mode

Introduced in v2.3.2, Watchlist allows users to monitor the database clusters in a full-screen view (a.k.a kiosk-mode), suitable for displaying the monitoring dashboards on a dedicated screen in an operation center. Administrators can curate a watchlist based on topics, where ClusterControl will visualize the related monitoring panels based on the chosen topics and clusters in full screen view.

Every created watchlist has its own unique URL, where administrator can share the generated URL with other non-administrative users for active monitoring purposes (ClusterControl user login is required). Here is what it looks like in action (click to enlarge):




**Info**

This feature is still in beta and feedbacks are welcome. See *Give us feedback*.

## 2.35.1 Access Watchlist management section

### ClusterControl GUI

Log in to your ***ClusterControl GUI → Activity center → Watchlists***. This will bring up a panel where you can view and edit watchlist.

## 2.35.2 Add clusters directly into watchlist

Apart from using the *Watchlist management section* to manage which clusters belongs to which watchlists, you can directly add clusters into a watchlist from the *Clusters Actions*.

### ClusterControl GUI

1. To add a cluster into a new watchlist, log in to your ***ClusterControl GUI → Clusters → choose a cluster → Actions → Watchlists → Add to new watchlist***.
2. If you already have watchlists created, you can also add the chosen cluster into any existing watchlists by going to ***ClusterControl GUI → Clusters → choose a cluster → Actions → Watchlists → toggle On of the available watchlists***. This cluster will be added to all enabled watchlist view immediately. Toggling Off will remove the cluster from the corresponding watchlist.

### 2.35.3 Create watchlist

#### ClusterControl GUI

1. Click on the **Create watchlist** button. It will open a new **Add watchlist** panel.
2. Specify a unique name for the watchlist under the **Name** text field. This unique name will be used to generate a unique URL for the watchlist.
3. Choose one or more **Topics** from the list. Topics defines key metrics or dimensions users want to monitor in their watchlists — such as load, database growth, or status. These are visualized in dashboards and grid views when the watchlist is activated.
4. Choose the group of dashboards that will be displayed in one page, either by **Topic** or **Cluster**. If **Cluster**, all dashboards for the selected topics will be grouped per cluster in one page. If **Topic**, dashboards for all clusters will be grouped per topic in one page.
5. Choose a grid layout to align elements into columns and rows format (columns x rows). Depending on the screen size, you might want to choose the right grid layout to maximize the full-screen view. For a wide screen, 3x2 is usually better than 2x3.
6. Choose one or more clusters, or all clusters from the **Clusters** dropdown list.

### 2.35.4 Monitor in kiosk-mode

#### ClusterControl GUI

1. As an administrator, you can directly open the watchlist panel by going to ***Actions → Open***. A new tab will be opened specifically for the full-screen monitoring dashbards.
2. Click on the `[ ]` icon to enter the full-screen view. **Edit** to edit the watchlist settings and **Pause** to pause the animation in the current page.

**Tip**

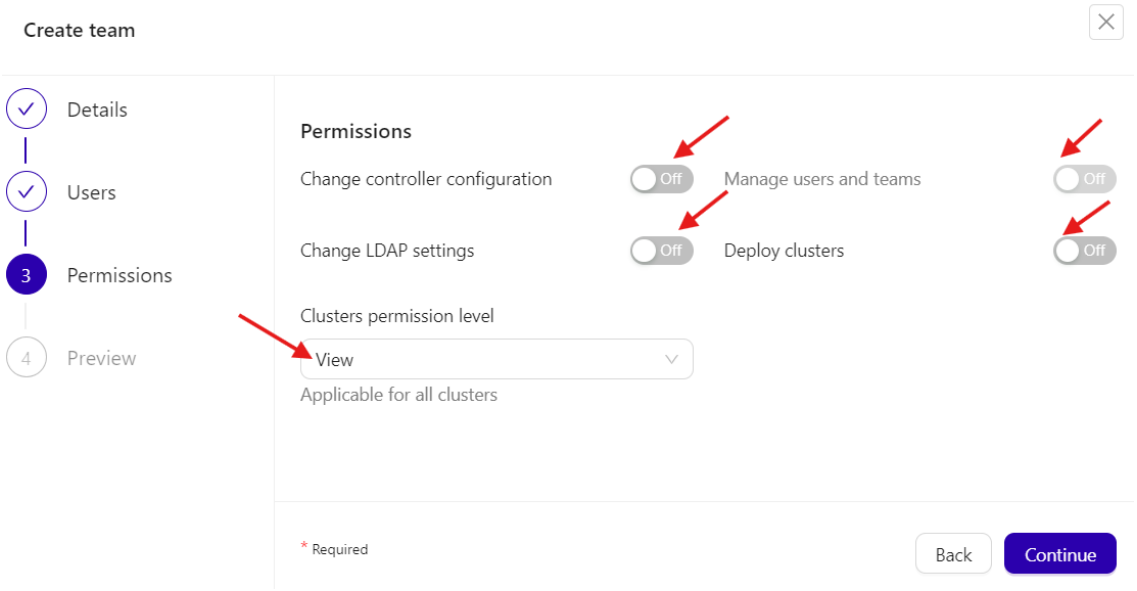
For non-administrative monitoring (commonly for NOC or SOC team), create a read-only user for this purpose. See *Share watchlist with other*.

### 2.35.5 Share watchlist with others

It is a common practice for the kiosk-mode monitoring to be passed over to the infrastructure monitoring team which has lower access level (can not edit the watchlist) to ClusterControl for monitoring-only purposes.

#### ClusterControl GUI

1. Create a read-only monitoring team called “monitor” by going to ***ClusterControl GUI → User management → Create user or team → Create team***. You may skip the **Users** part since we are creating a group first (will add the user later).
2. Under the **Permissions** section, toggle off all global permissions and choose “View” for cluster permission level, similar to the following example:




3. Create a user called "monitor" by going to ***ClusterControl GUI → User management → Create user or team → Create user***. Assign this user under group "monitor", which is a read-only group we created in the previous steps.
4. From the Watchlists tab, find the watchlist that you want to monitor in kiosk-mode ***Actions → Copy link***. Pass the "monitor" user credentials and the link to the operation team for monitoring. Use the provided link in the web browser and they are good to go.
5. Click on the `[ ]` icon to enter the full-screen view and **Pause** to pause the animation in the current page. Read-only user does not have the permission to edit the watchlist.

### 2.35.6 Delete watchlist

#### ClusterControl GUI

1. As an administrator, find the watchlist in the list and go to ***Actions → Delete***. Confirm the action and the watchlist will be deleted.

The Activity Center in ClusterControl is a centralized hub that provides a comprehensive overview of all system activities, events, and tasks. It consolidates critical information from alarms, jobs activity, and audit logs into a single, easy-to-navigate interface. This feature is designed to enhance visibility, streamline monitoring, and simplify troubleshooting by offering a unified view of what is happening within the database environment.

## 2.36 Alarms

Alarms indicate problems that could affect or degrade the cluster as a whole. Each alarm is categorized as:

- Cluster.
- Cluster recovery.
- Database health.
- Database performance.
- Host.
- Node.

- Network.

Accessing the alarms in ClusterControl can go to the following:

### ClusterControl GUI

1. Click on ***Actions → Details***, to get a detailed explanation of the problem, together with the recommended action (if available) to resolve the problem.
2. An alarm can be acknowledged by clicking on ***Actions → Mute***. When muted, no notification will be sent via email. However, an alarm cannot be deleted or dismissed.

### ClusterControl CLI

1. Check all the alarm through CLI

```
s9s alarm --list --long
```

2. Filter the alarms specific cluster.

```
s9s alarm --cluster-id=<clusterID> --list
```

3. Remove the alarm from CLI can be done by executing the following.

```
s9s alarm --delete --alarm-id=<alarmId>
```

The `alarmId` can be found from the list of alarm.

## 2.37 Jobs

Shows aggregated view of jobs that have been initiated and performed in a cluster by ClusterControl (e.g., promote replica, enable read-only, create/restore a backup, etc). Each job entry has a job title, job status, cluster name, the user who started the job, timestamp, and duration.

### ClusterControl GUI

1. Rollover on the ***Cluster*** entry will pop up a cluster information card summarizing the cluster status. Rollover on the ***When*** entry will pop up a tooltip containing the exact timestamp with timezone.
2. Click on the ***Actions*** ** → ** ***Details*** to see the job specification and messages. Inside the ***Job details*** popup, you have the ability to see the full job definition sent to the controller service for that particular job by clicking on the ***jobSpecs*** under the job's name. Underneath it is the full job messages in descending order (newer first) returned by the controller service. The ***Copy to clipboard*** button will copy the content of the job messages to the clipboard. The job messages are commonly the first thing to look for when troubleshooting an issue that is happening to a cluster. It is highly recommended to include the job messages when opening a support request from us.

### ClusterControl CLI

1. Check the history of the job

```
s9s jobs  --list --long
```

2. Dig the detailed logs of specific job

```
s9s job --log --job-id=<value>
```

The `job-id` can be found in the point `1` from the list of job are appear.

This allows administrators to monitor the progress of ongoing tasks, review the history of completed jobs, and troubleshoot any failed operations. The Jobs Activity feature is particularly useful for ensuring that automated tasks are executed as expected and for auditing changes made to the database environment.

## 2.38 Audit Log

Shows records of ClusterControl's events and changes:

- User authentication, cloud credentials, database configuration, and users changes
- CMON configuration changes and jobs
- ClusterControl user management changes (Users and ACLs), operational reports

Each log entry has the activity, type of activity, user, hostname or IP address (from where the user comes from), the corresponding cluster name, and timestamp.

### ClusterControl GUI

1. In the ClusterControl UI, go to ***Activity Center → Audit Log***, check the activity of the users.

The Audit Log is essential for maintaining accountability, ensuring security, and facilitating compliance with operational and regulatory standards.

Operational Reports offer a detailed view of the cluster's health, performance, and outstanding issues, serving as a comprehensive reporting tool for administrators. This feature is designed to facilitate the creation of stakeholder reports on the cluster's condition over a specified time frame.

Reports can be generated manually or scheduled for automatic creation, with the option to include email recipients during the setup process. Only users belonging to the admin team are authorized to create or access Operational Reports. Non-admin users are strictly prohibited from performing these actions.

## 2.39 Report templates

ClusterControl offers several predefined report templates to address various operational requirements. These templates are:

| Template | Description |
|---|---|
| Database Availability Reports | Details the uptime, downtime, and overall availability for each managed database cluster during the specified reporting period. |
| System Reports | Provides a summary of overall system health, covering node status, performance metrics, resource utilization, cluster events, and service availability over the reporting period. |
| Backup Reports | Summarizes the backup activities for each managed database cluster during the selected period. |
| Upgrade Reports | Offers an upgrade report for a managed cluster, including available package versions for the cluster (database packages, security packages, and libraries). |
| Database Growth Reports | Summarizes database growth over different timeframes, such as the last day, week, and month. |
| Schema Change Reports (MySQL-based systems only) | Provides reports for any modifications executed on the monitored databases. To utilize these predefined templates, you must configure the `schema_change_detection_address` and `schema_change_detection_databases` parameters in the ClusterControl settings. More information on these parameters can be found in the *references* section. |

**Note**

Note Operational reports are produced in HTML format, with each report corresponding to a specific cluster.

**Info**

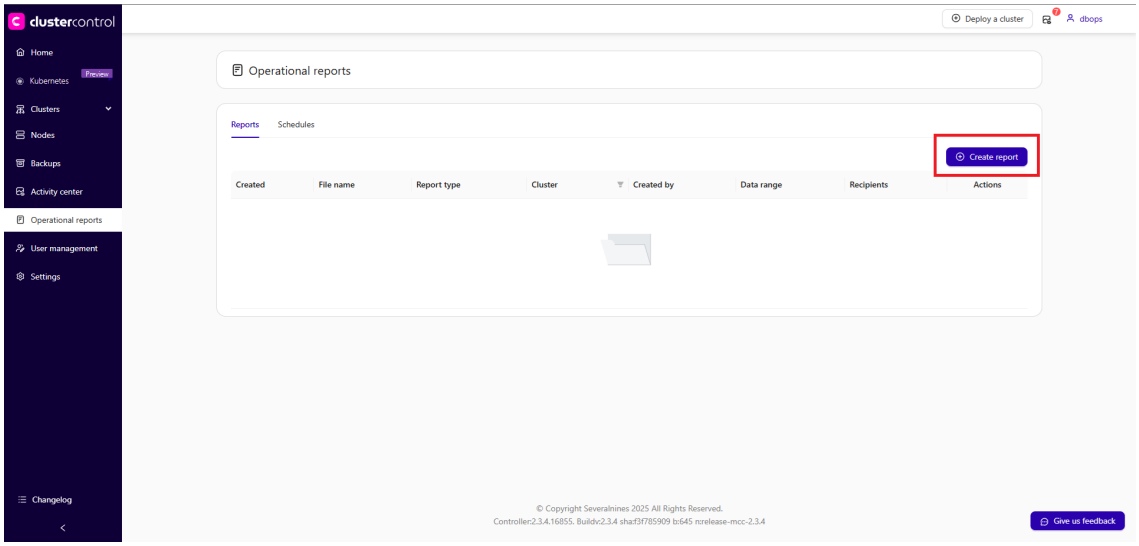
The reports generated may contain confidential information. It is essential to ensure that they are shared exclusively with the appropriate personnel.

## 2.40 Generating a report

The following steps show how to generate an operational report:

### ClusterControl GUI

1. Go to ***Operational Reports → Reports → Create report***:




2. Under the **Create report** section, specify the following information:
   - **Cluster**: Select a cluster from the dropdown.
   - **Type**: Select a report to be generated from the dropdown for this cluster.
   - **Data range**: Specify the report's data range in days.
   - **Recipients**: Specify the email recipients. This field supports adding multiple email addresses after an ++enter++. You can specify any email address, not limited to what has been configured in ClusterControl.
3. Click **Create** to generate the report.
4. The generated report will be listed under the **Reports** tab.

### ClusterControl CLI

1. List available template repots:

```
s9s report --list-templates
```

**Example**

```
$ s9s report --list-templates
report1
availability
backup
upgrade
schemachange
dbgrowth
```

2. Generate an availability report for cluster ID 37:

```
s9s report --create --cluster-id=37 --type='availability'
```

## 2.41 Listing and viewing reports

The following steps show how to list and view generated operational reports:

### ClusterControl GUI

1. Go to ***Operational Reports → Reports*** to see the list of generated reports.
2. Each report can be viewed or deleted by clicking the action button and choose **View** or **Delete**. When viewing via the GUI, the report will be opened in a new web browser window.

### ClusterControl CLI

1. List all generated operational reports:

```
s9s report --list
```

2. List all generated operational reports for cluster ID 37:

```
s9s report --list --cluster-id=37
```

3. View a generated report for cluster ID 37:

```
s9s report --cat --report-id=2 --cluster-id=37
```

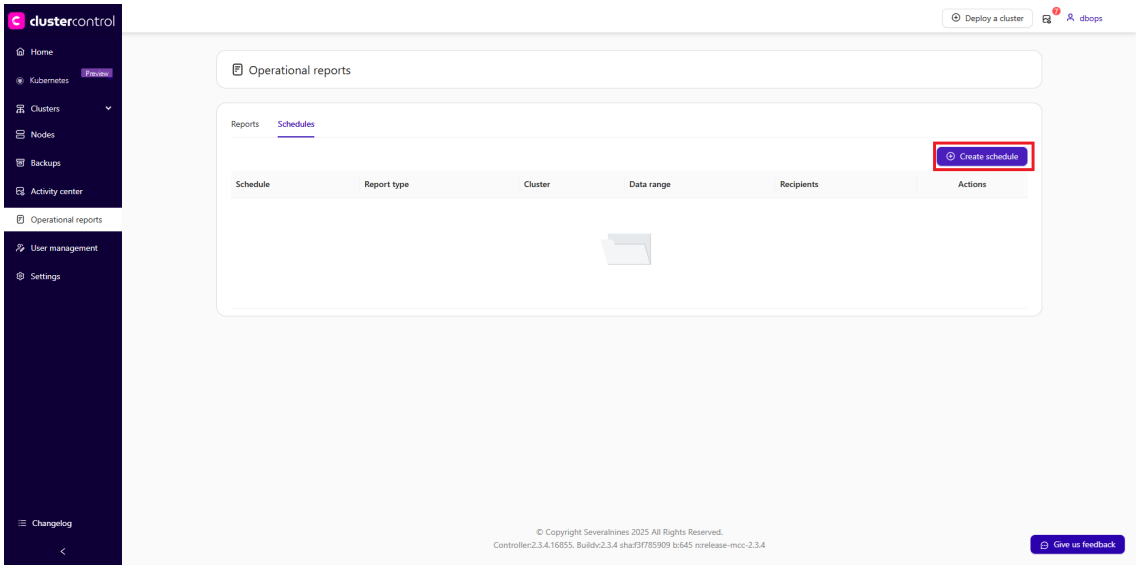
## 2.42 Scheduling a report

A report can be scheduled to be generated and delivered via email at specified intervals (e.g., daily, weekly, monthly).

The following steps show how to schedule an operational report:

### ClusterControl GUI

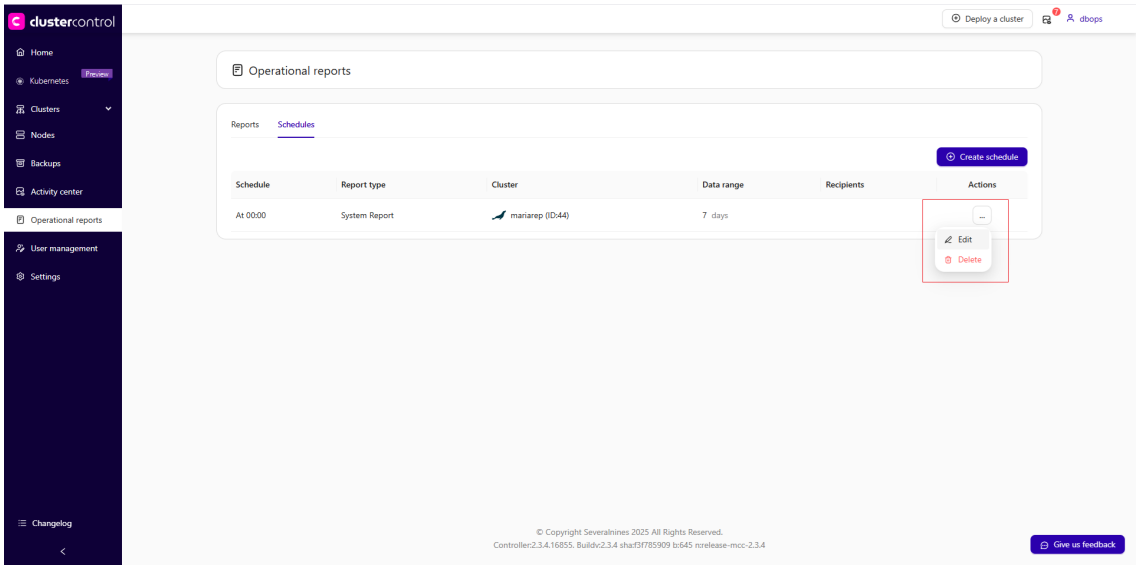
1. Go to ***Operational Reports → Schedules → Create schedule***:




2. Under the **Schedule report** section, specify the following information:

- **Cluster**: Select a cluster from the dropdown.
- **Type**: Select a report to be generated from the dropdown for this cluster.
- **Data range**: Specify the report's data range in days.
- **Recipients**: Specify the email recipients. This field supports adding multiple email addresses after an ++enter++. You can specify any email address, not limited to what has been configured in ClusterControl.
- **Set schedule**: Set the frequency or specific time to generate the report. The schedule can be configured to run minutely, hourly, daily, weekly, monthly, or yearly.

3. Click **Create** to save the schedule.
4. After the schedule is created, it can be edited or deleted by clicking the action button and choose **Edit** or **Delete**:




## 2.43 Management

This section highlights the key strategies and tools for managing your database clusters with ClusterControl. Learn how to simplify day-to-day operations, automate routine tasks, and maintain high availability and performance.

**See also**

- *Failover and Recovery →*
- *Cluster Management →*
- *Nodes Management →*
- *Configuration Management →*
- *Upgrades and Patches →*
- *Alerts and Notifications →*
- *Database Users and Privileges →*
- *Graceful Shutdown →*

- *Maintenance →*
- *Certificates →*

## 2.44 Failover and Recovery

ClusterControl is equipped with automated recovery algorithms designed to address common failures in database systems. It understands various database topologies and process management, enabling it to determine the optimal recovery strategy for a cluster.

ClusterControl offers two key recovery components:

1. **Node Recovery**: This attempts to restore a node to an operational state, covering issues such as a node being stopped outside of ClusterControl's awareness (e.g., via a user-initiated SSH stop command or an OOM process kill). See *Node Recovery*.
2. **Cluster Recovery**: This component aims to bring the entire cluster topology back to an operational state. See *Cluster Recovery*.

These two components are crucial for ensuring the highest possible service availability.

**Warning**

Attention

ClusterControl provides automatic failover and recovery for nodes within a single cluster. However, this functionality does not extend to replication between separate clusters. Consequently, automatic recovery is not supported for cluster-to-cluster replication. In the event that the primary cluster in Data Center A becomes unavailable, user intervention is required to promote the secondary cluster in Data Center B as the new active cluster. See *Create Replica Cluster*.

### 2.44.1 Node Recovery

ClusterControl ensures database node recovery from intermittent failures by monitoring process and connectivity. Similar to systemd, it keeps the database service running unless intentionally stopped via the ClusterControl GUI or CLI. The recovery includes all cluster-related services like ProxySQL, HAProxy, MaxScale, Keepalived, PgBouncer, Prometheus exporters, and garbd. Special attention to Prometheus exporters where ClusterControl will try to connect to the exporter's listening port for health check and verification. Thus, it's crucial to expose the exporter ports to ClusterControl and Prometheus server to make sure no false alarms during recovery. See *Firewall and security groups* for details.

Upon a node coming back online, ClusterControl re-establishes connection (using SSH persistent session) and performs necessary recovery actions:

1. It waits 30 seconds for systemd/init script to start monitored services/processes.
2. If services/processes remain down, ClusterControl automatically attempts to start the database service.
3. If ClusterControl is unable to recover the monitored services/processes, an alarm is raised.

**Note**

If a user initiates a database shutdown through ClusterControl GUI or CLI, ClusterControl will not automatically attempt to recover that specific node later. The user is expected to restart it either via the **Start Node** action in the ClusterControl GUI (or ClusterControl CLI), or by explicitly using an operating system command.

### 2.44.2 Cluster Recovery

ClusterControl excels at database recovery by understanding the topology and adhering to best practices. For clusters with built-in fault tolerance (like Galera, NDB Cluster, Redis Cluster, MongoDB Replicaset, and Elasticsearch), failover is automatic, managed by the database server through quorum calculations, heartbeats, and role switching. ClusterControl monitors this process, updating the **Topology** view and adjusting monitoring and management components for new roles, such as a new primary node in a replica set.

For database technologies lacking built-in fault tolerance and automatic recovery, such as MySQL/MariaDB Replication and PostgreSQL/TimescaleDB Streaming Replication, ClusterControl performs recovery procedures based on vendor best practices, as explained further down. Should recovery fail, user intervention is required, and an alarm notification will be issued.

In mixed or hybrid topologies, like an asynchronous replica attached to a Galera or NDB Cluster, ClusterControl will recover the node if cluster recovery is enabled.

While cluster recovery does not apply to standalone MySQL servers, it is still recommended to enable both node and cluster recoveries for this type of cluster in the ClusterControl GUI.

#### MySQL/MariaDB Replication

ClusterControl supports recovery of the following MySQL/MariaDB replication setup:

- Primary-replica with MySQL GTID
- Primary-replica with MariaDB GTID
- Primary-primary with MySQL GTID
- Primary-primary with MariaDB GTID
- Asynchronous replica attached to a Galera Cluster

ClusterControl will respect the following parameters when performing cluster recovery:

- `enable_cluster_autorecovery`
- `auto_manage_readonly`
- `repl_password`
- `repl_user`
- `replication_auto_rebuild_slave`
- `replication_check_binlog_filtration_bf_failover`
- `replication_check_external_bf_failover`
- `replication_failed_reslave_failover_script`
- `replication_failover_blacklist`

- replication_failover_events
- replication_failover_wait_to_apply_timeout
- replication_failover_whitelist
- replication_onfail_failover_script
- replication_post_failover_script
- replication_post_switchover_script
- replication_post_unsuccessful_failover_script
- replication_pre_failover_script
- replication_pre_switchover_script
- replication_skip_apply_missing_txs
- replication_stop_on_error
- replication_failover_dead_source_grace

For more details on each of the parameters, see *Configuration Options*.

ClusterControl adheres to specific rules when monitoring and managing primary-replica replication:

- All nodes, regardless of their role, are started with read_only=ON and super_read_only=ON.
- Only one primary (read_only=OFF) is permitted at any given time.
- The MySQL variable report_host is used to map the topology.
- If multiple nodes have read_only=OFF concurrently, ClusterControl automatically sets read_only=ON on all of them to prevent accidental writes. Manual intervention is then required to designate the actual primary by disabling read-only.

In the event of an active primary going offline, ClusterControl attempts a primary failover with the following sequence:

1. An alarm is raised after 3 seconds of the primary being unreachable from multiple nodes including ClusterControl node, replica nodes or load balancer nodes.
2. Replica availability is checked, ensuring at least one replica is reachable by ClusterControl.
3. A replica is chosen as a candidate for promotion to primary.
4. If GTID is enabled, ClusterControl calculates the probability of errant transactions.
5. If no errant transactions are detected, the chosen replica is promoted as the new primary.
6. A replication user is created and granted for use by the replicas.
7. All replicas previously pointing to the old primary are reconfigured to point to the newly promoted primary.
8. The replicas are started.
9. Logs are flushed on all nodes.

If the replica promotion fails, the recovery job is aborted by ClusterControl. To re-trigger the recovery job, user intervention or a ClusterControl service restart is necessary.

When the old primary becomes available again, it is started in read-only mode and is no longer part of the replication. User intervention is required to reincorporate the old primary into the replication by restaging it as a replica, or to remove the node from the cluster using ClusterControl.

### PostgreSQL/TimescaleDB Streaming Replication

ClusterControl supports recovery of the following PostgreSQL replication setup:

- PostgreSQL Streaming Replication
- TimescaleDB Streaming Replication

ClusterControl will respect the following parameters when performing cluster recovery:

- `enable_cluster_autorecovery`
- `repl_password`
- `repl_user`
- `replication_auto_rebuild_slave`
- `replication_failover_whitelist`
- `replication_failover_blacklist`

For more details on each of the parameters, see *Configuration Options*.

ClusterControl will obey the following rules for managing and monitoring a PostgreSQL streaming replication setup:

- `wal_level` is set to `replica` (or `hot_standby` depending on the PostgreSQL version).
- The parameter `archive_mode` is set to ON on the primary.
- Set `recovery.conf` or `recovery.signal` file on the replica nodes, which turns the node into a hot standby with read-only enabled.

In case the active primary goes down, ClusterControl will attempt to perform the cluster recovery in the following order:

1. After 10 seconds of primary unreachability, ClusterControl will raise an alarm.
2. Following a 10-second graceful waiting period, ClusterControl will initiate the primary failover job.
3. Sample the `replayLocation` and `receiveLocation` on all available nodes to identify the most advanced node.
4. Promote the most advanced node as the new primary.
5. Stop replicas.
6. Verify the synchronization state with `pg_rewind`.
7. Restarting replicas, pointing to the new primary.

If the replica promotion fails, ClusterControl will abort the recovery job. In such cases, user intervention or a ClusterControl service restart is required to re-trigger the recovery job.

**Warning**

Attention

When the old primary becomes available again, it will be forcefully shut down and will not rejoin the replication setup. User intervention is necessary. See further down.

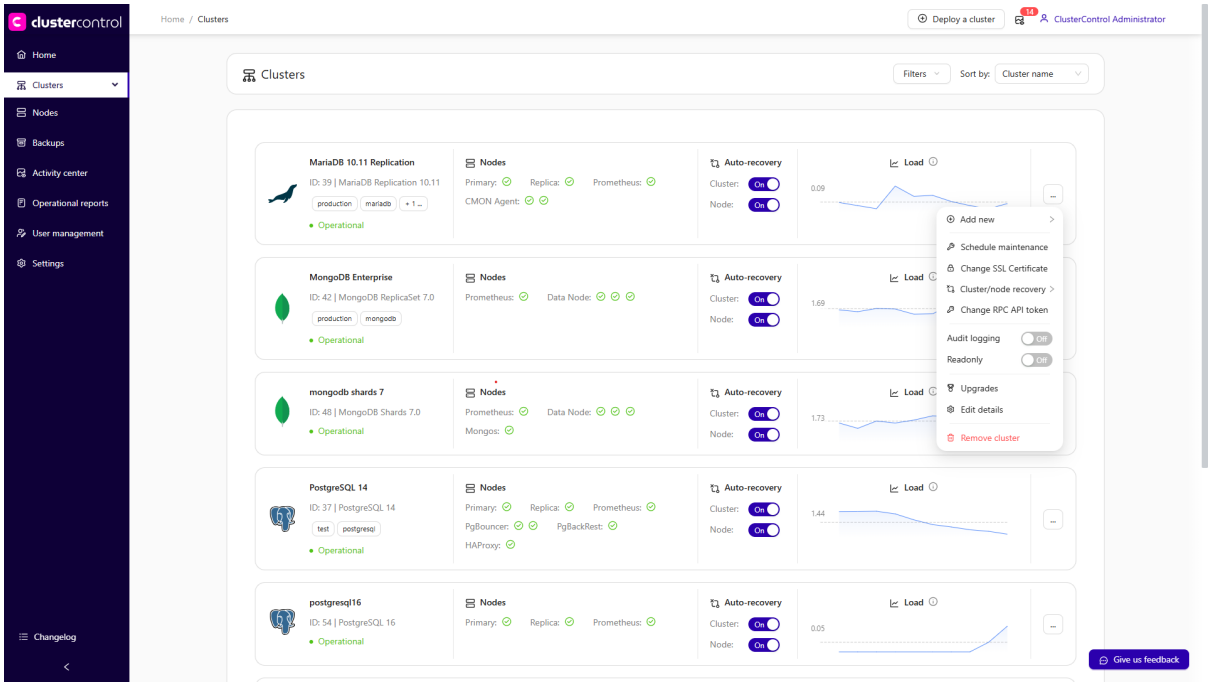
Upon the old primary coming back online, if the PostgreSQL service is running, ClusterControl will forcefully shut down the PostgreSQL service. This action prevents accidental writes, as the server would otherwise start without a recovery file (`recovery.conf` or `recovery.signal`) and thus be writable. You should expect the following lines will appear in `postgresql-{day}.log`:

```
2019-11-27 05:06:10.091 UTC [2392] LOG: database system is ready to accept␣
↪connections
2019-11-27 05:06:27.696 UTC [2392] LOG: received fast shutdown request
2019-11-27 05:06:27.700 UTC [2392] LOG: aborting any active transactions
2019-11-27 05:06:27.703 UTC [2766] FATAL: terminating connection due to␣
↪administrator command
2019-11-27 05:06:27.704 UTC [2758] FATAL: terminating connection due to␣
↪administrator command
2019-11-27 05:06:27.709 UTC [2392] LOG: background worker "logical␣
↪replication launcher" (PID 2419) exited with exit code 1
2019-11-27 05:06:27.709 UTC [2414] LOG: shutting down
2019-11-27 05:06:27.735 UTC [2392] LOG: database system is shut down
```

The PostgreSQL was started after the server was back online around 05:06:10 but ClusterControl performs a fast shutdown 17 seconds after that around 05:06:27. If this is something that you would not want it to be, you can disable node recovery for this cluster momentarily.

## 2.45 Cluster Management

This feature provides information and management options for database clusters managed by ClusterControl. Clicking on the ***Clusters*** (sidebar menu) will list out all database clusters on the main panel, each of the entries has a real-time summary of the cluster, together with the dropdown management menu (see *Cluster Actions*):



Each entry will have the following details:

- **Cluster name**: Configurable during import or deployment, also via the s9s command line.
- **Cluster ID**: Immutable ID assigned by ClusterControl after it is registered.
- **Cluster type**: The database cluster type, recognized by ClusterControl. See *Supported Databases*.
- **Nodes**: All nodes grouped under this particular cluster.
- **Auto recovery**: The ClusterControl automatic recovery settings. See *Failover and Recovery*.
- **Load**: Last 5 minutes of cluster load average.
- **…**: Every supported database cluster has its own set of dropdown menus. See *Cluster Actions*.

Choose a database cluster from the expandable list on the left-side menu to drill down on the cluster-specific features. The navigation breadcrumbs (top panel) will reflect with the corresponding cluster when selecting a cluster.

### 2.45.1 Cluster Actions

Provides shortcuts to the main cluster functionality. Each database cluster has its own set of functionality as described below:

### MySQL/MariaDB/Percona (replication/standalone)

| Feature | Description |
|---|---|
| **Add new → Replication node** | Adds a new or existing replica database node into the cluster. The new replica will automatically join and synchronize with the rest of the cluster. See *Add Replication Node*. |
| **Add new → Load balancer** | Adds a new or existing load balancer into the cluster. See *Create Load Balancer*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Encryption settings → Change SSL Certificate** | Opens the SSL certificate management wizard which allows you to create or choose an existing certificate for database client-server encryption |
| **Encryption settings → SSL encryption** | Toggles On to enable client-server SSL encryption. This action will restart the whole cluster, one node at a time Toggles Off to disable client-server SSL encryption. This action will restart the whole cluster, one node at a time |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Audit logging** | Enable audit logging (MariaDB-based clusters only) using policy-based monitoring and logging of connection and query activity. This feature will enable audit logging on all nodes in the cluster. See *Audit Logging*. |
| **Readonly** | Toggles On to enable cluster-wide read-only. ClusterControl will configure `read_only=ON` and `super_read_only=ON` (except MariaDB) on all database nodes in the cluster Toggles Off to disable cluster-wide read-only |
| **Upgrades** | Performs minor software upgrades for the database and load balancer software of this cluster, for example from 8.0.x to 8.0.y, in a rolling upgrade fashion. ClusterControl will perform the software upgrade based on what is available on the package repository for the particular vendor. See *Upgrades* |
| **Create database** | Create a new database schema on the primary node. For MySQL, the database name can contain letters, numbers, `$` and _ symbols. Maximum length is 64 characters. |
| **Edit details** | Edit the cluster's name and tags. |
| **Remove cluster** | Removing the cluster will delete all metadata we have on it. The database nodes will not be affected and will continue to operate as normal One has to explicitly type "REMOVE" in the text field before the cluster removal job is triggered. The cluster can be imported again in the future with fresh new metadata and configuration |

### Galera Cluster (Percona XtraDB/MariaDB)

| Feature | Description |
|---|---|
| **Add new → Node** | Adds a new or existing database node into the cluster. You can scale out your cluster by adding more database nodes. The new node will automatically join and synchronize with the rest of the cluster. See *Add Node*. |
| **Add new → Replication node** | Adds a new or existing replica database node into the cluster. The new replica will automatically join and synchronize with the rest of the cluster. See *Add Replication Node*. |
| **Add new → Load balancer** | Adds a new or existing load balancer into the cluster. See *Create Load Balancer*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Encryption settings → Change SSL Certificate** | Opens the SSL certificate management wizard which allows you to create or choose an existing certificate for database client-server encryption |
| **Encryption settings → Change Galera SSL Certificate** | Opens the SSL certificate management wizard which allows you to create or choose an existing certificate for Galera cluster nodes communication encryption |
| **Encryption settings → Galera SSL encryption** | Toggles On to enable Galera SSL encryption. This action will restart the whole cluster, one node at a time Toggles Off to disable Galera SSL encryption. This action will restart the whole cluster, one node at a time |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Readonly** | Toggles On to enable cluster-wide read-only. ClusterControl will configure `read_only=ON` and `super_read_only=ON` (except MariaDB) on all database nodes in the cluster Toggles Off to disable cluster-wide read-only |
| **Clone/Replicate the cluster → Clone** | This feature has been deprecated. Use *Create cluster from backup* instead. |

### PostgreSQL/TimescaleDB (streaming replication/standalone)

| Feature | Description |
|---|---|
| **Add new → Replication node** | Adds a new or existing replica database node into the cluster. The new replica will automatically join and synchronize with the rest of the cluster. See *Add Replication Node*. |
| **Add new → Load balancer** | Adds a new or existing load balancer into the cluster. See *Create Load Balancer*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Encryption settings → Change SSL Certificate** | Opens the SSL certificate management wizard which allows you to create or choose an existing certificate for database client-server encryption |
| **Encryption settings → SSL encryption** | Toggles On to enable client-server SSL encryption. This action will restart the whole cluster, one node at a time Toggles Off to disable client-server SSL encryption. This action will restart the whole cluster, one node at a time |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Audit logging** | Enable audit logging using policy-based monitoring and logging of connection and query activity. This feature will enable audit logging on all nodes in the cluster. See *Audit Logging*. |
| **Create replica cluster** | Create a new cluster that replicates this cluster. See *Create Replica Cluster*. |

### PostgreSQL Logical Replication

| Feature | Description |
|---|---|
| **Actions → Edit details** | Edit the cluster's name and tags. |
| **Actions → Remove cluster** | Remove the logical replication grouping for all clusters connected to it. The PostgreSQL database nodes or streaming replication cluster will be ungrouped from the logical binding and will continue to operate as a normal standalone node or streaming replication cluster After the removal, the WAL level will remain as logical, which is usable for streaming replication as well (despite the WAL size will be bigger if compared to replica or hot_standby) One must explicitly type "REMOVE" in the text field before triggering the cluster removal job |
| | *Pubs & Subs → Create publication* |
| **Publisher cluster** | Choose a database cluster from the dropdown list as the publisher cluster. |
| **Datab** | Choose a database from the list. The target database and its corresponding objects like tables must be created first in the publisher cluster. This list will be auto-populated by ClusterControl depending on the ***Publisher cluster*** |
| **Publication name** | Specify a publication name |
| **Include all tables** | Toggle on to include all tables under the chosen database Toggle off will open a dropdown editor where you can type the table name and press ++enter++. Multiple values are supported |
| | *Pubs & Subs → Create subscription* |
| **Subscribe cluster** | Choose a database cluster from the dropdown list as the subscriber cluster |
| **Subscription name** | Specify the subscription name |
| **Publication** | Choose an existing publication from the list |
| **Datab** | Choose a database from the list. The target database and its corresponding objects like tables must be created first in the subscriber cluster. This list will be auto-populated by ClusterControl depending on the Subscriber cluster |
| **copy data** | Toggle on to copy pre-existing data in the publications that are being subscribed to when the replication starts Toggle off will skip the initial snapshot stage. It is recommended for the user to manually perform the initial syncing before proceeding with this option |
| **Origin** | Make sure the subscription will only replicate changes that do not have an associated origin (i.e., changes made locally), while Any means the subscription will replicate all changes regardless of their origin, including those already replicated from other nodes This operation establishes a new logical cluster to manage publications and subscriptions. To enable bi-directional repli- |

**Warning**

Attention

For PostgreSQL logical replication, setting `origin=any` can lead to a potential infinite loop in bi-directional replication setups, as changes replicated from one node can be sent back to the originating node. To avoid this, users should carefully configure subscriptions and use `origin=none` to prevent such behavior.

### MongoDB

| Feature | Description |
|---|---|
| **Add node** | Scale the current MongoDB Replica Set or Sharded Cluster deployment by adding a single shard, Mongos, or config server. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Convert to shard** | Only for MongoDB Replica Set. Converts an existing MongoDB replica set to a sharded cluster by adding Mongos and config servers into the setup |
| **SSL encryption** | Toggles On to enable client-server SSL encryption. This action will restart the whole cluster, one node at a time Toggles Off to disable client-server SSL encryption. This action will restart the whole cluster, one node at a time |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery* |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Upgrades** | Performs minor software upgrades for the database and load balancer software of this cluster, in a rolling upgrade fashion. ClusterControl will perform the software upgrade based on what is available on the package repository for the particular vendor. See *Upgrades* |
| **Edit details** | Edit the cluster's name and tags. |
| **Remove cluster** | Removing the cluster will delete all metadata we have on it. The database nodes will not be affected and will continue to operate as normal One has to explicitly type "REMOVE" in the text field before the cluster removal job is triggered. The cluster can be imported again in the future with fresh new metadata and configuration |

### Redis

| Feature | Description |
|---|---|
| **Add node** | Adds a new or existing node into the cluster. You can scale out your cluster by adding more database nodes. The new node will automatically join and synchronize with the rest of the cluster. See *Add Node*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Upgrades** | Performs minor software upgrades for the database and load balancer software of this cluster, for example from 7.0.x to 7.0.y, in a rolling upgrade fashion. ClusterControl will perform the software upgrade based on what is available on the package repository for the particular vendor. See *Upgrades* |
| **Edit details** | Edit the cluster's name and tags. |
| **Remove cluster** | Removing the cluster will delete all metadata we have on it. The database nodes will not be affected and will continue to operate as normal One has to explicitly type "REMOVE" in the text field before the cluster removal job is triggered. The cluster can be imported again in the future with fresh new metadata and configuration |

### Microsoft SQL Server

| Feature | Description |
|---|---|
| **Add node** | Adds a new or existing node into the cluster. You can scale out your cluster by adding more database nodes. The new node will automatically join and synchronize with the rest of the cluster. See *Add Node*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Upgrades** | This feature is not available yet for MSSQL-based clusters |
| **Edit details** | Edit the cluster's name and tags. |
| **Remove cluster** | Removing the cluster will delete all metadata we have on it. The database nodes will not be affected and will continue to operate as normal One has to explicitly type "REMOVE" in the text field before the cluster removal job is triggered. The cluster can be imported again in the future with fresh new metadata and configuration |

### Elasticsearch

| Feature | Description |
|---|---|
| **Add node** | Adds a new Elasticsearch master, data, coordinator, or master-data node into the cluster. The new master or data node will automatically join and synchronize with the rest of the cluster. See *Add Node*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Upgrades** | This feature is not available yet for Elasticsearch-based clusters |
| **Edit details** | Edit the cluster's name and tags. |
| **Remove cluster** | Removing the cluster will delete all metadata we have on it. The database nodes will not be affected and will continue to operate as normal One has to explicitly type "REMOVE" in the text field before the cluster removal job is triggered. The cluster can be imported again in the future with fresh new metadata and configuration |

### Valkey

| Feature | Description |
|---|---|
| **Add node** | Adds a new or existing node into the cluster. You can scale out your cluster by adding more database nodes. The new node will automatically join and synchronize with the rest of the cluster. See *Add Node*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Upgrades** | Performs minor software upgrades for the database and load balancer software of this cluster, for example from 7.0.x to 7.0.y, in a rolling upgrade fashion. ClusterControl will perform the software upgrade based on what is available on the package repository for the particular vendor. See *Upgrades*. |
| **Edit details** | Edit the cluster's name and tags. |
| **Remove cluster** | Removing the cluster will delete all metadata we have on it. The database nodes will not be affected and will continue to operate as normal. One has to explicitly type "REMOVE" in the text field before the cluster removal job is triggered. The cluster can be imported again in the future with fresh new metadata and configuration. |

### ClickHouse

| Feature | Description |
|---|---|
| **Add new → Data node** | Adds a new ClickHouse data node to the cluster, either as a new replica in an existing shard or as a new shard. See *Add Database Node → ClickHouse*. |
| **Add new → Keeper node** | Adds a new node to the ClickHouse Keeper ensemble, growing it for quorum. See *Add Database Node → ClickHouse*. |
| **Schedule maintenance** | Schedules a cluster-wide maintenance mode, where ClusterControl will skip raising alarms and notifications while the mode is active All nodes in the cluster (regardless of the role) will be marked as under maintenance. A global banner will appear if there is upcoming maintenance for the corresponding cluster |
| **Cluster/node recovery → Cluster recovery** | Disables or enables cluster recovery If enabled, ClusterControl will attempt to recover the cluster if necessary. See *Cluster Recovery*. |
| **Cluster/node recovery → Node recovery** | Disables or enables node recovery If enabled, ClusterControl will attempt to recover the node if necessary. See *Node Recovery*. |
| **Change RPC API token** | Serves as the authentication string by ClusterControl UI to connect to the CMON RPC interface. Each cluster has its unique token |
| **Upgrades** | Performs software upgrades for the ClickHouse cluster in a rolling upgrade fashion, based on what is available on the package repository. See *Upgrades*. |
| **Edit details** | Edit the cluster's name and tags. |
| **Remove cluster** | Removing the cluster will delete all metadata we have on it. The database nodes will not be affected and will continue to operate as normal. One has to explicitly type "REMOVE" in the text field before the cluster removal job is triggered. The cluster can be imported again in the future with fresh new metadata and configuration. |

**Warning**

Not supported for ClickHouse

The following actions, available for other cluster types, are not available for ClickHouse:

- **Readonly** (cluster-wide read-only toggle): ClickHouse does not have a MySQL-style cluster-wide read-only mode.
- **Create database**: create databases directly on the ClickHouse cluster instead.
- **Encryption settings → Change/Enable SSL Certificate**: ClusterControl deploys ClickHouse with SSL/TLS enforced by default (plaintext ports are disabled), so there is no separate toggle to turn it on or off.

### 2.45.2 Cluster-specific actions

#### Bootstrap cluster

Bootstrap cluster refers to the process of bootstrapping or initializing a new Galera Cluster. When you're setting up a new Galera Cluster or performing a full backup restoration, you typically need to bootstrap it, which involves designating one node as the initial primary component of the cluster. This bootstrap node serves as the starting point for the cluster, and other nodes will then join this initial node to form the cluster.

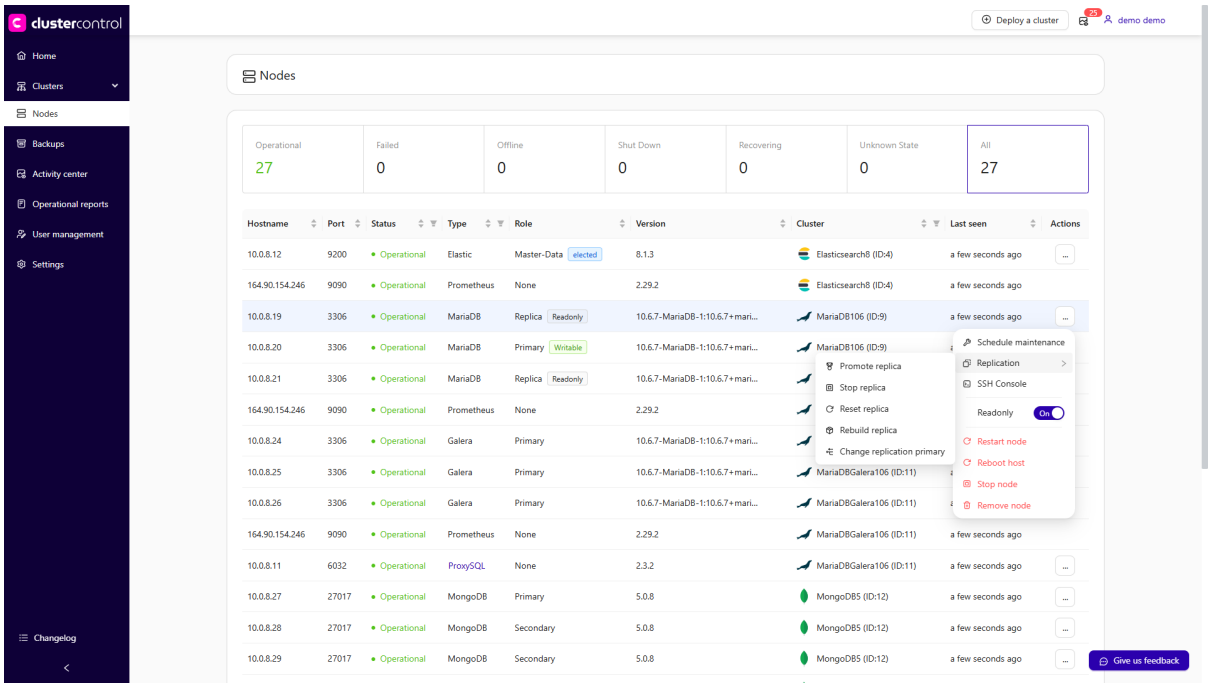
The following actions will be performed:

1. The cluster will be initialized from the selected node.
2. All nodes will be stopped unless they are already stopped.
3. When the bootstrap command is successful, the selected node will be Synced. The rest of the nodes will be started as joiners, one node at a time.

| Field | Description |
|---|---|
| Bootstrap node | Choose a database node from the list to be the reference node. If you are unsure, you may choose Auto Select from the dropdown where ClusterControl will determine the most suitable node to bootstrap based on the Galera node's sequence number. |
| Graceful shutdown timeout (in seconds) | ClusterControl will wait for this timeout to finish before forcing the node to stop. |
| Force stop the nodes (after shutdown time) | Toggles On to force stop the database node if it takes longer than the shutdown time, specified in Graceful shutdown timeout (in seconds), defaulting to 1800 seconds (30 minutes). Commonly useful in the case of forcing a database cluster to bootstrap when database locking or long-running transactions are happening. |
| Clear MySQL datadir on Joining nodes | Toggles On to clear the MySQL data directory on the joining nodes. This is often necessary after a restoration. |

## 2.46 Nodes Management

This feature provides information and management options for all nodes managed by ClusterControl. The Nodes view lists every managed node with a real-time summary of its status, last-seen timestamp, and a dropdown of available management actions.




Each entry displays the following details:

| Column | Description |
|---|---|
| **Hostname** | The node name recognized by ClusterControl. If you used an IP address during cluster deployment or import, the IP address is shown instead. |
| **Port** | The active port for the corresponding host. Every managed node is identified by a `host:port` pair. |
| **Status** | The node's current operational status. |
| **Type** | The node's cluster type. |
| **Role** | The node's role within its cluster (e.g., primary, replica, arbiter). |
| **Version** | The database or service version running on the node. |
| **Cluster** | Rolling over the cluster name displays a pop-up with cluster details. |
| **Last seen** | The timestamp of the last successful check-in between ClusterControl and the node. |
| **Actions** | A context-sensitive dropdown menu with management actions available for the node. See *Node Actions* below. |

### 2.46.1 Access the Nodes section

#### ClusterControl GUI

Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***. This will bring up the Nodes panel listing all managed nodes for the selected cluster.

#### ClusterControl CLI

Use the `s9s node` command to interact with nodes. Listing all nodes across all clusters:

```
s9s node --list --long
```

Listing nodes for a specific cluster (e.g., cluster ID 36):

```
s9s node --list --long --cluster-id=36
```

---

### 2.46.2 Node Actions

The **Actions** dropdown on each node row provides shortcuts to node-level operations. The available actions differ depending on the cluster type and the node's role. The sections below describe each action and how to perform it across the supported interfaces.

#### Restart node

Restarts the active monitored process of the selected node (e.g., the MySQL service, the HAProxy process). This is a service restart, not a system reboot. The action is only available when the service is currently running. The node enters maintenance mode automatically until the service is fully restarted.

> **Note**
>
> Note You can configure the **Graceful shutdown timeout** (default: 1800 seconds). If the node has not stopped within this period, you can toggle **Force stop the nodes (after shutdown time)** to send a `SIGKILL` signal and force the process down.

**Applies to:** MySQL/MariaDB, Galera Cluster, Group Replication, PostgreSQL/TimescaleDB, MongoDB, Redis, Microsoft SQL Server, Elasticsearch, Valkey, ClickHouse, HAProxy, PGBouncer, Keepalived, ProxySQL, Prometheus

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the node you want to restart.
3. Click **Actions (…) → Restart node**.
4. Optionally, adjust the **Graceful shutdown timeout** or toggle **Force stop the nodes (after shutdown time)**.
5. Confirm the action to proceed.

### ClusterControl CLI

Restart a MySQL node (e.g., 192.168.40.30:3306) in cluster ID 36:

```
s9s node \
    --restart \
    --cluster-id=36 \
    --nodes=1192.168.40.30:3306 \
    --wait
```

Restart a PostgreSQL node (e.g., 192.168.40.40:5432) in cluster ID 37:

```
s9s node \
    --restart \
    --cluster-id=37 \
    --nodes=192.168.40.40:5432 \
    --wait
```

---

## Reboot host

Initiates a full operating system reboot of the selected node. Once initiated, ClusterControl monitors the reboot progress every 5 seconds for up to 10 minutes (600 seconds). If the node does not come back online within this window, the operation is declared failed.

**Applies to:** MySQL/MariaDB, Galera Cluster, Group Replication, PostgreSQL/TimescaleDB, MongoDB, Redis, Microsoft SQL Server, Elasticsearch, Valkey, ClickHouse, HAProxy, PGBouncer, Keepalived, ProxySQL, Prometheus

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the node you want to reboot.
3. Click **Actions (…) → Reboot host**.
4. Confirm the action to proceed. ClusterControl will initiate the OS-level reboot and track its progress.

### ClusterControl CLI

There is no dedicated `s9s` command for a full host reboot. Use an SSH-based approach or leverage an OS-level automation tool to reboot the host. You can monitor node availability afterwards with:

```
s9s node --list --long --cluster-id=<cluster_id>
```

---

## Stop/Start node

Stops or starts the managed service on the selected node. For database nodes, ClusterControl stops or starts the database service bound to the given `host:port`. For load balancer nodes (HAProxy, ProxySQL), the load balancer process is stopped or started instead.

**Note**

Note By default, a graceful shutdown is attempted for 1800 seconds (30 minutes). Toggle **Force stop the nodes (after shutdown time)** to send `SIGKILL` if the node does not stop within this window. No data is lost during a graceful stop.

**Applies to:** MySQL/MariaDB, Galera Cluster, Group Replication, PostgreSQL/TimescaleDB, MongoDB, Redis, Microsoft SQL Server, Elasticsearch, Valkey, ClickHouse, HAProxy, PGBouncer, Keepalived, ProxySQL, Prometheus

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the target node.
3. Click **Actions (…) → Stop node** (or **Start node** if the node is currently stopped).
4. Optionally, configure the **Graceful shutdown timeout** or enable **Force stop the nodes (after shutdown time)**.
5. Confirm the action to proceed.

### ClusterControl CLI

Stop a MySQL node (e.g., 192.168.40.30:3306) in cluster ID 36:

```
s9s node \
    --stop \
    --cluster-id=36 \
    --nodes=192.168.40.30:3306 \
    --wait
```

Start a previously stopped MySQL node:

```
s9s node \
    --start \
    --cluster-id=36 \
    --nodes=192.168.40.30:3306 \
    --wait
```

Stop a PostgreSQL node (e.g., 192.168.40.40:54 32) in cluster ID 37:

```
s9s node \
    --stop \
    --cluster-id=37 \
    --nodes=192.168.40.40:5432 \
    --wait
```

### Remove node

Removes a node from ClusterControl management. Depending on the selected removal option, this can also stop the service or fully uninstall the software.

**Warning**

Warning!

Removing a node is a destructive operation with respect to ClusterControl's awareness of that node. Choose the appropriate removal option carefully.

**Removal options:**

| Option | Behavior |
|---|---|
| **Keep service running** | The node is unregistered from ClusterControl but the database or load balancer service continues running on the host. |
| **Stop service and keep files untouched** | The node is unregistered and the service is stopped. Data files and configuration files are preserved on the host. |
| **Stop and uninstall service** | The node is unregistered, the service is stopped, and all related software packages are removed using the system package manager. |

**Applies to:** MySQL/MariaDB, Galera Cluster, Group Replication, PostgreSQL/TimescaleDB, MongoDB, Redis, Microsoft SQL Server, Elasticsearch, Valkey, ClickHouse, HAProxy, PGBouncer, Keepalived, ProxySQL, Prometheus

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the node you want to remove.
3. Click **Actions (…) → Remove node**.
4. Select the desired **removal option** from the panel.
5. Confirm the action to proceed.

#### ClusterControl CLI

Remove node 192.168.40.30:3306 from cluster ID 36 (the service will be stopped and uninstalled):

```
s9s node \
    --remove \
    --cluster-id=36 \
    --nodes=192.168.40.30:3306 \
    --wait
```

#### SSH Console

Opens a web-based SSH terminal in a new browser window, allowing you to execute shell commands directly on the node from your browser. The session authenticates as the configured `os_user`.

**Applies to:** MySQL/MariaDB, Galera Cluster, Group Replication, PostgreSQL/TimescaleDB, MongoDB, Redis, Microsoft SQL Server, Elasticsearch, Valkey, ClickHouse, HAProxy, PGBouncer, Keepalived, ProxySQL, Prometheus

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the target node.
3. Click **Actions (…) → SSH Console**. A new browser tab or window will open with a terminal session on the selected host.

---

### 2.46.3 Replication actions

The following actions are specific to nodes that participate in replication topologies. Availability depends on the cluster type and the node's role.

#### Promote replica

Promotes the selected replica to become the new primary node. If the current primary is healthy, stop application traffic before promoting a replica to prevent data loss. Connections to the existing primary are terminated after a 10-second grace period.

> **Warning**
>
> Warning!
>
> Toggle **Force promotion of replica** to ignore errors (such as errant transactions or long-running queries) during promotion. Without this option, ClusterControl aborts the failover if an errant transaction is detected on the replica.

**Applies to:** MySQL/MariaDB (replication), PostgreSQL/TimescaleDB, Microsoft SQL Server

#### ClusterControl GUI

**MySQL/MariaDB and PostgreSQL**

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the replica node you want to promote.
3. Click **Actions (…) → Replication → Promote replica**.
4. Optionally, enable **Force promotion of replica** to bypass error checks.
5. Confirm the action to proceed.

**Redis / Valkey / Microsoft SQL Server**

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.

2. Locate the replica node.
3. Click **Actions (…) → Promote replica**.
4. Confirm the action to proceed.

### ClusterControl CLI

Promote a MySQL replica (192.168.40.31:3306) in cluster ID 36:

```
s9s cluster \
    --promote-slave \
    --cluster-id=36 \
    --nodes=192.168.40.31:3306 \
    --wait
```

Promote a PostgreSQL replica (192.168.40.41:5432) in cluster ID 37:

```
s9s cluster \
    --promote-slave \
    --cluster-id=37 \
    --nodes=192.168.40.41:5432 \
    --wait
```

---

## Stop/Start replica

Stops or starts the replication thread on the selected replica node without stopping the database service itself. This action is exclusive to replica nodes.

**Applies to:** MySQL/MariaDB (replication), Galera (if mix with a replica node(s))

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the replica node.
3. Click **Actions (…) → Replication → Stop replica** (or **Start replica** if the replication thread is currently stopped).
4. Confirm the action.

### ClusterControl CLI

Stop the replication thread on replica 192.168.40.31:3306 in cluster ID 36:

```
s9s node \
    --stop \
    --cluster-id=36 \
    --nodes=192.168.40.31:3306 \
    --wait
```

---

### Reset replica

Clears the replication state on the selected replica, causing it to forget its position in the primary's binary log and all connection parameters (primary host, port, user, and password). This is equivalent to running `RESET REPLICA ALL` on the replica. The replica service must be stopped before this action can be performed.

**Note**

Note Toggle **Force** to ignore errors encountered during the reset.

**Applies to:** MySQL/MariaDB (replication), Galera (if mix with a replica node(s))

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the replica node (ensure the replica thread is stopped first).
3. Click **Actions (…) → Replication → Reset replica**.
4. Optionally, toggle **Force** to bypass error checks.
5. Confirm the action.

---

### Rebuild replica

Rebuilds the selected replica node by re-synchronizing its data from a primary node. This is used when a replica has drifted too far behind, has become corrupted, or needs to be brought back into the replication topology cleanly.

Two methods are available:

- **From a primary node** – ClusterControl uses Percona XtraBackup (MySQL) or `pg_basebackup` (PostgreSQL) to stream a fresh copy of data directly from the primary. This uses the primary's I/O resources during the sync.
- **From a backup** – Toggle **Rebuild from a backup** and select any PITR-compatible backup from the dropdown. This reduces the impact on the primary during the rebuild.

**Applies to:** MySQL/MariaDB (replication), PostgreSQL/TimescaleDB, Microsoft SQL Server

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the replica node you want to rebuild.
3. Click **Actions (…) → Replication → Rebuild replica**.
4. Choose whether to rebuild from the primary directly or toggle **Rebuild from a backup** to use an existing PITR-compatible backup.
5. If rebuilding from a backup, select the desired backup from the dropdown list.
6. Confirm the action to proceed.

---

#### Change replication primary

Reconfigures the replication topology so that all replicas point to a different primary node. This action can be initiated from either a replica or a primary node.

**Applies to:** MySQL/MariaDB (replication), Galera (if mix with a replica node(s)), PostgreSQL/TimescaleDB

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate any replica or primary node in the cluster.
3. Click **Actions (…) → Replication → Change replication primary**.
4. Select the new primary node from the presented list.
5. Confirm the action. ClusterControl will reconfigure all replicas to replicate from the new primary.

#### ClusterControl CLI

Change the replication primary for cluster ID 36 so that replicas now follow 192.168.40.30:3306:

```
s9s cluster \
    --change-config \
    --cluster-id=36 \
    --nodes=192.168.40.30:3306 \
    --wait
```

---

### 2.46.4 Read-only and binary logging

#### Disable/Enable read-only

Sets or unsets the read-only mode on a database node. When enabled, ClusterControl sets `read_only=ON` and `super_read_only=ON` (MySQL 8.0 and later) on the target node.

**Applies to:** MySQL/MariaDB, Galera Cluster, PostgreSQL/TimescaleDB

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the target node.
3. Click **Actions (…) → Disable readonly** (to enable write access) or **Actions (…) → Enable readonly** (to set the node as read-only).
4. Confirm the action.

#### ClusterControl CLI

Enable `read_only` on a MySQL node (192.168.40.31:3306) in cluster ID 36:

```
s9s node \
    --change-config \
    --cluster-id=36 \
    --nodes=192.168.40.31:3306 \
    --opt-group=mysqld \
    --opt-name=read_only \
    --opt-value=ON
```

---

#### Disable/Enable binary logging

Updates the node's configuration to enable or disable MySQL binary logging. Enabling binary logging allows a replica to be added to the node and makes point-in-time recovery (PITR) possible. Disabling binary logging stops any attached replicas and prevents new replicas from being added. A server restart is required for either change to take effect.

> **Note**
>
> Note Toggle **Restart the node** to have ClusterControl immediately restart the node after the configuration update, applying the change without a separate step.

**Applies to:** Galera Cluster

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the Galera node.
3. Click **Actions (…) → Disable binary logging** or **Actions (…) → Enable binary logging**.
4. Optionally, toggle **Restart the node** to apply the change immediately.
5. Confirm the action.

---

### 2.46.5 Galera-specific actions

#### Resync node

Forces a full resync of the selected Galera node. ClusterControl removes all files in the node's data directory (`datadir`) and triggers a State Snapshot Transfer (SST) from a donor node. If a PITR-compatible full backup exists, ClusterControl will restore from it first to shorten the SST window and increase the probability of an Incremental State Transfer (IST), which is a non-blocking operation.

This action is typically used when a Galera node repeatedly fails to recover or when a filesystem issue is suspected.

> **Warning**
>
> Warning!

Wait for this operation to complete before starting another Galera node with **Initial Start**, as concurrent SST operations can disrupt the cluster.

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the Galera node that needs a resync.
3. Click **Actions (…) → Resync node**.
4. Confirm the action. ClusterControl will wipe the data directory and initiate an SST or IST from a suitable donor.

---

## Bootstrap Cluster

Bootstraps the Galera cluster from the selected node. ClusterControl stops all currently running Galera nodes before bootstrapping the cluster from the node you choose. Use this action when the entire cluster is down and needs to be started from a known-good node.

**Warning**

Warning!

All running Galera nodes will be stopped before the bootstrap begins. Ensure that the selected node has the most up-to-date data (highest `seqno`) to avoid data loss.

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the Galera node you want to bootstrap from (ideally the node with the highest sequence number).
3. Click **Actions (…) → Bootstrap Cluster**.
4. Confirm the action. ClusterControl will stop all running nodes and bootstrap the cluster from the selected node.

### ClusterControl CLI

Bootstrap the Galera cluster (cluster ID 36) from node 192.168.40.30:3306:

```
s9s cluster \
    --bootstrap \
    --cluster-id=36 \
    --nodes=192.168.40.30:3306 \
    --wait
```

---

### 2.46.6 MongoDB-specific actions

#### Step down node

Steps down the selected MongoDB primary, causing it to transition to a secondary and become ineligible for election as primary for a configurable number of seconds. The remaining voting members of the replica set will elect a new primary, excluding the stepped-down node for the specified duration.

**Applies to:** MongoDB (replica set primary)

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the MongoDB primary node.
3. Click **Actions (…) → Step down node**.
4. Specify the number of seconds during which the node is ineligible for re-election.
5. Confirm the action.

---

#### Freeze node

Prevents the selected MongoDB replica set member from seeking election for a specified number of seconds. To unfreeze the node before the duration elapses, issue the command again with a value of `0` seconds.

**Applies to:** MongoDB (replica set members)

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the MongoDB node.
3. Click **Actions (…) → Freeze node**.
4. Enter the number of seconds to freeze the node.
5. Confirm the action.

---

### 2.46.7 PostgreSQL-specific actions

#### Configure WAL

The Write-Ahead Log (WAL) is PostgreSQL's REDO log. WAL records all changes made to the database and is essential for replication, online backup, and point-in-time recovery (PITR). Enabling WAL archiving prevents the primary from recycling WAL segments before replicas have applied them, which would otherwise force a full replica rebuild.

!!! danger "Warning"

```
Changing the WAL archive mode or directory causes the loss of the␣
↪previously collected WAL archive, eliminating the ability to perform␣
↪point-in-time recovery (PITR) for any prior time frames.
```

**WAL configuration options:**

| Option | Description |
|---|---|
| **Archive Mode** | **Off** – WAL logs are not archived; PITR is not possible. **On** – WAL logs are archived only when the node is in primary mode. **Always** – WAL logs are archived regardless of the node's role (primary or replica). |
| **Compressed WAL Archive** | Enables compression for archived WAL logs. Enabled by default. |
| **Custom WAL Archive Directory** | Path to the WAL archive directory. Defaults to `/var/lib/pgsql/13/wal_archive` if left empty. |
| **Nodes** | WAL archiving can be configured on all database nodes simultaneously or on an individual node. |

**Applies to:** PostgreSQL/TimescaleDB

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the PostgreSQL node.
3. Click **Actions (…) → Configure WAL**. This opens the WAL configuration panel.
4. Set the **Archive Mode**, toggle **Compressed WAL Archive** as desired, and specify a **Custom WAL Archive Directory** if needed.
5. Select whether the change applies to **All nodes** or the individual node.
6. Confirm the action to apply the WAL configuration.

---

### Synchronous replication

Enables or disables synchronous streaming replication between the primary and the selected replica node. When enabled, the primary waits for the replica to confirm receipt of WAL data before acknowledging a transaction commit. Synchronous replication can be enabled per individual replica node but carries a notable performance overhead.

**Applies to:** PostgreSQL/TimescaleDB (replica nodes only)

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the PostgreSQL replica node.
3. Click **Actions (…) → Synchronous Replication**.
4. Toggle synchronous replication on or off for the selected replica.
5. Confirm the action.

---

### 2.46.8 ProxySQL-specific actions

#### Synchronize Instances

Synchronizes the ProxySQL configuration across multiple ProxySQL instances to ensure they remain identical. This operation overwrites the target's existing configuration.

Two synchronization modes are available:

- **Synchronize existing instances** – Exports the configuration from a source ProxySQL instance and imports it to one or more target instances.
- **Import Configuration file** – Restores a previously exported ProxySQL configuration backup to the selected instance.

**Applies to:** ProxySQL

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the ProxySQL node.
3. Click **Actions (…) → Synchronize Instances**.
4. Choose between **Synchronize existing instances** or **Import Configuration file**.
5. Select the source instance or the configuration backup file as appropriate.
6. Confirm the action.

---

#### Export configuration

Exports the full ProxySQL configuration from the selected instance to a backup file. The exported backup can be imported to any ProxySQL instance using the **Synchronize Instances → Import Configuration file** option.

The following configuration data is included in the export:

- Query Rules
- Host Groups and Servers
- Users and corresponding MySQL users
- Global Variables
- Scheduler
- `proxysql.cnf`

**Applies to:** ProxySQL

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the ProxySQL node.
3. Click **Actions (…) → Export configuration**.
4. Confirm the action. ClusterControl will generate and download the configuration backup file.

### 2.46.9 ClickHouse-specific actions

ClickHouse nodes are managed through the same generic actions described in *Node Actions* above (Restart node, Reboot host, Stop/Start node, Remove node, SSH Console). A few things are specific to how ClickHouse itself works:

- **Role column:** a ClickHouse data node always shows **Type**: `ClickHouse` and **Role**: `Replica`, even for a single-node (standalone) deployment; this reflects ClickHouse's own terminology, where every data node is a replica within a shard, rather than a ClusterControl-specific primary/replica distinction. A Keeper node instead shows **Role**: `Follower`. Keeper nodes run a separate `clickhouse-keeper` package/service, distinct from the `clickhouse-server` package that data nodes (and a standalone node's embedded Keeper) run.
- **Embedded ClickHouse Keeper:** on a standalone, single-node cluster, ClusterControl deploys ClickHouse with an embedded ClickHouse Keeper (the coordination service that replaces ZooKeeper), running inside the same `clickhouse-server` process. Adding a dedicated Keeper node (see *Add new → Keeper node*) installs the standalone `clickhouse-keeper` package instead.
- **SSL by default:** Inter-node communication is SSL-encrypted with per-node certificates by default.

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → choose the ClickHouse cluster → Nodes***.
2. The node list shows **Type**: `ClickHouse` and **Role**: `Replica` for each data (server) node, and **Type**: `Prometheus`, **Role**: `None` for the monitoring node. A Keeper node shows **Role**: `Follower`.

#### ClusterControl CLI

List ClickHouse nodes for cluster ID 4 (the `STAT` column's `C` flag identifies the controller host, not a database node; a Keeper node shows a `K` flag):

```
s9s node --list --long --cluster-id=4
```

#### ClusterControl RPC API

Node details are returned as part of the cluster info, via the `getClusterInfo` operation with `with_hosts` set to `true`:

```
curl -k -XPOST \
-d '{
   "operation": "getClusterInfo",
   "cluster_id": 4,
   "with_hosts": true,
   "authenticate": {
     "username": "myrpcadmin",
     "password": "mySuperStr0ngP@ssw0rd999"
   }
}' \
https://localhost:9501/v2/clusters
```

Both node kinds share `class_name: "CmonClickHouseHost"`, but differ in `nodetype` and `role`:

| | Data (server) node | Keeper node |
|---|---|---|
| `nodetype` | `clickhouse` | `clickhouse_keeper` |
| `role` | `replica` | `follower` |

A data node also has a nested `clickhouse` object with `keeper_server_id` and `replica_name`.

**Info**

Adding a shard, replica, or Keeper node is a cluster-wide action (**Actions… → Add new → Data node / Keeper node**), performed from Cluster Management rather than this page. See *User Guide → Management → Cluster Management → ClickHouse*.

---

### 2.46.10 Schedule maintenance for a node

Schedules a maintenance window for an individual node, suppressing alarms and notifications for the node during the specified period. While in maintenance mode, ClusterControl will not generate alerts for node events (e.g., threshold breaches, health check failures).

For full details on configuring, checking, and deactivating maintenance mode, see *Maintenance*.

**Warning**

Attention

If automatic node recovery is enabled, ClusterControl will still attempt to recover the node regardless of maintenance mode status. Disable node recovery before starting maintenance if you do not want ClusterControl to interfere.

**Applies to:** MySQL/MariaDB, Galera Cluster, PostgreSQL/TimescaleDB, MongoDB, Redis, Microsoft SQL Server, Elasticsearch, Valkey, ClickHouse, HAProxy, ProxySQL, Prometheus

#### ClusterControl GUI

**Schedule maintenance:**

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the target node.
3. Click **Actions (…) → Schedule maintenance**.
4. Specify the maintenance reason, define the duration, or set a custom start and end date/time.
5. Confirm to activate the maintenance window.

**Manage or stop active maintenance:**

1. Log in to your ***ClusterControl GUI → choose a database cluster → Nodes***.
2. Locate the node currently in maintenance (a blue wrench icon is shown next to its status).

3. Click **Actions (…) → Maintenance → Manage scheduled maintenance** to review the schedule details or delete it.
4. To stop an active maintenance window immediately, click **Actions (…) → Maintenance → Stop current maintenance** and confirm.

#### ClusterControl CLI

Schedule a one-day maintenance window for PostgreSQL node 192.168.40.40:5432 (ClusterControl expects UTC time):

```
s9s maintenance \
    --create \
    --nodes=192.168.40.40:5432 \
    --begin=2024-05-15T05:44:55.000Z \
    --end=2024-05-16T05:44:55.000Z \
    --reason='Upgrading RAM' \
    --batch
```

Schedule maintenance for a MySQL node 192.168.40.30:3306 starting tomorrow for one hour:

```
s9s maintenance \
    --create \
    --nodes=192.168.40.30:3306 \
    --begin="$(date --date='now + 1 day' --utc '+%Y-%m-%d %H:%M:%S')" \
    --end="$(date --date='now + 1 day + 1 hour' --utc '+%Y-%m-%d %H:%M:%S')
↪" \
    --reason="OS patching"
```

List all nodes currently under a maintenance period:

```
s9s maintenance --list --long
```

Delete (stop) a maintenance period by UUID:

```
s9s maintenance --delete --uuid=<maintenance_uuid>
```

## 2.47 Configuration Management

This feature enables administrators to define, apply, and monitor configurations for entire clusters or individual database nodes, ensuring consistency and compliance with best practices. With configuration management, users can automate updates, track configuration changes, and revert to previous versions if needed, simplifying maintenance and reducing the risk of misconfigurations that could impact performance or security.

### 2.47.1 Access the configuration management section

#### ClusterControl GUI

Log in to your ***ClusterControl GUI → choose a database cluster → Manage → Configurations***. This will bring up a panel where you can view and edit configurations.

### ClusterControl CLI

For CLI, the context of the configuration management is bounded at the node level. Hence, the `--nodes` flag is mandatory with a proper `<host>:<port>` value.

- List all database configuration options for a MySQL database node, 192.168.99.3:

```
s9s node --list-config --cluster-id=36 --nodes=192.168.99.3:3306
```

- List all database configuration options for a PostgreSQL database node, 192.168.99.6:

```
s9s node --list-config --cluster-id=37 --nodes=192.168.99.6:5432
```

### ClusterControl RPC API

For the RPC API, you can get the configuration in the node level and cluster level.

- List all database configuration options for a MySQL database node, 192.168.99.3:

```
curl -k -X POST \
  -d '{
    "cluster_id": 36,
    "hostname": "192.168.99.3",
    "port": 3306,
    "authenticate": {
      "username": "myrpcadmin",
      "password": "mySuperStr0ngP@ssw0rd999"
    },
    "operation": "getConfig"
  }' \
https://localhost:9501/v2/config
```

- List all database configuration options for a PostgreSQL database node, 192.168.99.6:

```
curl -k -X POST \
  -d '{
    "cluster_id": 36,
    "hostname": "192.168.99.6",
    "port": 5432,
    "authenticate": {
      "username": "myrpcadmin",
      "password": "mySuperStr0ngP@ssw0rd999"
    },
    "operation": "getConfig"
  }' \
https://localhost:9501/v2/config
```

- List all database configuration options for a PostgreSQL database node in all nodes:

```
curl -k -X POST \
 -d '{
   "cluster_id": 21,
   "authenticate": {
```

(continues on next page)

(continued from previous page)

```
    "username": "myrpcadmin",
    "password": "mySuperStr0ngP@ssw0rd999"
  },
  "operation": "getConfig"
 }' \
https://localhost:9501/v2/config
```

### 2.47.2 Refresh configuration files

#### ClusterControl GUI

Click on **Refresh config** to get the latest configuration files from all nodes managed by ClusterControl. This will trigger a job to pull the latest configuration files from all nodes. The configuration files output listed will be refreshed after the job is successfully executed.

#### ClusterControl CLI

Suppose the cluster ID is 36, and wait for the refresh configuration job to finish:

```
s9s cluster --import-config --cluster-id=36 --wait
```

### 2.47.3 View and edit configuration files

#### ClusterControl GUI

1. In the Configuration Management panel, you will see a list of configuration files for the selected database technology (e.g., `my.cnf` for MySQL, `postgresql.conf` for PostgreSQL, `proxysql.cnf` for ProxySQL).
2. Select a configuration file to view its current parameters. Selecting a configuration will appear as a new tab in the editor area.
3. ClusterControl allows you to edit configurations directly within this interface, similar to a text editor. Make your changes as needed, adjusting parameters to meet your performance, security, or operational needs.
4. Click **Save** to apply the change. ClusterControl will push the changes into the configuration file on the corresponding database node. If you would like to change a configuration across all nodes in the cluster, use the **Edit params** button as explained in the next section.

   !!! Info

   ```
   Editing the configuration file with the text editor will only push the␣
   ↪configurations to the database configuration file, and not the␣
   ↪database runtime. Restart the database service to apply those␣
   ↪changes into runtime, or use the **Edit params** button.
   ```

#### ClusterControl CLI

For CLI, the context of the configuration management is bounded at the node level. Hence, the `--nodes` flag is mandatory with a proper `<host>:<port>` value.

- Change a MySQL configuration option for a MySQL database node 192.168.99.3 called `max_connections` which is located under the `[mysqld]` section:

```
s9s node \
    --change-config \
    --cluster-id=36 \
    --nodes='192.168.99.3:3306' \
    --opt-group=mysqld \
    --opt-name=max_connections \
    --opt-value=300
```

- Change a PostgreSQL configuration option for a PostgreSQL database node, 192.168.99.6 called `log_line_prefix`:

```
s9s node \
    --change-config \
    --cluster-id=37 \
    --nodes='192.168.99.6:5432' \
    --opt-name=log_line_prefix \
    --opt-value='%m '
```

### ClusterControl RPC API

For RPC API, the context of the configuration management is bounded at the node level.

- Change a MySQL configuration option for a MySQL database node 192.168.99.3 called `max_connections` which is located under the `[mysqld]` section:

```
curl -k -X POST \
  -d '{
    "cluster_id": 36,
    "hostname": "192.168.99.3",
    "port": 3306,
    "authenticate": {
      "username": "myrpcadmin",
      "password": "mySuperStr0ngP@ssw0rd999"
    },
    "configuration": [
      {
        "name": "max_connections",
        "group": "MYSQLD",
        "value": "190"
      }
    ],
    "operation": "setConfig"
  }' \
  https://localhost:9501/v2/config
```

- Change a PostgreSQL configuration option for a PostgreSQL database node, 192.168.99.6 called `log_line_prefix`:

```
curl -k -X POST \
  -d '{
    "cluster_id": 21,
```

(continues on next page)

(continued from previous page)

```
    "hostname": "192.168.99.6",
    "port": 5432,
    "authenticate": {
      "username": "myrpcadmin",
      "password": "mySuperStr0ngP@ssw0rd999"
    },
    "configuration": [
      {
        "name": "log_line_prefix",
        "value": "%m"
      }
    ],
    "operation": "setConfig"
  }' \
  https://localhost:9501/v2/config
```

### 2.47.4 Compare new configuration with existing configuration

#### ClusterControl GUI

Use **Show diff** button to compare the new configuration with the existing configuration before hitting the **Save** button. ClusterControl will open a new panel to compare these two configuration versions side-by-side before and after changes.

The left panel displays the existing configuration (changes are highlighted in red) while the right-panel displays the currently modified (yet to apply) configuration (changes are highlighted in green).

Click **Revert Changes** to rollback and discard the change, or click **Save** to commit and push the new configuration to the database node. ClusterControl will perform a file-system replace to the existing configuration file on the database node with the new version.

**Note**

Note ClusterControl does not store configuration changes history so there is no file versioning after saving. Only one version exists at one time on the managed node.

### 2.47.5 Apply configuration changes across nodes

#### ClusterControl GUI

Use **Change Parameters** if you want to change a specific configuration option across database nodes in the cluster.

1. Click on **Change Parameters** button and ClusterControl will load the options of Database Instances, Groups (MySQL only), and Parameters in a table format.
2. Choose the database instance you want to apply the configuration changes. The option can be specific node or all the nodes.
3. Click dropdown menu on the Group, and select the category to which the parameters belong.
4. Look up the configuration and search in the filtering field for the parameter and you can edit the new configuration value by filling up the **New Value**.

5. After making changes, press **Proceed** and the configuration will be pushed to all database nodes' configuration file and runtime (only if applicable).

### ClusterControl CLI

For CLI, the command must be repeated multiple times for every database node.

- Change a MySQL configuration option for multiple MySQL database nodes in cluster ID 36 (192.168.99.3, 192.168.99.4, 192.168.99.5) called `max_connections` which is located under the `[mysqld]` section:

```
s9s node \
    --change-config \
    --cluster-id=36 \
    --nodes='192.168.99.3:3306' \
    --opt-group=mysqld \
    --opt-name=max_connections \
    --opt-value=300

s9s node \
    --change-config \
    --cluster-id=36 \
    --nodes='192.168.99.4:3306' \
    --opt-group=mysqld \
    --opt-name=max_connections \
    --opt-value=300

s9s node \
    --change-config \
    --cluster-id=36 \
    --nodes='192.168.99.5:3306' \
    --opt-group=mysqld \
    --opt-name=max_connections \
    --opt-value=300
```

- Change a PostgreSQL configuration option for multiple PostgreSQL database nodes in cluster ID 37, (192.168.99.6, 192.168.99.7) called `log_line_prefix`:

```
s9s node \
    --change-config \
    --cluster-id=37 \
    --nodes='192.168.99.6:5432' \
    --opt-name=log_line_prefix \
    --opt-value='%m '

s9s node \
    --change-config \
    --cluster-id=37 \
    --nodes='192.168.99.7:5432' \
    --opt-name=log_line_prefix \
    --opt-value='%m '
```

### ClusterControl RPC API

Similar with the CLI, the RPC API calls must be repeated multiple times for every database node.

- Change a MySQL configuration option for multiple MySQL database nodes in cluster ID 36 (192.168.99.3, 192.168.99.4, 192.168.99.5) called `max_connections` which is located under the `[mysqld]` section:

```
curl -k -X POST \
  -d '{
    "cluster_id": 36,
    "hostname": "192.168.99.3",
    "port": 3306,
    "authenticate": {
      "username": "myrpcadmin",
      "password": "mySuperStr0ngP@ssw0rd999"
    },
    "configuration": [
      {
        "name": "max_connections",
        "group": "MYSQLD",
        "value": "190"
      }
    ],
    "operation": "setConfig"
  }' \
  https://localhost:9501/v2/config


curl -k -X POST \
  -d '{
    "cluster_id": 36,
    "hostname": "192.168.99.4",
    "port": 3306,
    "authenticate": {
      "username": "myrpcadmin",
      "password": "mySuperStr0ngP@ssw0rd999"
    },
    "configuration": [
      {
        "name": "max_connections",
        "group": "MYSQLD",
        "value": "190"
      }
    ],
    "operation": "setConfig"
  }' \
  https://localhost:9501/v2/config


  curl -k -X POST \
    -d '{
```

(continues on next page)

(continued from previous page)

```
    "cluster_id": 36,
    "hostname": "192.168.99.5",
    "port": 3306,
    "authenticate": {
      "username": "myrpcadmin",
      "password": "mySuperStr0ngP@ssw0rd999"
    },
    "configuration": [
      {
        "name": "max_connections",
        "group": "MYSQLD",
        "value": "190"
      }
    ],
    "operation": "setConfig"
  }' \
  https://localhost:9501/v2/config
```

### 2.47.6 Monitor configuration consistency

#### ClusterControl GUI

ClusterControl allows you to monitor and verify that configurations are consistent across your cluster. Go to ***ClusterControl GUI → choose a database cluster → Performance → DB Variables*** to identify configuration drift (differences in configurations between nodes).

Entries highlighted in red indicate the configuration options are not identical across nodes. Note that some configuration options may be distinct across nodes and does not necessary mean a poor configuration.

#### ClusterControl CLI

To compare output between two commands, we can use `diff -y` with side-by-side output.

- Compare the current database configurations (from the configuration file) for two MySQL database nodes, 192.168.99.3 and 192.168.99.4:

```
diff -y <(s9s node --list-config --nodes=192.168.99.3:3306) <(s9s node --
↪list-config --nodes=192.168.99.4:3306)
```

- Compare the current database configurations (from the configuration file) for two PostgreSQL database nodes, 192.168.99.7 and 192.168.99.8:

```
diff -y <(s9s node --list-config --nodes=192.168.99.7:5432) <(s9s node --
↪list-config --nodes=192.168.99.8:5432)
```

### 2.47.7 Configuration template

All services configured by ClusterControl use a base configuration template available under `/usr/share/cmon/templates` on the ClusterControl node. You can directly modify the file to suit your deployment policy however, this directory will be replaced after a package upgrade.

To make sure your custom configuration template files persist across upgrades, store your template files under the `/etc/cmon/templates` directory (ClusterControl 1.6.2 and later). When ClusterControl loads up the template files for deployment, files under `/etc/cmon/templates` will always have higher priority over the files under `/usr/share/cmon/templates`. If two files having identical names exist on both directories, the one located under `/etc/cmon/templates` will be used.

Here is an example of how one would create a custom configuration template file:

```
# on the ClusterControl node, as privileged user
mkdir -p /etc/cmon/templates
cp /usr/share/cmon/templates/my.cnf.repl80 /
 ↪etc/cmon/templates/my.cnf.repl80-custom
vi /etc/cmon/templates/my.cnf.repl80-custom     # make your changes and save
```

Go to the ***ClusterControl GUI → Deploy a cluster → Create a database cluster*** and start deploying a database cluster. Choose the created configuration file above from the **Configuration template** dropdown.

### Dynamic variables

Inside the provided template files, many configuration variables are configurable dynamically (depending on the detected host specifications and user's input in the deployment wizard) by ClusterControl during deployment. These variables are represented with a capital letter enclosed by `@`, for example, `@DATADIR@`. The following shows the list of variables supported by the ClusterControl, grouped by the cluster type:

### MySQL/MariaDB/Percona Server

| Variable | Description |
|---|---|
| @BASEDIR@ | The default is /usr. Value specified during cluster deployment takes precedence. |
| @DATADIR@ | The default is /var/lib/mysql. Value specified during cluster deployment takes precedence. |
| @MYSQL_PORT@ | The default is 3306. Value specified during cluster deployment takes precedence. |
| @BUFFER_POOL_SIZE@ | Automatically configured based on the host's RAM. |
| @LOG_FILE_SIZE@ | Automatically configured based on the host's RAM. |
| @LOG_BUFFER_SIZE@ | Automatically configured based on the host's RAM. |
| @BUFFER_POOL_INSTAN | Automatically configured based on the host's CPU. |
| @SERVER_ID@ | Automatically generated based on member's server-id. |
| @SKIP_NAME_RESOLVE@ | Automatically configured based on MySQL variables. |
| @MAX_CONNECTIONS@ | Automatically configured based on the host's RAM. |
| @ENABLE_PERF_SCHEMA@ | The default is disabled. Value specified during cluster deployment takes precedence. |
| @WSREP_PROVIDER@ | Automatically configured based on the Galera vendor. |
| @HOST@ | Automatically configured based on hostname/IP address. |
| @GCACHE_SIZE@ | Automatically configured based on disk space. |
| @SEGMENTID@ | The default is 0. Value specified during cluster deployment takes precedence. |
| @WSREP_CLUSTER_ADDRESS | Automatically configured based on members in the cluster. |
| @WSREP_SST_METHOD@ | Automatically configured based on the Galera vendor. |
| @BACKUP_USER@ | Default is backupuser. |
| @BACKUP_PASSWORD@ | Automatically generated and configured for backupuser. |
| @GARBD_OPTIONS@ | Automatically configured based on garbd options. |
| @READ_ONLY@ | Automatically configured based on replication role. |
| @SEMISYNC@ | The default is disabled. Value specified during cluster deployment takes precedence. |
| @NDB_CONNECTION_POO | Automatically configured based on the host's CPU. |
| @NDB_CONNECTSTRING@ | Automatically configured based on members in the MySQL cluster. |
| @LOCAL_ADDRESS@ | Automatically configured based on the host's address. |
| @GROUP_NAME@ | Default is grouprepl. Value specified during cluster deployment takes precedence. |
| @PEERS@ | Automatically configured based on members in the Group Replication cluster. |

### MongoDB

| Variable | Description |
|---|---|
| @DATADIR@ | The default is /var/lib/mongodb. Value specified during cluster deployment takes precedence. |
| @MONGODB_PORT@ | The default is 27017, 27018, 27019 (depending on the cluster type). Value specified during cluster deployment takes precedence. |
| @LOGDIR@ | Automatically configured based on vendor. |
| @HOST@ | Automatically configured based on hostname/IP address. |
| @SMALLFILES@ | Automatically configured based on disk space. |
| @PIDFILEPATH@ | Automatically configured based on MongoDB data directory. |
| @REPLICASET_NAME@ | The default is my_mongodb_N. Value specified during cluster deployment takes precedence. |

### Redis Cluster/Redis Sentinel/Valkey Cluster

| Variable | Description |
|---|---|
| @BIND_ADDRESS@ | Automatically configured based on the host's address. |
| @PORT@ | The default is 6379. Value specified during cluster deployment takes precedence. |
| @DATADIR@ | The default is /var/lib/redis. Value specified during cluster deployment takes precedence. |
| @REPLICATION_PASSWORD@ | Automatically configured based on the value specified during the deployment. |
| @REPLICA_OF@ | Automatically configured based on members in the topology. |
| @CLUSTER_CONFIG_FILE@ | Automatically configured based on cluster type. |
| @CLUSTER_NODE_TIMEOUT@ | The default is 15000 in milliseconds. Value specified during cluster deployment takes precedence. |
| @CLUSTER_BUS_PORT@ | The default is 16379. Value specified during cluster deployment takes precedence. |
| @CLUSTER_REPLICA_VALIDITY_FAC | The default is 10. Value specified during cluster deployment takes precedence. |
| @ACL_DB_ADMIN_USER@ | Automatically configured based on members in the topology. |
| @SENTINEL_PASSWORD@ | Automatically configured based on the value specified during the deployment. |
| @PASSWORD@ | Automatically configured based on the value specified during the deployment. |
| @MAXMEMORY_BYTES@ | Automatically configured based on the host's RAM. |
| @MAXMEMORY_POLICY@ | The default is allkeys-lru. |
| @CLUSTER_ENABLED@ | Automatically configured based on cluster type. |

### Elasticsearch

| Variable | Description |
|---|---|
| @CLUSTER_NAME@ | Automatically configured based on the value specified during the deployment. |
| @NODE_NAME@ | Automatically configured based on hostname/IP address. |
| @NODE_ROLES@ | Automatically configured based on node's role. |
| @DATADIR@ | The default is /var/lib/elasticsearch. Value specified during cluster deployment takes precedence. |
| @SNAPSHOTS_LOCATION@ | Automatically configured based on the value specified during the deployment. |
| @BIND_ADDRESS@ | Automatically configured based on hostname/IP address. |
| @PUBLISHED_ADDRESS@ | Automatically configured based on hostname/IP address. |
| @HTTP_PORT@ | The default is 9200. Value specified during cluster deployment takes precedence. |
| @DISCOVERY_SEED_HOSTS@ | Automatically configured based on hostname/IP address of the members in the topology. |
| @CLUSTER_INITIAL_MASTER_NOI | Automatically configured based on members in the topology. |
| @SECURITY_ENABLED@ | Automatically configured based on the value specified during the deployment. |

## 2.48 Upgrades and Patches

Performs database and load-balancer-related package upgrades.

ClusterControl upgrades database and load balancer packages by utilizing the vendor's package repository. The upgrade process depends on the availability of updated packages in this repository. If the vendor repository is outdated or inaccessible (due to network restrictions or proxy configurations), ClusterControl cannot perform the upgrade as new packages will not be available in the server's repository list.

!!! danger "Warning"

```
A database upgrade is a risky operation and ClusterControl does not support␣
↪an upgrade rollback operation. Please perform this activity on a␣
↪test/staging cluster first before proceeding with the actual upgrade.␣
↪Upgrades should only be performed when there is as little traffic as␣
↪possible on the cluster. Performing a full database backup (or taking a␣
↪full VM snapshot) before an upgrade will increase the chance of recovery␣
↪in case a rollback is needed. Taking a backup (before the upgrade) or␣
↪restoring a backup (to rollback) is not part of this particular upgrade␣
↪job.
```

### 2.48.1 Upgrade procedures

Database cluster upgrade procedures vary based on the cluster's topology, as detailed below.

#### Primary-replica replication

ClusterControl upgrades primary-replica setups by updating replica/secondary nodes first (in random order), followed by the primary node. Expect a service interruption during the primary node upgrade until it restarts.

**Warning**

Attention

ClusterControl does not perform automatic switchover before upgrading the primary node, which is always upgraded last. Always test upgrade procedures in a non-production environment before applying them to production databases.

#### Multi-primary

In a multi-primary setup, ClusterControl upgrades database nodes randomly, one node at a time. If an upgrade fails, the entire process stops, necessitating manual intervention for node recovery or reinstallation. Consequently, a suitable maintenance window with minimal cluster traffic is crucial for upgrades.

**Tip**

If database connections are load-balanced, performing the entire upgrade without database service downtime may be possible. Always test upgrade procedures in a test environment before applying them to a production database.

### 2.48.2 Minor upgrade

Performs minor version software upgrades for database and load balancer software. Minor upgrade version formatting is notably different depending on the database type and vendor (from `x` to `y`):

- MySQL/Percona Server for MySQL – 8.0.x to 8.0.y
- MariaDB – 11.4.x to 11.4.y
- MongoDB – 7.0.x to 7.0.y
- PostgreSQL/TimescaleDB/EntepriseDB – 16.x to 16.y
- Redis Cluster/Sentinel – 7.x to 7.y
- ProxySQL – 2.6.x to 2.6.y
- PgBouncer – 1.22.x to 1.22.y
- HAProxy – 2.1.x to 2.1.y
- MariaDB MaxScale – 22.08.x to 22.08.y

---

#### Checking for available upgrades

To determine if a newer version is available for your database or load balancer, follow the instructions below for either the ClusterControl GUI or CLI.

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Clusters***
2. Select the desired cluster from the list.
3. Go to ***Actions (…) → Upgrades → Minor upgrades***.
4. ClusterControl will display a list of database and load balancer nodes within the cluster that can be upgraded.
5. Click **Check for upgrades**. This initiates a background job to query the package repositories of the database and load balancer nodes for the latest available versions.
6. Once the job is complete, verify if a new version is available.

### ClusterControl CLI

- To initiate a check for newer package versions on cluster ID 17 and wait for the process to finish, execute the following command:

```
s9s cluster --check-pkg-upgrades --cluster-id=17 --wait
```

---

## Performing minor version upgrade

Minor upgrades are performed online on the selected node(s), one node at a time. The node will be stopped, then the software will be updated, and then the node will be started again. If a node fails to upgrade, the upgrade process is aborted and manual intervention is required to recover or reinstall the node.

To perform a minor version upgrade, either for the database cluster or load balancer, or both, follow the instructions below for either the ClusterControl GUI or CLI.

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Clusters***
2. Select the desired cluster from the list.
3. Go to ***Actions (…) → Upgrades → Minor upgrades***.
4. ClusterControl will display a list of database and load balancer nodes within the cluster that can be upgraded.
5. Check all the boxes for the nodes that you want to upgrade. Clicking on the top checkbox will automatically check all the boxes underneath it.
6. Select **Upgrade**. ClusterControl will start an upgrade process even if a newer version is not available. If no upgrade is needed, the job will check package versions and report "Nothing to upgrade."
7. Monitor the job log messages closely.

### ClusterControl CLI

- Update database and load balancer packages to the newest minor version across all nodes in cluster ID 1:

```
s9s cluster \
    --upgrade-cluster \
    --cluster-id=1 \
    --wait
```

- Update database packages to the newest minor version for all MySQL database nodes in cluster ID 1:

```
s9s cluster \
    --upgrade-cluster \
    --cluster-id=1 \
    --nodes="192.168.0.81:3306;192.168.0.82:3306;192.168.0.83:3306" \
    --wait
```

### 2.48.3 Major upgrade

**Info**

ClusterControl only supports PostgreSQL-based clusters for major version upgrade, for example from PostgreSQL 16.x to PostgreSQL 17.x.

Performs major software upgrades for the database software. Only one major version upgrade is supported at a time, for example, from PostgreSQL 13.x to 14.x. At the moment, only PostgreSQL-based clusters are supported. If you want to upgrade from PostgreSQL 12.x to 14.x, you have to upgrade stage-by-stage to PostgreSQL 13.x first, followed by PostgreSQL 14.x.

For a primary-replica replication setup, ClusterControl will perform the upgrade starting from the replica/secondary nodes and will eventually perform the upgrade on the primary node of a cluster (secondary first, primary last). During the eventual primary node upgrade, expect a service disruption to the database cluster service until the primary node is online again after restarting. If a node fails to be upgraded, the upgrade job is aborted and manual intervention is required to recover or reinstall the node. Due to this fact, it is important to have a proper maintenance window, which should only be performed when there is as little traffic as possible to the cluster.

#### Performing major version upgrade

Major upgrades are performed online on all database nodes in the cluster, one node at a time. The node will be stopped, then the software will be updated, and then the node will be started again. If a node fails to upgrade, the upgrade process is aborted and manual intervention is required to recover or reinstall the node.

**Warning**

Attention

Depending on the upgrade method chosen, you might require the storage capacity to hold 2 data sets.

To perform a major version upgrade for the database cluster, follow the instructions below for either the ClusterControl GUI or CLI.

### ClusterControl GUI

1. Navigate to ***ClusterControl GUI → Clusters***
2. Select the desired cluster from the list. Only PostgreSQL-based clusters are supported.
3. Go to ***Actions (…) → Upgrades → Major upgrades***.
4. Under the **Configuration** tab, ClusterControl will display the database vendor, current major version and the upgrade major version. This is informational and not configurable. ClusterControl only allows one major version upgrade at a time.
5. Choose one of the upgrade method from the dropdown:
   - **copy** – This is the default method with `pg_upgrade`. The `--copy` option is used which will copy "files at a time" which is faster than doing regular backup and restore. This requires disk storage capacity to hold both the old and the new files. Note that data stored in user-defined tablespaces is not being copied to the new cluster during an upgrade. It remains in its original file system location but is duplicated into a subdirectory indicating the version number of the new cluster. To manually relocate files that are stored in a tablespace after upgrading, move the files to the new location and update the symbolic links to point to the files.
   - **link** – This method uses the `--link` option with `pg_ugprade`. Hard links will be used instead of copying (`--copy`) files which is faster and requires no extra disk storage. It is crucial to refrain from starting the old cluster after this step to avert any potential data corruption with the new cluster since they share the data.
   - **pgdumpall** – This is an alternative method to `pg_upgrade` and uses `pgdumpall` to copy files by performing a backup on the old cluster and then restoring it on the new cluster. This requires disk storage capacity to hold the backup for the old and new files.
6. Accept the disclaimer to proceed with the upgrade.
7. Under the **Advanced** tab, configure the following:
   - **Temporary master port** - The temporary PostgreSQL port of the older version. ClusterControl will reconfigure the PostgreSQL port of the older instances to this port for the duration of the upgrade, since the common port 5432, will be taken over by the newer PostgreSQL instance after the upgrading job completes.
   - **Temporary upgrade candidate port** - The temporary PostgreSQL port of the newer version while the upgrade job is running. ClusterControl distinguishes every node with a "hostname:port" pair. This provides a temporary entity for the newer database instance for the duration of the upgrade before it takes over the actual PostgreSQL port. This port will be released after the upgrade job is completed.
   - **Delete old node** - Toggle on to delete the older PostgreSQL instance data if the upgrade succeeds. Note that all previous data will be deleted and this action cannot be rolled back.
   - **Preserve configuration settings** - When enabled, important PostgreSQL configuration parameters will be preserved during major version upgrades instead of using default values for the new version.
8. Proceed to the **Preview** section and click **Submit**. ClusterControl will trigger a major upgrade job and the progress can be monitored under ***ClusterControl GUI → Activity Center → Jobs***.

### ClusterControl CLI

- Perform major upgrade from PostgreSQL 14 to PostgreSQL 15 for cluster ID 21:

```
s9s cluster \
    --upgrade-cluster \
    --upgrade-to-version=15 \
    --cluster-id=21 \
    --log
```

- Perform major upgrade from PostgreSQL 15 to 16 for cluster ID 1, preserve the configuration and use upgrade method `--link`:

```
s9s cluster \
    --upgrade-cluster \
    --upgrade-to-version=16 \
    --upgrade-method=link \
    --preserve-configuration \
    --cluster-id=1 \
    --log
```

---

## 2.49 Alerts and Notifications

ClusterControl provides multiple ways to view and send alerts when an alarm raised. The alarms can be viewed within ClusterControl GUI and CLI, and the corresponding alerts can be sent via email, messaging platform (Slack, Telegram), incident management systems (PagerDuty, OpsGenie, VictorOps, ilert) or web hooks.

Email notification is built-in into ClusterControl logic and requires a *mail server configuration* and *recipients configuration*. For other alerting endpoints like messaging platform, incident management and web hooks, ClusterControl relies on another module call `cmon-events`, provided by package `clustercontrol-notifications`. This module allows ClusterControl alarms to be forwarded to third-party services and easily configurable via *Notification Services*.

**See also**

To learn more about ClusterControl notification module, see *ClusterControl Notifications*.

### 2.49.1 Alarms

The alarms page provides administrators with an aggregated view of alarms raised for all clusters monitored by ClusterControl. Each alarm entry has a header, details on the issue, severity level, category, corresponding cluster name, corresponding host, and timestamp. Alarms indicate problems that could affect or degrade the cluster as a whole. Each alarms is categorized as:

- Cluster
- Cluster recovery
- Database health
- Database performance

- Host
- Node
- Network

#### View alarms

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Clusters → choose a database cluster → Alarms***. This will open a page where you can view alarms for the particular cluster.
2. Alternatively, you can view all alarms for all clusters by going to ***ClusterControl GUI → Activity Center → Alarms***.
3. Click on ***Actions → Details***, to get a detailed explanation of the problem, together with the recommended action (if available) to resolve the problem.
4. An alarm can be acknowledged by clicking on ***Actions → Mute***. When muted, no notification will be sent out for this type of alarm until it is unmuted. An alarm cannot be deleted or dismissed.
5. You can also see the most recent alarms summarized in the **Home** page under the **Recent alarms** section (bottom right).

**Tip**

Rollover on the ***Cluster*** entry will pop up a cluster information card summarizing the cluster status.

#### ClusterControl CLI

For CLI:

- List all alarms for all clusters:

```
s9s alarm --list
```

- List all alarms for cluster ID 1:

```
s9s alarm --cluster-id=1 --list
```

### 2.49.2 Email notifications

When an alarm is raised, ClusterControl can send an alert to any mail recipients. To receive an email from ClusterControl, one has to configure a remote SMTP mail server (or use local MTA) and enable email notifications for the particular cluster, as shown in the following sections.

#### Configure SMTP mail server

After installation, by default, ClusterControl does not send out email until a mail server is configured, or a local mail transfer agent (Postfix, Sendmail or Exim) is installed on activated on the ClusterControl server.

### ClusterControl GUI

1. To configure an SMTP mail server, go to ***ClusterControl GUI → Settings → Configure now***. It will open a wizard called "Configure mail server". Choose **SMTP Server**.
2. In the "Configure mail server: SMTP" window, configure the following:
   - **Server Address**: The SMTP mail server address that you are going to use to send the email.
   - **Port**: The SMTP port for the mail server. Usually, this value is 25 or 587, depending on your SMTP mail server configuration.
   - **User name**: The SMTP user name. Leave empty if no authentication required.
   - **Password**: The SMTP password for **User name**. Leave empty if no authentication required.
   - **Reply to/from**: Specify the sender of the email. This will appear in the 'From' field of the mail header.
   - **TLS/SSL required**: Check this box if ou want to use TLS/SSL for extra security. The mail server must support TLS/SSL.
3. Check the **Send test email** to test out the SMTP configuration. If you receive an email from ClusterControl, click **Save** and proceed to configure the mail recipients. See *Configure mail recipients*.

### Configure local MTA

To use local MTA to send out emails, make sure the local MTA is installed and verified using the test email button available at ***ClusterControl GUI → Settings → Configure now → Sendmail → Send test email***.

### Sendmail

On ClusterControl server, install the following packages:

### RHEL/CentOS/AlmaLinux/Rocky Linux

```
yum install sendmail mailx
```

### Debian/Ubuntu

```
apt-get install sendmail mailutils
```

Start the Sendmail service:

```
systemctl start sendmail
```

Verify if it works:

```
$ echo "test message" | mail -s "test subject" myemail@example.com
```

Replace `myemail@example.com` with your email address.

### Postfix

Many Linux distributions come with Sendmail as the default MTA. To replace Sendmail and use other MTA, e.g Postfix, you just need to uninstall Sendmail, install Postfix and start the service. Following example shows commands that need to be executed on the ClusterControl node as the root user for RockyLinux 9:

```
systemctl stop sendmail
yum remove sendmail -y
yum install postfix mailx cronie -y
systemctl enable postfix
systemctl start postfix
```

### Configure mail recipients

After a mail server is configured, or a MTA is installed and activated, one has to enable email notifications for the clusters.

### ClusterControl GUI

1. Go to ***ClusterControl GUI → Clusters → choose a database cluster → Settings → Email notifications***. It will open a wizard to help you configure the email recipients. This page configures email notifications for alarms generated for the corresponding database cluster.
2. Each recipient (chosen from a user group) must be enabled before you can configure the delivery settings. Choose one group under **User group** (only groups that has at least read permission to the cluster), select a user under the **Users in the selected group** dropdown and click **Enable**.
3. After enabled, configure the **Digest delivery times** which when to send a digested (summary) email every day to the selected recipient. The timezone will follow the user's timezone and can be configured under the **User profile** setting.
4. Configure **Digest delivery times** which is the maximum number of non-digest email notifications should be sent per day to the selected recipient. The default is 100. This is to protect spamming the recipients for events that are set to "Deliver". See *Events*.
5. Next, configure the frequency of the emails to be sent for each event's category. See *Events*.
6. You can add an external email address (a user that does not exist in ClusterControl) by clicking on **Add an external email**.

### Events

On every event severity, you may choose the frequency of raised alarms/events to be delivered to the recipient:

| Action | Description |
|---|---|
| **Ignore** | Ignore if an alarm is raised. |
| **Deliver** | Send notification immediately via email once an alarm is raised. |
| **Digest** | Send a summary of alarms raised every day at **Digest delivery time**. |

The following table shows the event category that you can forward to a specific recipient:

| Event | Description |
|---|---|
| **Network related logs** | Network-related messages, e.g. host unreachable, SSH issues. |
| **Cmon database logs** | Internal CMON database-related messages. |
| **Mail logs** | Mail system-related messages. |
| **Cluster logs** | Cluster-related messages, e.g. cluster failed. |
| **Cluster configuration logs** | Cluster configuration messages, e.g. software configuration messages. |
| **Recovery logs** | Recovery messages like cluster or node recovery failures. |
| **Node related logs** | Message related to nodes, e.g. node disconnected, missing GRANT, failed to start HAProxy, failed to start NDB cluster nodes. |
| **Host logs** | Host-related messages, e.g. CPU/disk/RAM/swap alarms. |
| **Health logs** | Database health-related messages, e.g. memory usage of MySQL servers, connections. |
| **Database performance** | Alarms for long-running transactions and deadlocks. |
| **Installation logs** | Software installation-related messages. |
| **Backup logs** | Messages about backups. |
| **Other logsp** | Other uncategorized messages. |

### 2.49.3 Forward alarms to third-party services

Alarms can be forwarded to third-party services like messaging platforms (Slack, Telegram), incident management systems (PagerDuty, ServiceNow, OpsGenie, VictorOps, ilert) and web hooks. See *Integration → Notification Services*.

## 2.50 Database Users and Privileges

ClusterControl offers a centralized feature for managing database users and their privileges, providing a comprehensive summary for your cluster. Any changes made through this feature are automatically synchronized across all database nodes in the cluster.

Supported database clusters:

- MySQL/MariaDB replication & standalone
- MySQL/MariaDB Galera Cluster
- MySQL Group Replication
- PostgreSQL/TimescaleDB
- Redis/Valkey

### 2.50.1 Best practices

When utilizing this feature, adhere to the following recommendations:

- Operational notes:
  - Changes made via ClusterControl are applied to all nodes in the cluster
  - Privilege changes take effect immediately

    - Ensure proper replication state before making user changes
- Security and administration recommendations:
    - Avoid granting global privileges unless necessary
    - Prefer database-specific privileges for application users
    - Regularly audit unused accounts
    - Use strong passwords and limit host access
    - Do not modify ClusterControl system users unless required

### 2.50.2 Access the database users management

#### ClusterControl GUI

Log in to your ***ClusterControl GUI → choose a database cluster → Manage → DB Users*** (for Redis-based clusters, it is ***Access Control***). You will see a panel where you can view and manage database users and privileges.

The panel view is different depending on the cluster type. See below:

- *MySQL/MariaDB-based clusters*
- *PostgreSQL-based clusters*
- *Redis-based clusters*

#### ClusterControl CLI

To manage database users using CLI, use the `account` or `accounts` command (both are aliases).

To view the manual or help pages, use the `--help` flag or, `man` command:

```
s9s account --help
s9s accounts --help
man s9s account
man s9s-account
```

### 2.50.3 MySQL/MariaDB-based clusters

ClusterControl provides a centralized interface to manage MySQL and MariaDB database users across your cluster. This feature allows administrators to view, create, modify, and audit database users and their privileges directly from the GUI without manual SQL commands.

**Info**

ClusterControl understands the MySQL/MariaDB topology and creates database users on all nodes where applicable in the cluster. This includes the supported load balancer nodes like ProxySQL.

### Common system users for MySQL

ClusterControl automatically manages several system users required for cluster operation:

| User | Purpose |
|---|---|
| cmon | Internal cluster management |
| cmonagent | ClusterControl query monitoring agent |
| backupuser | Database backup and restore |
| cmonexporter | Database metrics collection (Prometheus exporter) |
| proxysql-monitor | ProxySQL monitoring |
| rpl_user | Database replication |
| maxscale | MariaDB MaxScale monitoring |

**Warning**

Warning!

Modifying these users may impact cluster functionality.

### List MySQL/MariaDB database users

### ClusterControl GUI

The user list provides a tabular overview of all database users. In the main panel, you can filter the list by filing up the textbox with the magnifying glasss icon.

| Column | Description |
|---|---|
| **User** | MySQL username and host (e.g., `user@host`) |
| **Active** | Indicates whether the account is currently active |
| **Privileges** | Displays granted privileges |

Click on **Refresh** instructs ClusterControl to read the database user information for all hosts, and aggregate them in one view.

Privileges are grouped for clarity in boxes. The left-side of the box contains the granted scope of the database objects, for example:

- `*.*` - Global privileges. The granted privilege applies to all databases and all tables.
- `performance_schema` - The granted privilege applies to all tables/views for database `performance_schema`.
- `performance_schema.hosts` - The granted privilege applies to table/view of `hosts`, under database `performance_schema`.

The right-side of the box contains the comma-separted list of granted privileges, for example, `ALTER`, `CREATE`, `ALTER ROUTINE`, `REPLICATION CLIENT PROCESS` and `SELECT`. Administrative privileges such as `GRANT OPTION` or `PROXY` are also shown when applicable.

### ClusterControl CLI

- Lists all database users for cluster ID 1:

```
s9s account \
    --list \
    --cluster-id=1
```

- Lists all database users for cluster ID 1, complete with the privileges for every database objects:

```
s9s account \
    --list \
    --long \
    --cluster-id=1
```

- Show all database users `myapp` from cluster ID 1, complete with their privileges:

```
s9s account --list --long --cluster-id=1 | grep myapp
```

### ClusterControl RPC API

- Get all database user accounts for cluster ID 1, authenticated as ClusterControl user `ccreader` that has view-only permission level:

```
curl -k -XPOST \
-d '{
    "operation": "listAccounts",
    "cluster_id": 1,
    "authenticate":
        {
        "username": "ccreader",
        "password": "thesupers3curepassw0rd"
        }
    }' \
https://localhost:9501/v2/clusters
```

## Create a MySQL/MariaDB user

### ClusterControl GUI

1. Click on **Create DB User** to open the user creation wizard.
2. In the **Create database user** panel, specify the following:
    - **Username**: Database username.
    - **Password**: Password for ***Username***. The minimum requirement is 4 characters.
    - **Hostname**: Hostname, IP address, or IP address range of the user or client. For the IP address range, use the SQL wildcard `%`, for example, `192.168.10.%` to allow 192.168.10.0/24.

- **Privileges**: Use the statement builder to add more grant statements. Click on **Add Privileges** to add another grant statement builder for this user.
  - **Grant**: Specify the privilege for this user. If the Privileges text box is active, it list all possible privileges of the server.
  - **on**: Specify the database or table name. It can be in `*.*`,`{database_name}`, `{database_name}.*` or `{database_name}.{table_name}` format.
- **Advanced**: Includes fine-grained control over the usage of this user.
  - **Max Queries Per Hour**: Maximum queries this user can perform in an hour. Default is 0 (unlimited).
  - **Max Updates Per Hour**: Maximum update operations this user can perform in an hour. Default is 0 (unlimited).
  - **Max Connections Per Hour**: Maximum connections allowed for this user in an hour. Default is 0 (unlimited).
  - **Max User Connections**: Maximum connections allowed for this user ever. Default is 0 (unlimited).
  - **Requires SSL**: Toggle **On** if the user must be authenticated using SSL.

3. To create the user, click **Create**. ClusterControl manages the entire process where it ensures the changes are made on the correct database node (the primary) and any relevant load balancer nodes, and then it propagates the updated user to all other database nodes, including flushing and loading the changes for runtime effect.
4. A toast message will appear on the top-right indicating the status of the user creation. If no error, the user will be listed in the User List view.

### ClusterControl CLI

- Create a new MySQL user account called `myuser` with password `secr3tP4ss`, and allow it to have `ALL PRIVILEGES` on database `shop_db` while `SELECT` only on table `account_db.payments`:

```
s9s account \
    --create \
    --cluster-id=1 \
    --account="myuser:secr3tP4ss@192.168.0.100" \
    --privileges="shop_db.*:ALL;account_db.payments:SELECT"
```

- Create a new MariaDB user account called `mydbuser` on cluster ID 50, and allow this user to access from all hosts in network 192.168.0.0/24 to database `mydbshop`:

```
s9s account \
    --create \
    --cluster-id=50 \
    --account='mydbuser:Passw0rd999@192.168.0.%' \
    --privileges="mydbshop.*:ALL"
```

- Create a new MySQL user account called `myapp` on cluster ID 1, and create a database with the same name `myapp` and assign the user with `ALL PRIVILEGES` to the database:

```
s9s account \
    --create \
    --cluster-id=1 \
    --account='myapp:Passw0rd999@192.168.0.%' \
    --with-database
```

**Example**

```
$ s9s account \
    --create \
    --cluster-id=1 \
    --account='myapp:Passw0rd999@192.168.0.%' \
    --with-database

Using plugin ''.
192.168.73.11:3306: Server supports new MySQL authentication syntax.␣
↪Server has version 8.4.5-5.1.
Account 'myapp' created.
Database 'myapp' created.
Access for 'myapp' to 'myapp' granted.
Importing account on ProxySql host(s).
Created.
```

### ClusterControl RPC API

- Create a new database user called `mydbuser2` for cluster ID 1 and allow it to access from 192.168.10.0/24 with `ALL PRIVILEGES` for database `mydb2` and SELECT only for database `myapp`, authenticated as ClusterControl user `ccrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "createAccount",
    "cluster_id": 1,
    "account":
        {
       "user_name": "mydbuser2",
       "password": "pass123",
       "host_allow": "192.168.10.%",
       "grants": "mydb2:ALL;myapp:SELECT"
        },
    "authenticate":
        {
        "username": "ccrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
    }' \
https://localhost:9501/v2/clusters
```

!!! Info

```
  If the same `createAccount` operation is called twice with an␣
↪identical account information, the latter call will be automatically␣
↪converted to an `ALTER` operation (edit an existing database user).
```

### Modify MySQL/MariaDB users

#### ClusterControl GUI

1. Click on the action button **...** → ***Edit*** to edit an existing user.
2. In the **Edit database user** panel, specify the following:
   - **New Password**: Password for the database user. The minimum requirement is 4 characters.
   - **Privileges**: Use the statement builder to add more grant statements. Click on **Add Privileges** to add another grant statement builder for this user.
     - **Grant**: Specify the privilege for this user. If the Privileges text box is active, it list all possible privileges of the server.
     - **on**: Specify the database or table name. It can be in `*.*`,`{database_name}`, `{database_name}.*` or `{database_name}.{table_name}` format.
   - **Advanced**: Includes fine-grained control over the usage of this user.
     - **Max Queries Per Hour**: Maximum queries this user can perform in an hour. Default is 0 (unlimited).
     - **Max Updates Per Hour**: Maximum update operations this user can perform in an hour. Default is 0 (unlimited).
     - **Max Connections Per Hour**: Maximum connections allowed for this user in an hour. Default is 0 (unlimited).
     - **Max User Connections**: Maximum connections allowed for this user ever. Default is 0 (unlimited).
     - **Requires SSL**: Toggle **On** if the user must be authenticated using SSL.
3. To modify the user's settings, click **Save**. ClusterControl manages the entire process where it ensures the changes are made on the correct database node (the primary) and any relevant load balancer nodes, and then it propagates the updated user to all other database nodes, including flushing and loading the changes for runtime effect.
4. A toast message will appear on the top-right indicating the status of the user modification. If no error, the user will be listed in the User List view.

#### ClusterControl CLI

- List all database users to find the one that you want to edit:

```
s9s account \
    --list \
    --long \
    --cluster-id=1
```

- Modify an existing database user `'dataadmin'@'%'` by adding another MySQL administrative privilege called `PROCESS`:

```
s9s account \
    --grant \
    --cluster-id=1 \
    --account='dataadmin@%' \
    --privileges="PROCESS"
```

- Reset the password for user `'mydb'@'192.168.73.15'` to `MySuperDuperS3cureP455w0rd`:

```
s9s account \
    --create \
    --cluster-id=1 \
    --account='mydb:MySuperDuperS3cureP455w0rd@192.168.73.15'
```

**Example**

```
$ s9s account \
    --create \
    --cluster-id=1 \
    --account='mydb:MySuperDuperS3cureP455w0rd@192.168.73.15'

192.168.73.11:3306: Found auth plugin 'caching_sha2_password' for account
↪'mydb'@'192.168.73.15'.
Updating account 'mydb'@'192.168.73.15'.
192.168.73.11:3306: Alter user processed: ALTER USER 'mydb'@'192.168.73.15
↪' IDENTIFIED WITH caching_sha2_password BY 'xxx xx'
Importing account on ProxySql host(s).
Created.
```

```
!!! Info

    The `--create` flag is capable of updating an existing user, as well as␣
↪creating a new user if does not exist.
```

- Modify an existing database user `'user1'@'192.168.10.14'` by adding a number of database privileges on table `db1.reports`:

```
s9s account \
    --grant \
    --cluster-id=1 \
    --account='user1@192.168.10.14' \
    --privileges="db1.reports:SELECT,INSERT,UPDATE"
```

- Modify an existing database user `'user1'@'192.168.10.14'` by revoking `INSERT` and `UPDATE` database privileges from table `db1.reports`:

```
s9s account \
    --revoke \
    --cluster-id=1 \
    --account='user1@192.168.10.14' \
```

(continues on next page)

(continued from previous page)

```
    --privileges="db1.reports:INSERT,UPDATE"
```

### ClusterControl RPC API

- Reset the password for an existing database user `'mydbuser2'@'192.168.73.14'` on cluster ID 1 to `mySuperSecr3tP455` and limit the database user connections to 30, authenticated as ClusterControl user `myrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "updateAccount",
    "cluster_id": 1,
    "account":
        {
       "user_name": "mydbuser2",
       "password": "mySuperSecr3tP455",
       "host_allow": "192.168.73.14",
       "max_user_connections": 30
        },
    "authenticate":
        {
        "username": "myrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
    }' \
https://localhost:9501/v2/clusters
```

- Modify privileges for an existing database user `'mydbuser2'@'192.168.73.14'` on cluster ID 1 by adding `DELETE` and `TRUNCATE` to database `testDB`, authenticated as ClusterControl user `myrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "grantPrivileges",
    "cluster_id": 1,
    "privileges": "testDB.*:DELETE,TRUNCATE",
    "account":
        {
       "user_name": "mydbuser2",
       "host_allow": "192.168.73.14"
        },
    "authenticate":
        {
        "username": "myrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
    }' \
https://localhost:9501/v2/clusters
```

### Drop MySQL/MariaDB users

When dropping a database user, ClusterControl manages the entire process where it ensures the removals are made on the primary database node, or one of the primary nodes if it is a multi-primary cluster.

### ClusterControl GUI

1. Click on the action button **…** → ***Delete*** and click **Delete** to confirm removing an existing user.
2. A toast message will appear on the top-right indicating the status of the user deletion.

### ClusterControl CLI

- Drop a database user `'joe'@'%'` on cluster ID 1:

```
s9s account \
    --delete \
    --cluster-id=1 \
    --account="joe"
```

> **Tip**
>
> If a host value is missing in the `--account` option, the host value will default to `%`.

- Drop a database user `'mydb2'@'192.168.40.%'` on cluster ID 1:

```
s9s account \
    --delete \
    --cluster-id=1 \
    --account='mydb2@192.168.40.%'
```

### ClusterControl RPC API

- Drop a database user `'mydbuser2'@'192.168.73.15'` for cluster ID 1, authenticated as ClusterControl user `myrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "deleteAccount",
    "cluster_id": 1,
    "account":
        {
       "user_name": "mydbuser2",
       "host_allow": "192.168.73.15"
        },
    "authenticate":
        {
        "username": "myrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
    }' \
https://localhost:9501/v2/clusters
```

**Warning**

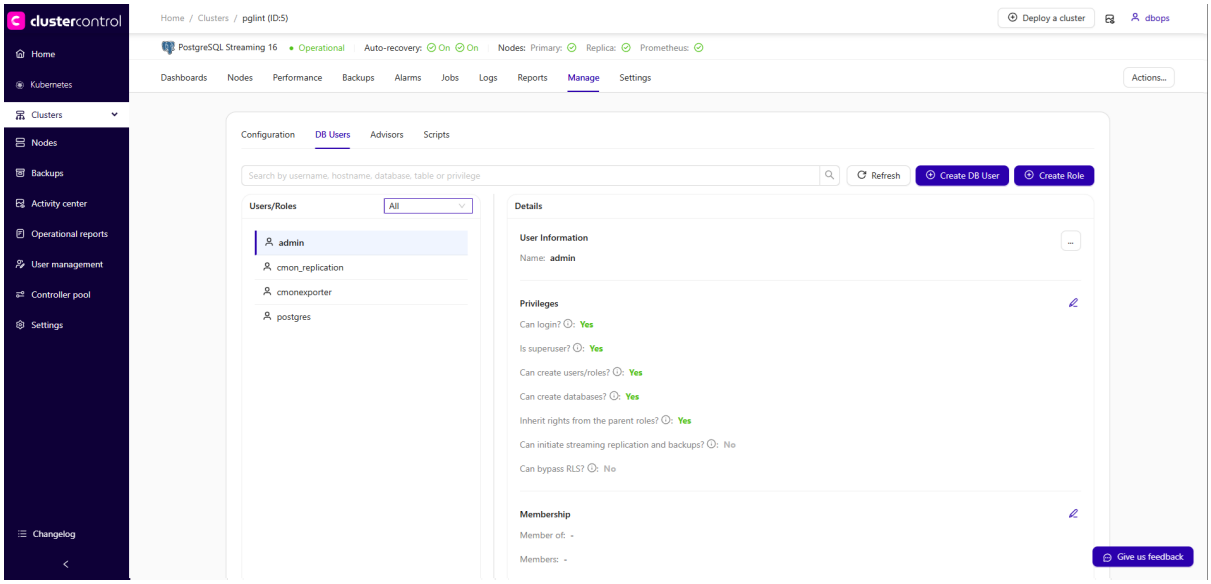
Attention

If the database cluster has a ProxySQL load balancer, ClusterControl would create the corresponding user on the ProxySQL as well. Deleting the database user here does not reflect the ProxySQL user. Separate deletion on ProxySQL is necessary for a full user removal.

### 2.50.4 PostgreSQL-based clusters

Since the release of ClusterControl 2.4.0, you can manage roles, enforce password policies, and configure fine grained permissions (down to the table level) directly from the GUI. It's also improved the user experience and enhanced `pg_hba.conf` synchronization to ensure consistent security settings across nodes and reduce connectivity issues.

To get started, navigate to ***Cluster → Manage → DB Users***. From there, you can create or modify roles, assign privileges, and adjust password policies through a guided interface.




#### Common system users for PostgreSQL

ClusterControl automatically manages several system users required for cluster operation:

| User | Purpose |
|---|---|
| cmon | Internal cluster management |
| cmonagent | ClusterControl query monitoring agent |
| backupuser | Database backup and restore |
| cmonexporter | Database metrics collection (Prometheus exporter) |
| rpl_user | Database replication |

**Warning**

Warning!

Modifying these users may impact cluster functionality.

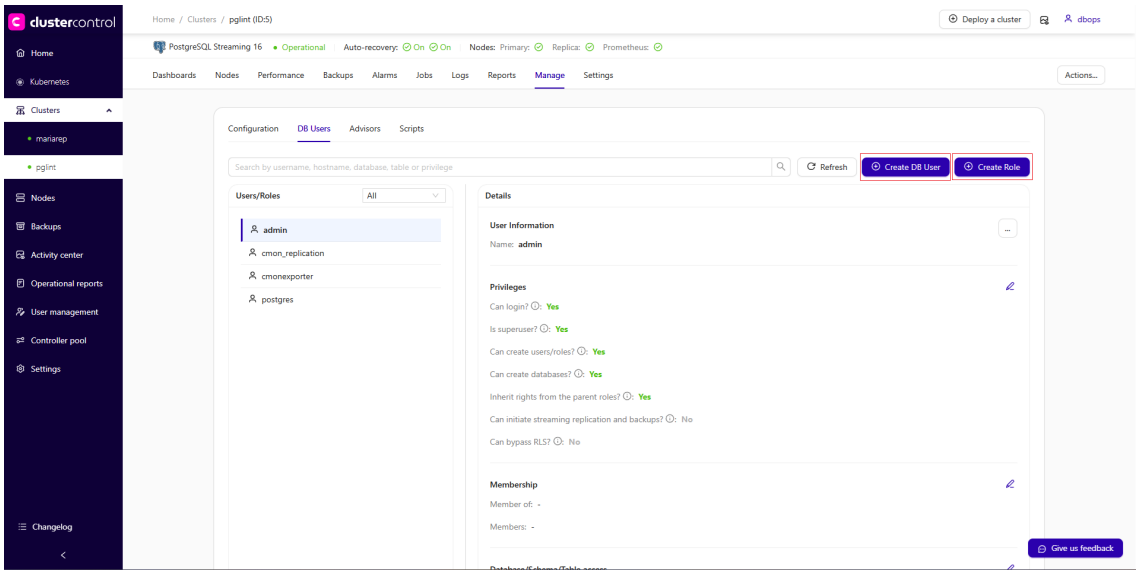
### Roles and Users creation

PostgreSQL role and user creation can be fully managed from the GUI. As a best practice, we recommend creating roles first, defining the required privileges at the role level, and then assigning those roles to users. This approach improves access control management, simplifies permission updates, and helps maintain consistency across your PostgreSQL environment.

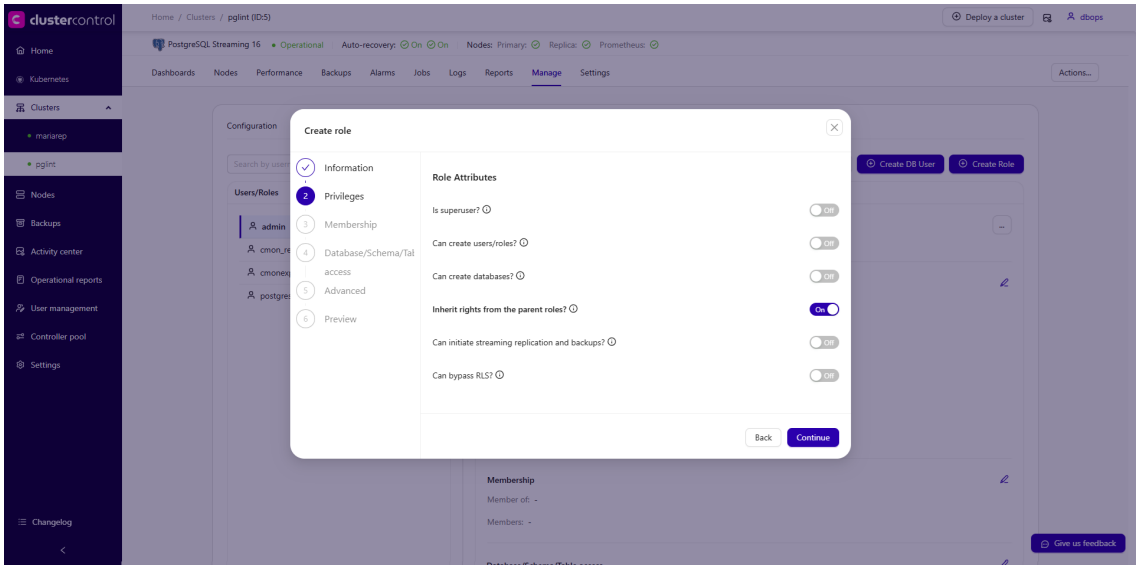
To create and configure roles or users, go to the *DB Users* section in ClusterControl, where you can define role attributes, assign privileges, and manage role memberships.

### ClusterControl GUI

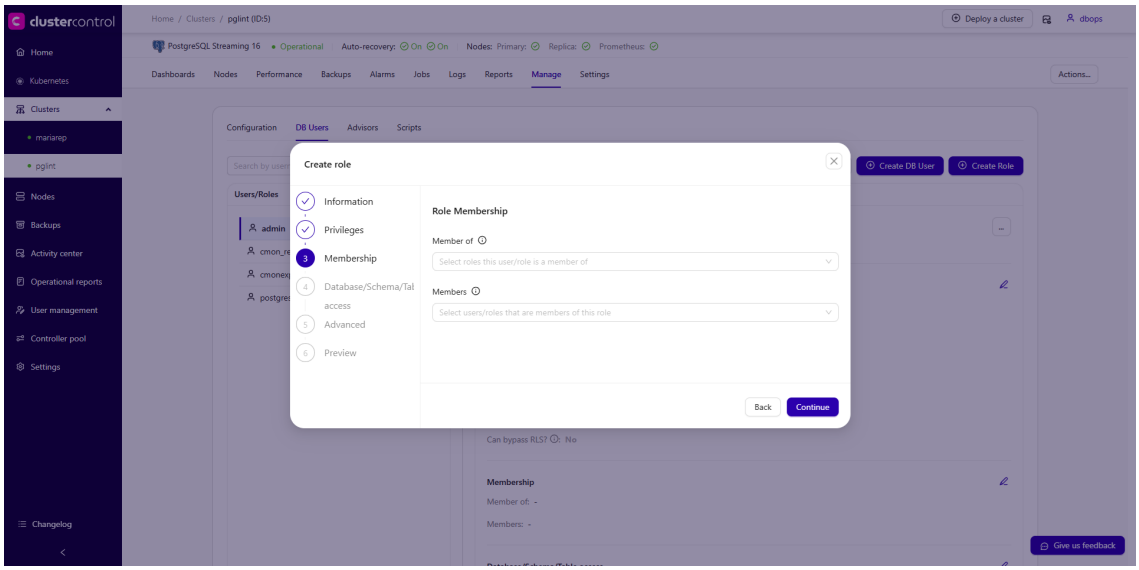
1. On the **DB Users** menu, click **Create Role** to create a new role and click **Create DB User** to create a new user.




2. A dialog will appear. Complete the following steps.
3. **Information**
   - For role creation, enter the role name in the **Role Name** input field.
   - For user creation, enter the **Username** and **Password** in the respective input fields.
4. **Role Privileges:** Configure the appropriate privileges for the role.




5. **Role Membership:** Configure role memberships or inheritance as needed. In this section, you can define both "Member of" and "Members" relationships.

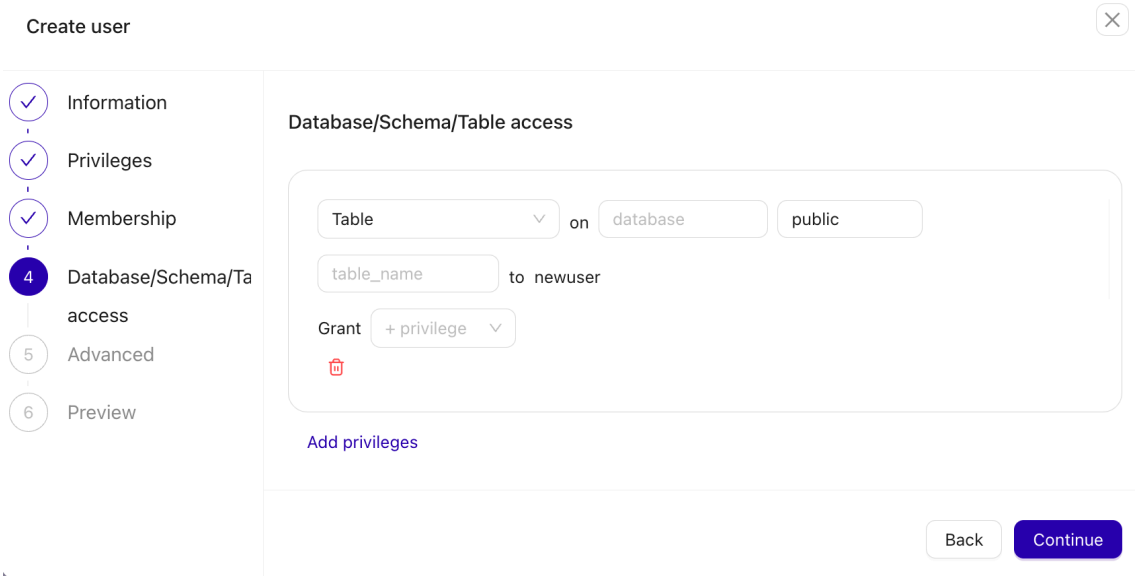



   - `Member of`: Assign this role to existing roles so it inherits their privileges.
   - `Members`: Add other roles or users as members of this role, allowing them to inherit the privileges defined here.

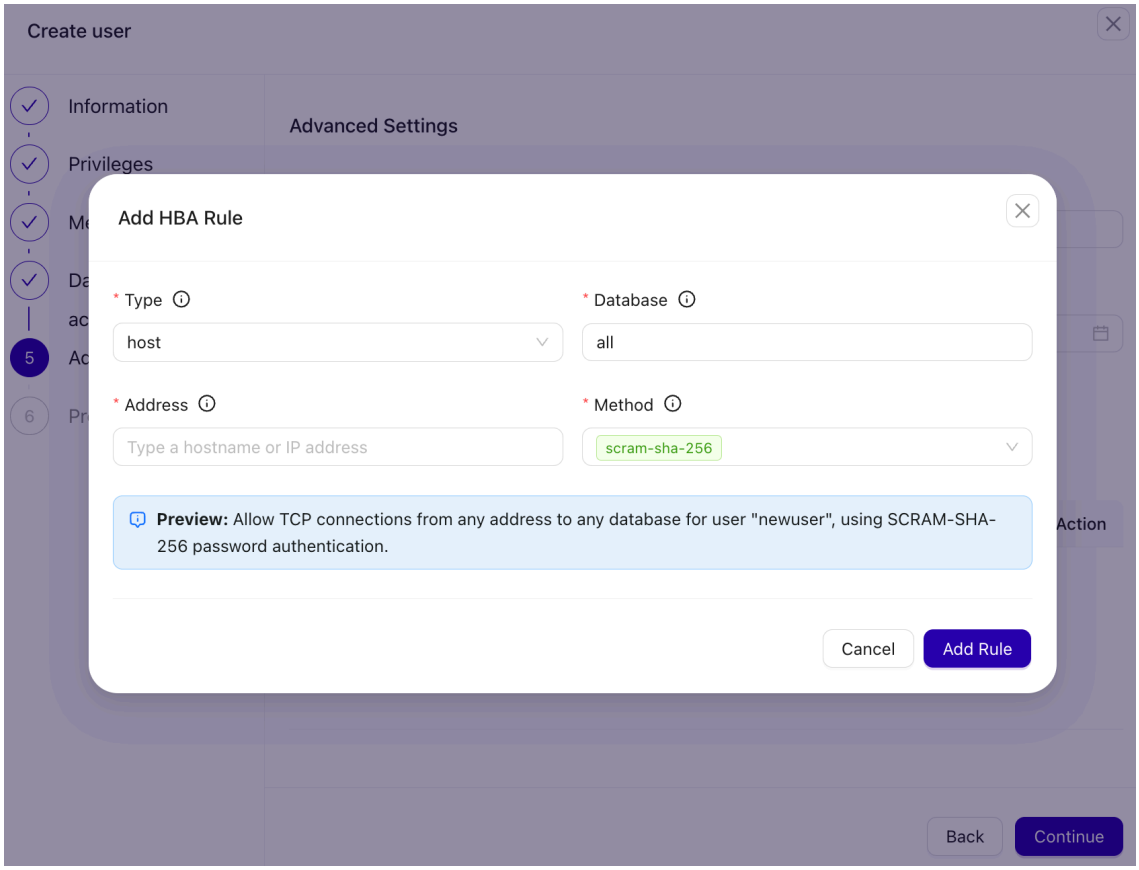
6. **Database/Schema/Table access:** Configure access at the database, schema, or table level using explicit object-type selectors instead of writing out a wildcard grant string.

   - **Object type**: Combo box to choose the object level to grant access to: `Database`, `Schema`, `Table`, `All tables in Schema`, `Sequence`, `All sequences in Schema`, `Function/Procedure`, or `All functions in schema`.
   - The text boxes shown next to **on** depend on the object type selected, since each type needs a different set of identifiers to locate the object. For example, choosing `Table` shows text boxes for **database**, **schema** (defaults to `public`), and **table_name**; choosing `Database` shows only a **database** text box.
   - The grant always applies **to** the role or user being created; this is shown as read-only text, not an editable field.

- **Grant**: Combo box to select one or more privileges available for the selected object type.
- Click the trash icon to remove a line, or **Add privileges** to add another line with the same object type, text box, and grant structure, so you can configure multiple privileges in the same section.



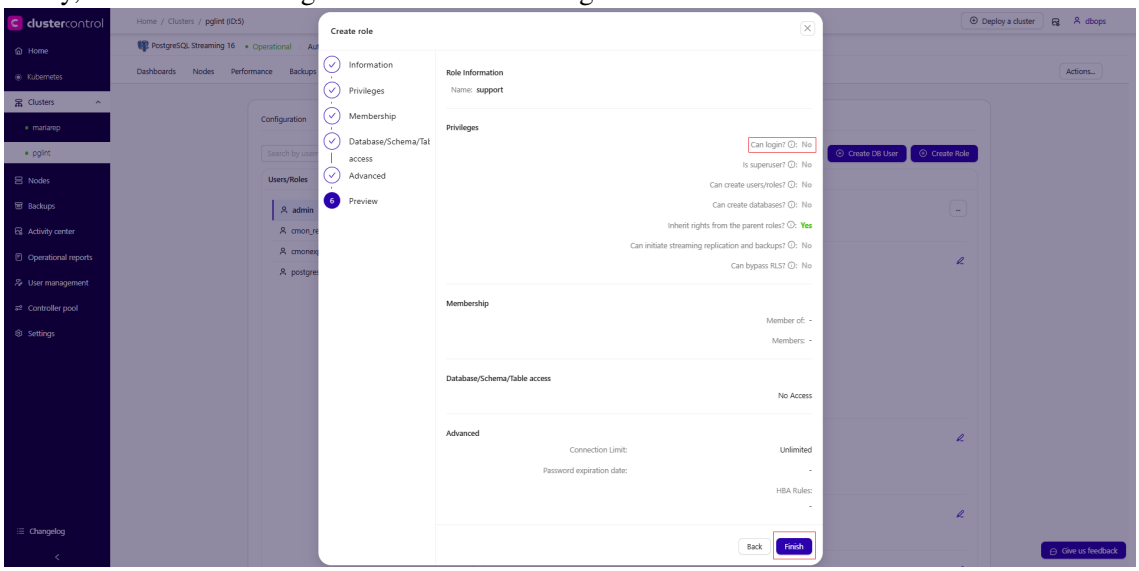

7. **Advanced:** Configure additional options, including connection limits, password expiration, and pg_hba settings.
   - **Connection limit**: Maximum concurrent connections (`-1` = unlimited, `0` = no connections). Default `-1`.
   - **Password expiration** (optional): Date box for when the account password expires. Empty by default.
   - **HBA Rules**: Click **Add HBA Rule** to open a form with **Type**, **Database**, **Address**, and **Method** fields; there is no separate **User** field here, since the rule always applies to the account you are creating (shown in the live preview, for example `for user "newuser"`).




- **Type:** Specifies the connection type. Common options:
  - `host`: Matches TCP/IP connections (IPv4 and IPv6) and standard network connections
  - `hostssl`: Matches TCP/IP connections that require SSL/TLS
  - `local`: Matches Unix socket connections (local machine only)
- **Database:** Specifies which database(s) the rule applies to.
  - `all`: Applies to all databases (default)
  - A specific database name (for example: `postgres`, `template1`, `product`)
- **Address:** Defines the client source address allowed to connect. Supported formats:
  - Single IP address (`10.10.0.25/32`)
  - Subnet or CIDR notation (`10.10.0.0/24`)
  - All IPv4 addresses (`0.0.0.0/0`): *not recommended*
  - IPv6
- **Method:** Specifies the authentication method used to validate client connections. Common methods:

| Method | Description | Typical Use Case | Security Level |
|---|---|---|---|
| `trust` | Allows connection without authentication. | Testing or internal environments only. | Not secure |
| `reject` | Explicitly denies a connection attempt. | Blocking specific IPs or subnets. | - |
| `scram-sh` | Secure password-based authentication using SCRAM. | Recommended for production environments. | High |
| `md5` | Legacy password authentication using MD5 hashing. | Backward compatibility with older clients. | Medium |
| `password` | Sends password in clear text (use only with SSL). | Legacy systems requiring plain password auth. | Low (without SSL) |
| `gss` | Uses GSSAPI (Kerberos). | Enterprise environments with Kerberos SSO. | High |
| `sspi` | Windows SSPI authentication (Active Directory). | Windows domain environments. | High |
| `ident` | Uses ident protocol to verify OS user over TCP/IP. | Legacy or controlled internal networks. | Low Medium |
| `peer` | Authenticates based on matching OS user (Unix socket only). | Local Linux connections. | High (local only) |
| `ldap` | Authenticates against an LDAP server. | Centralized enterprise authentication. | High (depends on LDAP config) |
| `radius` | Uses a RADIUS server for authentication. | Enterprise centralized authentication. | High |
| `cert` | Uses SSL client certificate authentication. | High security environments requiring mutual TLS. | Very High |
| `pam` | Uses Pluggable Authentication Modules (PAM). | Linux systems integrating with system authentication. | High (depends on PAM setup) |

8. Lastly, review the configuration before clicking the Finish button in the **Preview** section.



### ClusterControl CLI

Create a new PostgreSQL role called `vendor`, and allowed the host in the network subnet 10.10.20.0/24 to access the database `postgres` with readonly privilege:

```
s9s account \
    --create \
    --cluster-id=5 \
    --account='vendor@10.10.20.0/24' \
    --privileges="postgres.*:SELECT"
```

Create a new PostgreSQL user called `vender_abc`

```
s9s account \
    --create \
    --cluster-id=5 \
    --account='vender_abc:Passw0rd999'
```

The difference between a Role and a User in s9s cli is the password. During creation, an account created with a password will be flagged as a User, while an account created without a password will be flagged as a Role.

### ClusterControl RPC API

This request creates a role or user in PostgreSQL User Management (v2). For example, it creates a role named `vendor` with allowed host `10.10.20.0/24` and `readonly` privileges on tables in the `postgres` database.

```
curl -k -XPOST \
-d '{
    "cluster_id": 5,
    "operation": "createAccount",
    "account": {
    "class_name": "CmonAccount",
    "user_name": "vendor",
    "ssl_type": "NONE",
    "max_user_connections": 0,
    "max_questions": 0,
    "max_updates": 0,
    "max_connections": -1,
    "member_of": [],
    "members": [],
    "allowed_hosts": [
        {
        "type": "host",
        "database": "all",
        "user": "vendor",
        "address": "10.10.20.0/24",
        "method": "scram-sha-256",
        "inherited": false
        }
    ],
```

(continues on next page)

(continued from previous page)

```
    "grants":
↪"NOSUPERUSER,NOLOGIN,INHERIT,NOCREATEROLE,NOCREATEDB,NOREPLICATION,NOBYPASSRLS;postgres
    },
    "authenticate": {
    "username": "myrpcusr",
    "password": "mySuperStr0ngP@ssw0rd999"
    }
}' \
https://10.10.15.100:9501/v2/clusters
```

**Note**

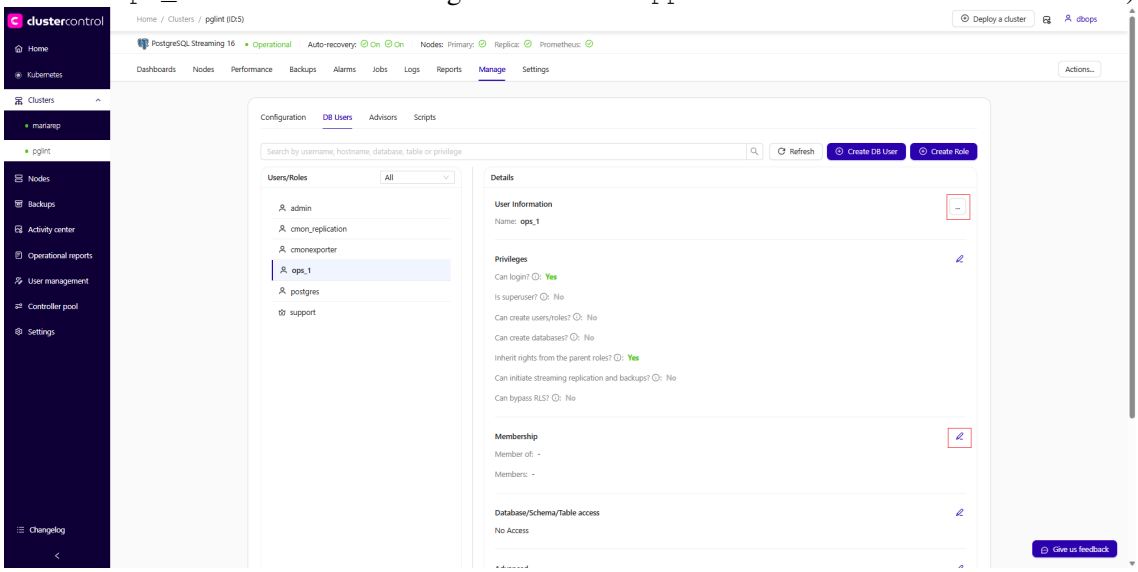
Note Roles created from the UI cannot log in to the database by default. You can verify this in the **Preview section**, where it shows: `Can login? : No`

### Roles and Users modification

After creating roles and users, you can modify them as needed. This includes changing passwords, updating privileges (LOGIN, SUPERUSER, CREATEDB, CREATEROLE), managing role memberships, and updating `pg_hba.conf` policies. To make changes, navigate to the management section in the UI.

### ClusterControl GUI

1. Navigate to **Cluster → Manage → DB Users**.
2. Select the role or user you want to modify.
3. Click on the action button **… → Edit** or click the **edit (pencil)** icon on each section.
4. Modify the user or role. (for example, modifying the user `ops_1` and attaching the `support` role to the user)




5. Review the changes and ensure everything is correct
6. Click **Save** to apply the updates.

### ClusterControl CLI

Coming soon (To be updated once available)

### ClusterControl RPC API

This request modifies a role or user in PostgreSQL User Management (v2). For example, it modifies user `vendor_abc` to become a member of role `vendor`.

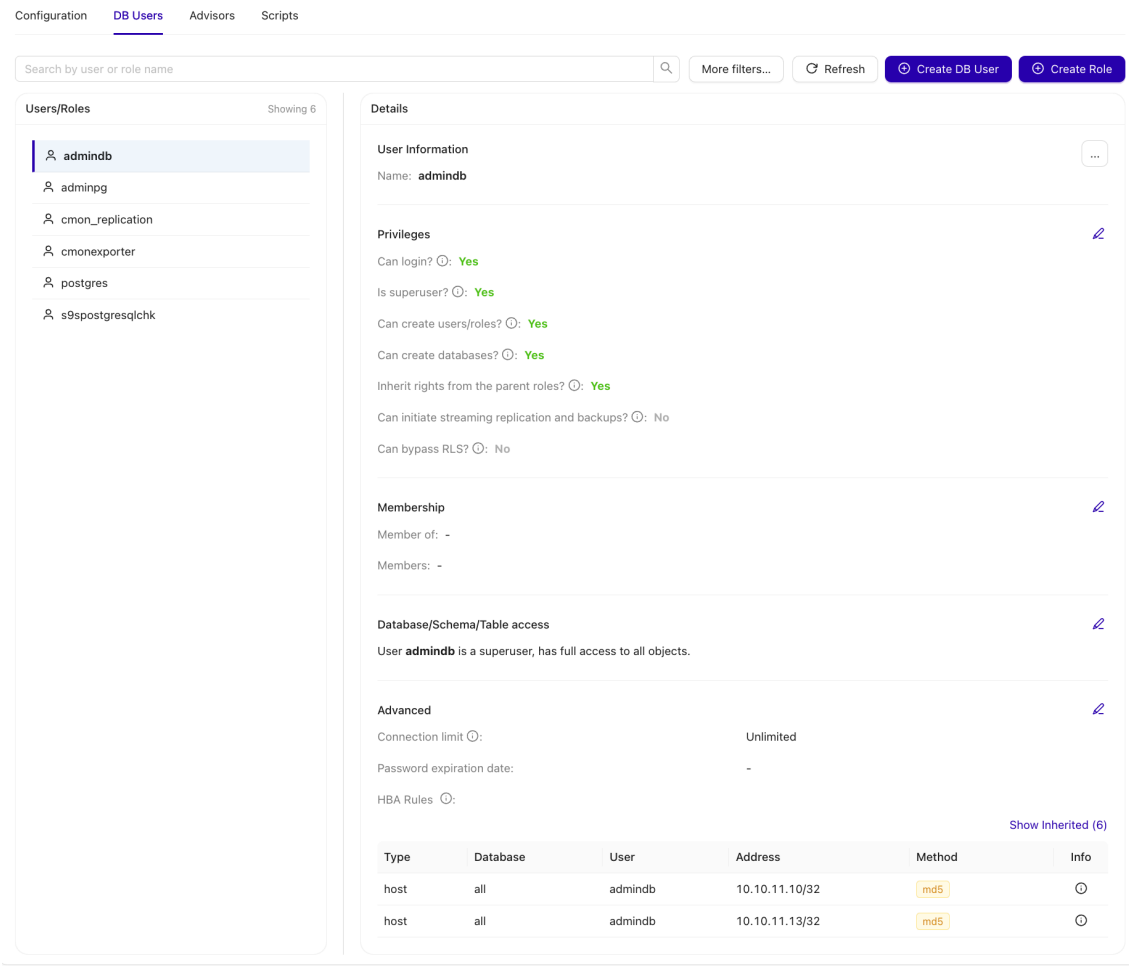
```
curl -k -XPOST \
-d '{
    "cluster_id": 5,
    "operation": "updateAccount",
    "account": {
    "class_name": "CmonAccount",
    "user_name": "vendor_abc",
    "ssl_type": "NONE",
    "max_connections": -1,
    "member_of": [
        {
        "name": "vendor",
        "with_admin": false
        }
    ],
    "members": [],
    "grants":
↪"NOSUPERUSER,LOGIN,INHERIT,NOCREATEROLE,NOCREATEDB,NOREPLICATION,NOBYPASSRLS"
    },
    "authenticate": {
    "username": "myrpcusr",
    "password": "mySuperStr0ngP@ssw0rd999"
    }
}' \
https://10.10.15.100:9501/v2/clusters
```

### pg_hba.conf management

Beyond adding a rule during role or user creation (see **Advanced** in *Roles and Users creation*), you can manage the full set of `pg_hba.conf` entries for a cluster from any user's **Edit** dialog.

### ClusterControl GUI

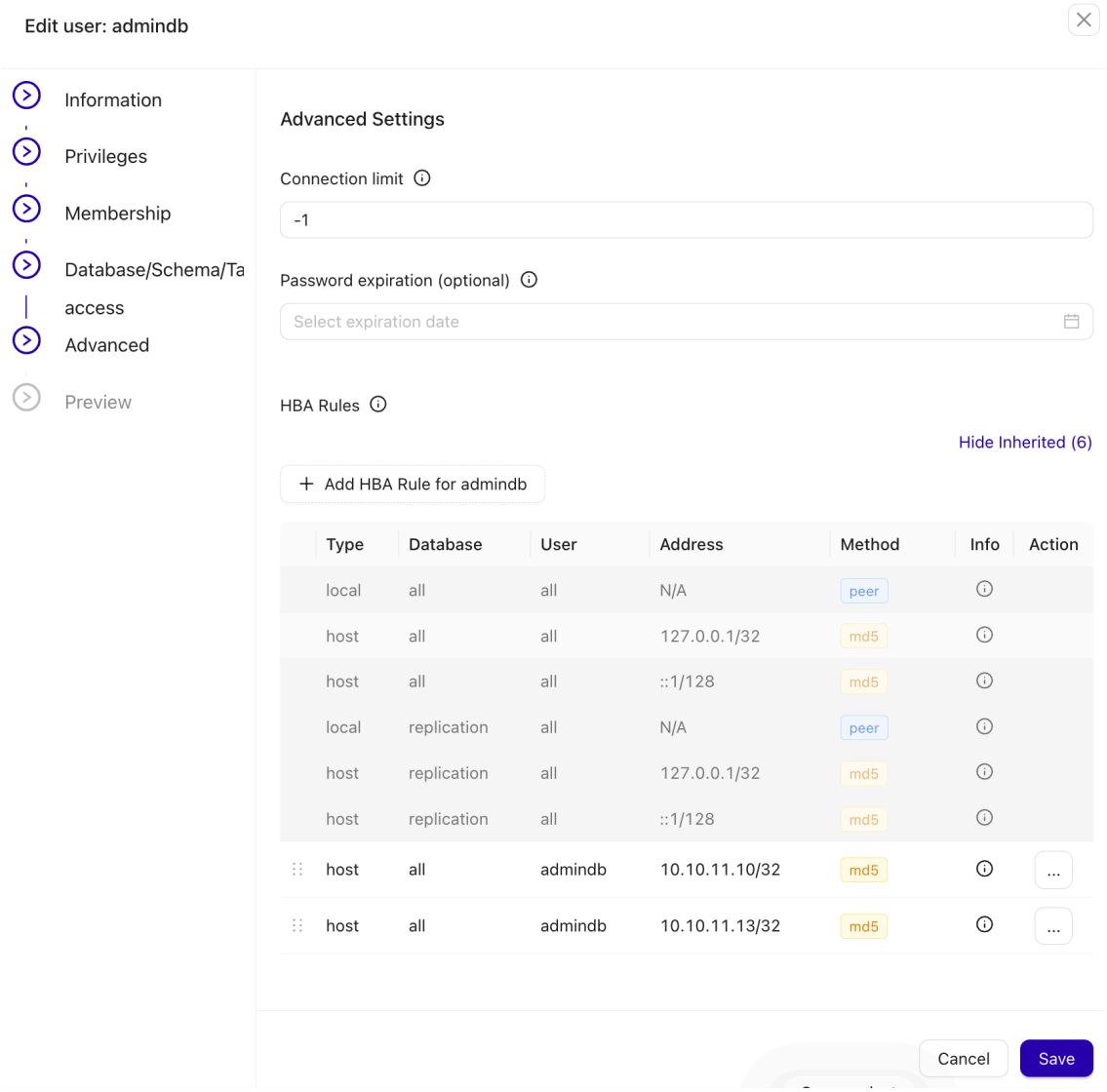
1. Navigate to **Cluster → Manage → DB Users**.
2. Select a user to see its read-only **Details** panel, which lists the HBA rules that apply to that user under **Advanced** (**Type**, **Database**, **User**, **Address**, **Method**, **Info**), or click **Edit** on a user to manage the full rule list.




3. In the **Edit** dialog, under **Advanced → HBA Rules**, use **Show Inherited / Hide Inherited** (labeled with a count, for example **Hide Inherited (6)**) to toggle the display of PostgreSQL's built-in and ClusterControl's default rules alongside your own.

4. Click **Add HBA Rule for `<username>`** to define a new rule, using the same **Type**, **Database**, **Address**, and **Method** fields, defaults, and live preview text described in *Roles and Users creation*. As with the create-user form, there is no separate **User** field; the rule always applies to the account whose **Edit** dialog you're in.

5. Only custom (non-inherited) rules show a drag handle and an **…** button in the **Action** column, letting you reorder them (by dragging, or via **Move to top**, **Move to bottom**, **Move up**, **Move down** in the menu) or **Edit/Delete** them. Inherited/default rules show neither and cannot be moved.




6. Every change is tracked, so you can see the history of modifications made to the file over time.

**Tip**

To view the raw `pg_hba.conf` file as a plain text file instead of the rule editor above, go to **Cluster → Manage → Configurations**. See *Configuration Management*.

## User lifecycle operations

In addition to creating and editing users, you can change a user's ability to log in without deleting the account:

### ClusterControl GUI

1. Navigate to **Cluster → Manage → DB Users**.
2. Select the target user and click on the action button **… → Lock**, **Disable**, or **Enable**.
   - **Lock**: temporarily prevents login while preserving the account, its roles, and its privileges.
   - **Disable**: turns off login capability for the account.

- **Enable**: restores login capability for a previously locked or disabled account.

#### Search and filtering

The **DB Users** list can be filtered by role or status (for example, to quickly find every locked or disabled user), using the search box above the user list.

### 2.50.5 Redis-based clusters

The Access Control feature in ClusterControl for Redis-based clusters (including Valkey cluster, Redis Sentinel and Valkey Sentinel setups) allows administrators to manage Redis ACL users, permissions, and security policies. This functionality is built upon Redis's native permissions and Access Control List (ACL) mechanism, which is detailed in the Redis documentation and Valkey documentation. For ClusterControl command-line users, the same management features are accessible via the `account` command.

This feature simplifies the configuration of Redis authentication and authorization by providing a centralized interface to define:

- Users
- Command permissions
- Key access patterns
- Channel access
- Categories

#### Common system users for Redis

ClusterControl automatically manages several system users required for Redis/Valkey cluster operation:

| User | Purpose |
|---|---|
| default | Default Redis user |
| replica-user | Database replication |
| sentinel-user | Redis/Valkey Sentinel user for failover coordination |

**Warning**

Warning!

Modifying these users may impact cluster functionality.

#### View Access Control List (ACL)

#### ClusterControl GUI

The access control list provides a tabular overview of all database users. In the main panel, you can filter the list by filing up the textbox with the magnifying glasss icon.

The ACL table displays the following information for each user:

| Field | Description |
|---|---|
| User | Redis ACL username |
| Status | Whether the user is enabled or disabled |
| Categories | Command categories assigned (e.g., `+@all`, `-@all`) |
| Channels | Pub/Sub channel access patterns |
| Keys | Key pattern access (e.g., `~*`) |
| Commands | Specific allowed commands (e.g., `+get`, `+set`) |

Click on **Refresh** instructs ClusterControl to read the database user information for all hosts, and aggregate them in one view.

### ClusterControl CLI

- Lists all database users for cluster ID 1.

```
s9s account \
    --list \
    --cluster-id=1
```

- Lists all database users for cluster ID 1, complete with the privileges for every database objects:

```
s9s account \
    --list \
    --long \
    --cluster-id=1
```

## Create a Redis user

### ClusterControl GUI

1. Click on **Create ACL rule** to open the user creation wizard.
2. In the **Create ACL rule** panel, specify the following:
    - **Username**: Database username. This field is mandatory.
    - **Password**: Password for ***Username***. If no password is specified user will be set with `nopass` special string. It means that it will be possible to authenticate as such user with any password.
    - **Categories**: Predefined groups of commands in Redis ACLs used to manage user permissions. Examples include `+@admin` for administrative commands, `+@read` for read-only commands, and `+@write` for write commands. This field is mandatory.
    - **Channels**: Custom channels in Redis Pub/Sub messaging that users can be allowed or denied access to via ACLs. Channels facilitate the broadcasting of messages to multiple subscribers.
    - **Keys**: Custom channels in Redis Pub/Sub messaging that users can be allowed or denied access to via ACLs. Channels facilitate the broadcasting of messages to multiple subscribers.
    - **Commands**: Specific Redis commands that users can be allowed or denied access to in ACLs. Commands perform various operations such as setting values (`set`), retrieving data (`get`) or deleting keys (`del`). Prefix the command with plus '+' to allow or '–' to deny.

3. To create the user, click **Create**. ClusterControl manages the entire process where it ensures the changes are made on the correct database node (the primary), and ensure the user is propagated to all other database nodes.
4. A toast message will appear on the top-right indicating the status of the user creation. If no error, the user will be listed in the list view.

### ClusterControl CLI

- Create a new Redis user account called `cache1` with password `secr3tP4ss` on cluster ID 4, and allow this user all commands except the dangerous ones for all channels (`&*`) and keys(`~*`):

```
s9s account \
    --create \
    --cluster-id=4 \
    --account='cache1:secr3tP4ss' \
    --privileges='+@all -@dangerous'
```

- Create a new non-password Redis user called `nopassuser` on cluster ID 4, and allow this user to run all list-related commands on all channels and keys:

```
s9s account \
    --create \
    --cluster-id=4 \
    --account='nopassuser' \
    --privileges='+@list'
```

### ClusterControl RPC API

- Create a new Redis user called `mycache` for cluster ID 1 and allow this user to run all admin-related commands except the dangerous ones on all keys and channels, authenticated as ClusterControl user `ccrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "createAccount",
    "cluster_id": 1,
    "account":
        {
       "user_name": "mycache",
       "password": "passW0rd123",
       "host_allow": "%",
       "grants": "+@admin -@dangerous"
        },
    "authenticate":
        {
        "username": "ccrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
    }' \
https://localhost:9501/v2/clusters
```

!!! Info

```
  If the same `createAccount` operation is called twice with an␣
↪identical account information, the latter call will be automatically␣
↪converted to an `ALTER` operation (edit an existing database user).
```

### Modify Redis users

#### ClusterControl GUI

1. Click on the action button **...** → ***Edit*** to edit an existing ACL rule.
2. In the **Edit ACL rule** panel, specify the following:
   - **Password**: New password for the user. If no password is specified user, the current password will be kept. If you want to set the user without password (with `nopass` special string), create a new user with blank password instead.
   - **Categories**: Predefined groups of commands in Redis ACLs used to manage user permissions. Examples include `+@admin` for administrative commands, `+@read` for read-only commands, and `+@write` for write commands. This field is mandatory.
   - **Channels**: Custom channels in Redis Pub/Sub messaging that users can be allowed or denied access to via ACLs. Channels facilitate the broadcasting of messages to multiple subscribers.
   - **Keys**: Custom channels in Redis Pub/Sub messaging that users can be allowed or denied access to via ACLs. Channels facilitate the broadcasting of messages to multiple subscribers.
   - **Commands**: Specific Redis commands that users can be allowed or denied access to in ACLs. Commands perform various operations such as setting values (`set`), retrieving data (`get`) or deleting keys (`del`). Prefix the command with plus '+' to allow or '–' to deny.
3. To modify the user's settings, click **Save**. ClusterControl manages the entire process where it ensures the changes are made on the correct database node (the primary), and ensure the user is propagated to all other database nodes.
4. A toast message will appear on the top-right indicating the status of the user modification. If no error, the user will be listed in the User List view.

#### ClusterControl CLI

- List all database users to find the one that you want to edit:

```
s9s account \
    --list \
    --long \
    --cluster-id=1
```

- Modify an existing database user `dataadmin` by allowing administrative commands called `@admin`:

```
s9s account \
    --grant \
    --cluster-id=1 \
    --account='dataadmin' \
    --privileges="+@admin"
```

- Reset the password for user `mydb` to `MySuperDuperS3cureP455w0rd`:

```
s9s account \
    --create \
    --cluster-id=1 \
    --account='mydb:MySuperDuperS3cureP455w0rd'
```

!!! Info

```
  The `--create` flag is capable of updating an existing user, as well␣
↪as creating a new user if does not exist.
```

### ClusterControl RPC API

- Reset the password for an existing database user `mydbuser` on cluster ID 1 to `mySuperSecr3tP455`, authenticated as ClusterControl user `myrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "updateAccount",
    "cluster_id": 1,
    "account":
        {
       "user_name": "mydbuser2",
       "password": "mySuperSecr3tP455"
        },
    "authenticate":
        {
        "username": "myrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
    }' \
https://localhost:9501/v2/clusters
```

- Modify privileges for an existing database user `mydbuser2` on cluster ID 1 by allowing `@admin` and denying `@dangerous` commands to all databases, authenticated as ClusterControl user `myrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "grantPrivileges",
    "cluster_id": 1,
    "privileges": "+@admin -@dangerous",
    "account":
        {
       "user_name": "mydbuser2"
        },
    "authenticate":
        {
        "username": "myrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
```

(continues on next page)

(continued from previous page)

```
    }' \
https://localhost:9501/v2/clusters
```

### Drop Redis users

When dropping a database user, ClusterControl manages the entire process where it ensures the removals are made on the primary database node, or one of the primary nodes if it is a multi-primary cluster.

### ClusterControl GUI

1. Click on the action button **…** → ***Delete*** and click **Delete** to confirm removing an existing user.
2. A toast message will appear on the top-right indicating the status of the user deletion.

### ClusterControl CLI

- Drop a Redis user 'joe' on cluster ID 1:

```
s9s account \
    --delete \
    --cluster-id=1 \
    --account="joe"
```

### ClusterControl RPC API

- Drop a Redis user 'mycache' for cluster ID 1, authenticated as ClusterControl user `myrpcadmin` that has MANAGE permission level:

```
curl -k -XPOST \
-d '{
    "operation": "deleteAccount",
    "cluster_id": 1,
    "account":
        {
       "user_name": "mycache"
        },
    "authenticate":
        {
        "username": "myrpcadmin",
        "password": "mySuperStr0ngP@ssw0rd999"
        }
    }' \
https://localhost:9501/v2/clusters
```

In a testing environment, you might need to perform a shutdown on ClusterControl and monitored database hosts in a graceful way. Depending on the clustering technology, the order of startup and shutting down is vital to keep the whole cluster in sync and ensure a smooth start-up operation in the future.

It's recommended to let the ClusterControl node to be the last one to shut down since it needs to oversee the state of the monitored hosts and saves it into the CMON database. When starting up the database cluster at the later stage, ClusterControl will perform a proper start-up procedure based on the last known state of the monitored hosts.

ClusterControl needs to know whether the database cluster that you are shutting down was shutdown outside of the ClusterControl domain. The following subsections provide details on the order of steps to perform in order to perform a graceful database cluster/server shutdown using ClusterControl.

## 2.51 MySQL/MariaDB Replication

Shutting down:

1. Shutdown the application manually. This usually outside of the ClusterControl domain.
2. Shutdown the Keepalived (if exists) by doing system shut down or through ClusterControl UI.
3. Shutdown the load balancer service (if exists) by doing system shut down or through ClusterControl UI.
4. Shutdown the replicas by using ***ClusterControl → Nodes → choose the server → Shutdown Node → Execute***.
5. Shutdown the primary by using ***ClusterControl → Nodes → choose the server → Shutdown Node → Execute***.
6. Shutdown ClusterControl host by doing system shut down.

Starting up:

1. Power on the ClusterControl host. Ensure you are able to connect to the UI and see the last state of the database cluster.
2. Power on the primary. If auto recover is turned on, this primary will be started automatically.
3. Once the primary is started, power on the remaining replicas.
4. Disable read-only on the primary by using ***ClusterControl → Nodes → choose the primary → Disable Read-Only → Execute***.
5. Power on the load balancer service (if exists).
6. Start the application manually. This usually outside of the ClusterControl domain.

## 2.52 MySQL Galera

Shutting down:

1. Shutdown the application manually. This usually outside of the ClusterControl domain.
2. Shutdown the Keepalived (if exists) by doing system shut down or through ClusterControl UI.
3. Shutdown the load balancer service (if exists) by doing system shut down or through ClusterControl UI.
4. Shutdown the database cluster by using ***ClusterControl → Cluster Actions → Stop Cluster → Proceed***.
5. Shutdown ClusterControl host by doing system shut down.

Starting up:

1. Power on the ClusterControl host. Ensure you are able to connect to the UI and see the last state of the database cluster.

2. If auto-recovery is turned on, the cluster will be started automatically. Otherwise, go to ***ClusterControl → Cluster Actions → Bootstrap Cluster → Proceed***.
3. If auto-recovery is turned on, the load balancer and virtual IP service will be started automatically. Otherwise, start the load balancer service and virtual IP service accordingly.
4. Start the application manually. This usually outside of the ClusterControl domain.

## 2.53 MySQL Cluster (NDB)

Shutting down:

1. Shutdown the application manually. This usually outside of the ClusterControl domain.
2. Shutdown the Keepalived (if exists) by doing system shut down or through ClusterControl UI.
3. Shutdown the load balancer service (if exists) by doing system shut down or through ClusterControl UI.
4. Shutdown the MySQL API servers by using ***ClusterControl → Nodes → pick the server → Shutdown Node → Execute***.
5. Shutdown the Data (NDB) servers by using ***ClusterControl → Nodes → pick the server → Shutdown Node → Execute***.
6. Shutdown the MySQL Cluster management servers by using ***ClusterControl → Nodes → pick the server → Shutdown Node → Execute***.
7. Shutdown ClusterControl host by doing system shut down.

Starting up:

1. Start ClusterControl host. Ensure you are able to connect to the UI and see the last state of the database cluster.
2. If auto recover is turned on, the cluster will be started automatically. Otherwise, start the nodes in this order - MySQL management, MySQL data, MySQL API.
3. Start the load balancer service (if exists).
4. Start the virtual IP service (if exists).
5. Start the application manually. This usually outside of the ClusterControl domain.

## 2.54 MongoDB ReplicaSet

Shutting down:

1. Shutdown the application manually. This usually outside of the ClusterControl domain.
2. Shutdown the secondaries by using ***ClusterControl → Nodes → pick the server → Shutdown Node → Execute***.
3. Shutdown the primary by using ***ClusterControl → Nodes → pick the server → Shutdown Node → Execute***.
4. Shutdown ClusterControl host by doing a system shut down.

Starting up:

1. Start ClusterControl host. Ensure you are able to connect to the UI and see the last state of the database cluster.
2. Start the primary. If auto recover is turned on, this primary will be started automatically.
3. Once the primary is started, start the remaining secondaries.
4. Start the application manually. This usually outside of the ClusterControl domain.

## 2.55 PostgreSQL/TimeScaleDB Replication

Shutting down:

1. Shutdown the application manually. This usually outside of the ClusterControl domain.
2. Shutdown the slaves by using ***ClusterControl → Nodes → pick the server → Shutdown Node → Execute***.
3. Shutdown the master by using ***ClusterControl → Nodes → pick the server → Shutdown Node → Execute***.
4. Shutdown ClusterControl host by doing system shut down.

Starting up:

1. Start ClusterControl host. Ensure you are able to connect to the UI and see the last state of the database cluster.
2. Start the master. If auto recover is turned on, this master will be started automatically.
3. Once the master is started, start the remaining slaves.
4. Start the application manually. This usually outside of the ClusterControl domain.

**Note**

Note If the database server was being shut down gracefully outside of ClusterControl knowledge (through command line init script, systemd or `kill -15`), ClusterControl would still attempt to recover the database node if Node AutoRecovery is turned on. Unless the node is marked as 'Under Maintenance'.

## 2.56 Maintenance

ClusterControl offers a maintenance mode for individual nodes or entire clusters. This feature suppresses alarms and notifications for a set or scheduled duration. Users can also document the reason for the maintenance, such as RAM upgrades or OS patching, for auditing. While in maintenance mode, ClusterControl will not automatically alter the node's state unless manual actions are taken.

**Warning**

Attention

If automatic cluster/node recovery is enabled, ClusterControl will always recover a node/cluster regardless of the maintenance mode status. Do not forget to disable cluster/node recovery to avoid ClusterControl interfering with your maintenance tasks.

### 2.56.1 Type of maintenance mode

ClusterControl has a built-in logic to handle two type of maintenance mode:

- Cluster-wide maintenance
- Individual node maintenance

#### Cluster-wide

Cluster-wide maintenance mode lets you schedule a maintenance window for the entire cluster rather than on a per-node basis. This is suitable if you want to perform maintenance on all nodes in the cluster in the same maintenance window. When active:

- Alarms and notifications are suppressed; ClusterControl will not generate alerts (e.g., threshold breaches, failover notifications) for cluster events during the maintenance period.
- All nodes are marked "under maintenance". Every node regardless of role (primary, replica, proxy, etc.) is shown with the maintenance badge in ClusterControl GUI, and a global banner flags the upcoming or active window.
- Automatic recovery can be disabled (optional). By default recovery/failover remains enabled, but you can turn it off to avoid "false" repairs during planned work.

#### Individual node

Schedule a maintenance window for a specific cluster node when targeted maintenance is required. While active:

- Alarms and notifications are suppressed; ClusterControl will not generate alerts (e.g., threshold breaches, failover notifications) for node events during the maintenance period.
- Only the selected node is marked "under maintenance" and will display a maintenance badge in the ClusterControl GUI.
- Automatic recovery can be disabled (optional). By default recovery/failover remains enabled, but you can turn it off to avoid "false" repairs during planned work.

### 2.56.2 Configuring maintenance mode

Maintenance mode can be configured from ClusterControl GUI and also using ClusterControl CLI.

#### ClusterControl GUI

**Cluster-wide maintenance mode**

1. Log in to your ***ClusterControl GUI → choose a database cluster → Actions → Schedule maintenance***.
2. Choose a database cluster.
3. Go to ***Actions → Schedule maintenance***. This will bring up a panel where you can activate a cluster-wide maintenance mode.

**Node maintenance mode**

1. Log in to your ***ClusterControl GUI → Clusters***.
2. Choose a database cluster.

3. Go to ***Nodes → select the node → Actions → Schedule maintenance***. This will bring up a panel where you can activate maintenance mode for the corresponding node.

Alarms and notifications will be reactivated once the maintenance period is over, or the operator explicitly disables it by going to ***Nodes → select the node → Actions → Disable Maintenance Mode*** or **_choose a database cluster → Actions → Disable Maintenance Mode_**.

#### ClusterControl CLI

- Create a maintenance period for PostgreSQL node 10.35.112.21, starting at 05:44:55 AM for one full day (cmon expects UTC time to create a maintenance):

```
s9s maintenance \
     --create \
     --nodes=10.35.112.21:5432 \
     --begin=2024-05-15T05:44:55.000Z \
     --end=2024-05-16T05:44:55.000Z \
     --reason='Upgrading RAM' \
     --batch
```

- Create a new maintenance period for 192.168.1.121 which shall start tomorrow and finish an hour later:

```
s9s maintenance --create \
       --nodes=192.168.1.121 \
       --begin="$(date --date='now + 1 day' --utc '+%Y-%m-%d %H:%M:%S')
↪'" \
       --end="$(date --date='now + 1day + 1 hour' --utc '+%Y-%m-%d
↪%H:%M:%S')" \
       --reason="Upgrading software."
```

### 2.56.3 Checking maintenance status

Check the maintenance status allowing you to quickly determine if maintenance mode is active. Alarms and notifications will be deactivated if the maintenance mode is active.

#### ClusterControl GUI

**Cluster-wide maintenance mode**

1. Log in to your ***ClusterControl GUI → Clusters***.
2. Choose a database cluster.
3. If you see a blue wrench icon next to the cluster status, it means the maintenance mode is active for the cluster. Roll over on the wrench icon will show a pop-up card with more details on the maintenance like status, end time, who iniatiated the maintenance and the reason.

**Node maintenance mode**

1. Log in to your ***ClusterControl GUI → Nodes***.
2. Choose a node from the list.

3. If you see a blue wrench icon next to the node status, it means the maintenance mode is active for the node. Roll over on the wrench icon will show a pop-up card with more details on the maintenance like status, end time, who iniatiated the maintenance and the reason.

#### ClusterControl CLI

- List out all clusters or nodes that are under maintenance period:

```
s9s maintenance --list --long
```

### 2.56.4 Deactivating maintenance mode

Maintenance mode automatically ends after the set end time. You can also manually deactivate it earlier once maintenance is finished and the server is ready. Alarms and notifications will be reactivated once the maintenance period is over.

To deactivate the maintenance mode:

#### ClusterControl GUI

**Cluster-wide maintenance mode**

1. Log in to your ***ClusterControl GUI → Clusters***.
2. Choose the database cluster.
3. Go to ***Actions → Maintenance → Stop current maintenance***. Confirm the action in the confirmation box to proceed.

**Node maintenance mode**

1. Log in to your ***ClusterControl GUI → Clusters***.
2. Choose the database cluster.
3. Go to ***Nodes → select the node → Actions → Maintenance → Stop current maintenance***. Confirm the action in the confirmation box to proceed.

#### ClusterControl CLI

- Find the item UUID by listing out all nodes that are under maintenance:

```
s9s maintenance --list --long
```

- Delete a maintenance period for UUID 70346c3:

```
s9s maintenance --delete --uuid=70346c3
```

## 2.57 Certificate Management

An SSL/TLS certificate is a small file that proves the identity of a server (and sometimes a client) and enables encrypted connections between them. In ClusterControl, a **Certificate Authority (CA)** is a self-signed "master" certificate that can be used to sign other certificates, called **server** and **client** certificates. Once signed by the same CA, the server and its clients trust one another and can establish an encrypted connection.

Certificate Management is ClusterControl's central repository for generating, importing, and organizing these keys and certificates. Certificates created or imported here can be assigned to any database cluster managed by ClusterControl to enable client-server SSL/TLS encryption, without needing to run any manual OpenSSL commands. You can access it by going to ***ClusterControl GUI → Settings → Certificate management***.

By default, generated and imported keys and certificates are stored under `/var/lib/cmon/ca` on the ClusterControl Controller host.

**Info**

Certificate Management only creates and stores keys/certificates. To actually turn on encryption for a cluster's database connections using a certificate from this repository, see **Encryption settings** under *Cluster Management → Cluster Actions*.

---

### 2.57.1 Best practices and management notes

- **Plan your Common Name (CN) values carefully.** The CN used for a server or client certificate must be different from the CN used for its issuing CA certificate. If they are the same, the certificate and key files will not work with servers compiled using OpenSSL.
- **Use a key length of 2048 bits or higher.** The longer the key, the harder it is to crack, at a small performance cost when establishing new connections.
- **Track expiration dates.** An expired certificate can silently break replication or client connections. Set a realistic **Expiration Date (days)** and calendar a reminder to renew or reissue before it lapses.
- **Prefer Revoke over Delete.** Revoking a certificate marks it invalid (and can be published in a Certificate Revocation List) while keeping a record of it. Deleting removes the certificate and key files permanently and cannot be undone. Only delete when you are certain the certificate is no longer needed anywhere.
- **Regenerate the CRL after revoking.** If you rely on Certificate Revocation Lists to reject revoked certificates, regenerate the issuing CA's CRL right after a revocation so the change takes effect.
- **Back up the CA repository.** The `/var/lib/cmon/ca` directory holds every key and certificate ClusterControl manages. Back it up along with your regular ClusterControl controller backups so you can recover encrypted clusters after a disaster.
- **Duplicate certificates are not recreated.** Importing or generating a certificate with information identical to an existing one will not create a second copy.
- **No dedicated CLI command exists yet.** Certificate generation, import, revocation, and deletion are managed through the ClusterControl GUI or the RPC API. The `s9s` CLI can only enable/disable SSL on an existing cluster using certificates that already exist in the repository.

---

### 2.57.2 Access Certificate Management

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI***.
2. Click **Settings** in the left sidebar.

3. Click **Certificate management**. This opens a panel listing all folders and certificates currently stored in the repository.

### ClusterControl CLI

The `s9s` CLI does not have a dedicated command to browse or manage the certificate repository. It can, however, use certificates that already exist in the repository to enable SSL on a cluster, or supply your own certificate files when registering an existing cluster:

| Option | Description |
|---|---|
| `--enable-ssl` | Enables SSL connections on the cluster's nodes. |
| `--disable-ssl` | Disables SSL connections on the cluster's nodes. |
| `--with-ssl` | Sets up SSL while creating a new cluster. |
| `--ssl-ca=PATH` | Path (on the controller) to an existing SSL CA file to use. |
| `--ssl-cert=PATH` | Path (on the controller) to an existing SSL certificate file to use. |
| `--ssl-key=PATH` | Path (on the controller) to an existing SSL key file to use. |
| `--ssl-pass=PASSWD` | Password for an existing CA private key, if any, when registering a cluster. |

Enable SSL on an existing MySQL/MariaDB cluster (ID 36) using certificates already stored in the repository:

```
s9s cluster \
    --enable-ssl \
    --cluster-id=36 \
    --wait
```

### ClusterControl RPC API

Certificate operations are available under the `/v2/ca` path. All calls require a TLS connection and, in most cases, an authenticated session or inline credentials. See *ClusterControl RPC API → Authenticating with the API* for details.

| Operation | Description |
|---|---|
| `listCerts` | List CA, server, and client certificates. |
| `certInfo` | Get details about a specific certificate. |
| `create` | Create a signed (self-signed or CA-signed) certificate. |
| `revoke` | Mark a certificate as revoked. |
| `delete` | Delete a certificate and its private key files. |
| `move` | Move/rename a certificate within the CA storage. |
| `crl` | Create a Certificate Revocation List (CRL) signed by a specified CA. |
| `importlocal` | Import a certificate already present on the controller host. |

For the full request/response reference of every operation, see *Reference Manuals → API/SDK → ClusterControl RPC API → Certificate Authority (CA)*.

### 2.57.3 View the certificate list

The main panel shows every folder and certificate in the repository as a browsable tree, together with a toolbar for common actions.

| Field/Butto | Description |
|---|---|
| **Create Folder** | Creates a new folder under the currently selected directory, useful for grouping certificates (for example, one folder per cluster). |
| **Refresh** | Reloads the certificate and folder list from the repository. |
| **More…** | Opens the menu to **Generate Client/Server Certificate**, **Generate Self-signed Certificate**, or **Import certificate**. |

Clicking on any certificate in the tree opens its details, including its status (Issued/Revoked), validity period, key size, and which hosts or clusters currently use it.

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Certificate management***.
2. Click on a folder to expand it, or click **Create Folder** to add a new one.
3. Click on any certificate to view its details in the side panel.
4. Click **Refresh** at any time to make sure you are looking at the latest state of the repository.

#### ClusterControl CLI

The `s9s` CLI has no dedicated command to browse the certificate repository. Use the **ClusterControl GUI** or the **RPC API** tab instead.

#### ClusterControl RPC API

List every CA, server, and client certificate in the repository:

```
curl -k -XPOST \
-d '{ "operation": "listCerts" }' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

Get the details of a specific certificate by its `id` (as returned by `listCerts`):

```
curl -k -XPOST \
-d '{
    "operation": "certInfo",
    "id": 2
    }' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

### 2.57.4 Generate a self-signed Certificate Authority (CA)

A self-signed CA is the "root of trust" you use to sign other certificates. Create one first if you don't already have a CA to work with.

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Certificate management***.
2. Click **More… → Generate Self-signed Certificate**.
3. Fill in the certificate details:

| Field | Description |
|---|---|
| **Path** | Certificate repository path. Click the file browser on the left side to change it. Default is `/var/lib/cmon/ca`. |
| **Certificate Authority and Key Name** | A name without an extension, for example `MyCA` or `ca-cert`. |
| **Description** | A short description for the certificate authority. |
| **Country** | Country name, chosen from the dropdown. |
| **State** | State or province name. |
| **Locality** | City name. |
| **Organization** | Organization name. |
| **Organization unit** | Unit or department name. |
| **Common name** | The server's fully qualified domain name (FQDN) or your own name. This value must be different from the Common Name used for any server/client certificate this CA will sign. |
| **Email** | Contact email address. |
| **Key length (bits)** | Key size in bits. Use 2048 or higher. |
| **Expiration Date (days)** | Number of days the certificate stays valid. |

4. Click **Generate** to create the CA certificate and key, or **Reset** to clear the form and start over.

#### ClusterControl CLI

The `s9s` CLI has no dedicated command to generate a Certificate Authority. Use the **ClusterControl GUI** or the **RPC API** tab instead.

#### ClusterControl RPC API

Create a self-signed CA certificate named `exampleCa1`, valid for 3650 days (10 years):

```
curl -k -XPOST \
-d '{
   "operation":"create",
   "type":"ca",
   "name":"exampleCa1",
   "validity":3650,
```

(continues on next page)

(continued from previous page)

```
    "data":{
      "keybits":4096,
      "CN":"My CA certificate.",
      "description":"This is an example CA certificate.",
      "emailAddress":"john.doe@severalnines.com"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

!!! info

```
`type`, `name`, and `validity` are required for every `create` call. Set␣
↪`type` to `ca` to generate a Certificate Authority, or to the appropriate␣
↪type for a server or client certificate, as described in [Generate a␣
↪client or server certificate](#generate-a-client-or-server-certificate).
```

### 2.57.5 Generate a client or server certificate

Once you have a CA (or if you only need a quick self-signed certificate), you can generate the server certificate that a database will present, and the client certificate used by applications or replicas to connect.

**Warning**

Attention

The Common Name (CN) used for the server and client certificates must each be different from the Common Name used for the CA certificate. Using the same CN causes the certificate and key files to stop working on servers compiled using OpenSSL.

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Certificate management***.
2. Click **More… → Generate Client/Server Certificate**.
3. Fill in the certificate details:

| Field | Description |
|---|---|
| **Certificate Authority** | Select an existing CA from the left-hand side menu to sign this certificate, or leave it empty to generate a self-signed certificate instead. |
| **Type** | **Server** – for use by the database server. **Client** – for use by an application or replicating node. |
| **Certificate and Key Name** | The name ClusterControl uses to generate the certificate and key files. For example, entering `Severalnines` generates `severalnines.crt` and `severalnines.key`. |
| **Description** | A short description for the certificate. |
| **Country** | Country name, chosen from the dropdown. |
| **State** | State or province name. |
| **Locality** | City name. |
| **Organization** | Organization name. |
| **Organization unit** | Unit or department name. |
| **Common name** | The server's FQDN or your own name. Must differ from the CA's Common Name. |
| **Email** | Contact email address. |
| **Key length (bits)** | Key size in bits. Use 2048 or higher. |
| **Expiration Date (days)** | Number of days the certificate stays valid. |

4. Click **Generate** to create the certificate and key, or **Reset** to clear the form.

#### ClusterControl CLI

The `s9s` CLI has no dedicated command to generate a client or server certificate. Use the **ClusterControl GUI** or the **RPC API** tab instead.

#### ClusterControl RPC API

The `create` operation accepts `type: "server"` and `type: "client"` in addition to `type: "ca"`. To sign a server or client certificate with an existing CA, use the **ClusterControl GUI** wizard instead (see the GUI tab above), which handles CA selection for you. See *Reference Manuals → API/SDK → ClusterControl RPC API → create* for the full parameter reference.

---

### 2.57.6 Import an existing certificate

If you already have certificates issued by an external or corporate CA, import them into the repository so ClusterControl can use them the same way as a generated one.

Before importing, make sure to:

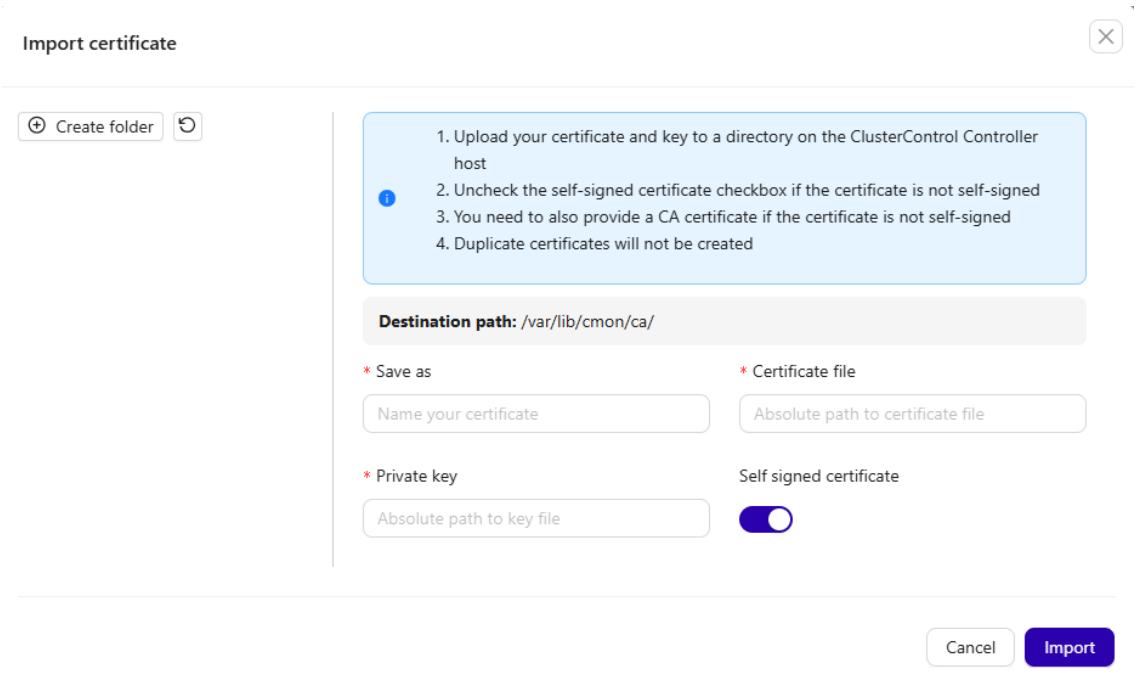
1. Upload your certificate and key files to a directory on the ClusterControl Controller host.
2. Uncheck **Self-signed Certificate** if the certificate is not self-signed.
3. Also provide the CA certificate if the imported certificate is not self-signed.

**Note**

Note Duplicate certificates will not be created. Importing the same certificate twice has no additional effect.

### ClusterControl GUI

1. Log in to your *ClusterControl GUI → Settings → Certificate management*.
2. Click **More… → Import certificate**.




3. Fill in the import details:

| Field | Description |
|---|---|
| **Destination Path** | Where the certificate will be imported to. Click the file explorer on the left to change the path. |
| **Save As** | Name to store the imported certificate under. |
| **Certificate File** | Absolute path to the certificate file, for example `/home/user/ssl/file.crt`. |
| **Private Key File** | Absolute path to the key file, for example `/home/user/ssl/file.key`. |
| **Self-signed Certificate** | Uncheck this if the certificate is not self-signed. |

4. Click **Import** to start the import process.

### ClusterControl CLI

The `s9s` CLI has no dedicated command to import a certificate into the repository. Use the **ClusterControl GUI** or the **RPC API** tab instead.

#### ClusterControl RPC API

Import a client certificate, its key, and its issuing CA certificate from files already present on the controller host:

```
curl -k -XPOST \
-d '{
   "operation":"importlocal",
   "cert_file": "/tmp/TestCLIENT.crt",
   "key_file": "/tmp/TestCLIENT.key",
   "ca_file": "/tmp/TestingCA007.crt",
   "name": "testing/import/testClient",
   "name_ca": "testing/import/testCA"
}' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

`ca_file`/`name_ca` are optional and only needed when the certificate being imported is not self-signed and not already known to ClusterControl.

---

### 2.57.7 Move or rename a certificate

Use this to reorganize the certificate repository, for example moving a certificate into a per-cluster folder after generating it in the root directory.

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Certificate management***.
2. Select the certificate you want to reorganize.
3. Drag it into the destination folder, or use the certificate's **Actions** menu to move it, then confirm the new path.

#### ClusterControl CLI

The `s9s` CLI has no dedicated command to move or rename a certificate. Use the **ClusterControl GUI** or the **RPC API** tab instead.

#### ClusterControl RPC API

Move (and effectively rename) certificate ID 228 into a new folder path:

```
curl -k -XPOST \
-d '{
   "operation":"move",
   "id": 228,
   "name": "examples/demoCA/cert01"
}' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

---

### 2.57.8 Revoke a certificate

Revoking marks a certificate as no longer trusted without deleting its files, which keeps a record for audit purposes. This is the recommended way to retire a certificate.

#### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Certificate management***.
2. Select the certificate you want to revoke.
3. Open its **Actions** menu and click **Revoke**.
4. Confirm the action.

#### ClusterControl CLI

The `s9s` CLI has no dedicated command to revoke a certificate. Use the **ClusterControl GUI** or the **RPC API** tab instead.

#### ClusterControl RPC API

Revoke certificate ID 227:

```
curl -k -XPOST \
-d '{
   "operation":"revoke",
   "id": 227
}' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

**Note**

Note Regenerate the issuing CA's Certificate Revocation List (CRL) after revoking so client applications that check the CRL immediately stop trusting the revoked certificate. Pass the **issuing CA's** `id` to the `crl` operation, not the revoked certificate's own `id`. A CRL can only be generated from a CA certificate that is still valid (not itself revoked). See *Generate a Certificate Revocation List (CRL)*.

---

### 2.57.9 Generate a Certificate Revocation List (CRL)

A Certificate Revocation List (CRL) is a signed list of certificates a given CA has revoked. Distributing an up-to-date CRL lets database servers and clients reject connections that present a revoked certificate, even if that certificate hasn't expired yet.

#### ClusterControl CLI

The `s9s` CLI has no dedicated command to generate a Certificate Revocation List. Use the **RPC API** tab instead.

### ClusterControl RPC API

Generate a CRL signed by CA certificate ID 229:

```
curl -k -XPOST \
-d '{
   "operation":"crl",
   "id": 229
}' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

The response includes the path to the generated `.crl` file inside the certificate repository, which you can then distribute to the hosts that need to validate it.

**Warning**

Attention

The `id` must belong to a CA certificate that is still valid (status `Issued`). If the CA itself has already been revoked, this call fails with `No valid certificate found with id <id>`.

---

## 2.57.10 Delete a certificate

!!! danger “Warning”

```
Deleting removes the certificate and private key files permanently. This is␣
↪a destructive, unusual operation. Use [Revoke](#revoke-a-certificate)␣
↪instead whenever possible.
```

### ClusterControl GUI

1. Log in to your ***ClusterControl GUI → Settings → Certificate management***.
2. Select the certificate you want to remove.
3. Open its **Actions** menu and click **Delete**.
4. Confirm the action.

### ClusterControl CLI

The `s9s` CLI has no dedicated command to delete a certificate. Use the **ClusterControl GUI** or the **RPC API** tab instead.

### ClusterControl RPC API

Delete certificate ID 1001:

```
curl -k -XPOST \
-d '{
   "operation":"delete",
```

(continues on next page)

(continued from previous page)

```
    "id": 1001
}' \
-b cookies.jar \
https://localhost:9501/v2/ca
```

## 2.58 Backup and Restore

This section focuses on essential strategies and tools for backing up and restoring your database clusters with ClusterControl. Learn how to ensure data safety and enable quick recovery from potential data loss. You can also scale your cluster with a backup.

**See also**

- *Backup & Restore Overview →*
- *Create Backup →*
- *Schedule Backup →*
- *Restore Backup →*
- *Restore External Backup →*
- *Backup Verification →*
- *Create Cluster from Backup →*
- *Upload Backup to Cloud →*
- *Download Backup from Cloud →*

## 2.59 Overview

ClusterControl provides comprehensive backup and restore capabilities to ensure the continued availability and integrity of your database systems. This overview outlines how ClusterControl simplifies the process of creating reliable backups, managing your backup strategy, and performing efficient data restoration for various database technologies.

### 2.59.1 What you can do

ClusterControl's backup and restore features fall into four broad areas, across its supported database engines:

**Backup creation and scheduling**

- Physical or logical backups, on demand or on a schedule, from any database node.
- Full, incremental, or partial (specific schema/table) backups.
- Backup encryption, and compression with an adjustable compression level.
- Automatic backup host selection based on role (replica or primary), with automatic failover to another node if the selected one is unavailable.

- Backup throttling to limit the impact on the source node.
- Configurable backup file naming and subdirectory format.
- Incremental and differential backup chaining, grouped by backup type.

**Storage and retention**

- Store backups locally, on the controller, or in cloud object storage (public or any S3-compatible private provider).
- Stream backups directly to cloud storage, skipping local storage entirely, for supported methods.
- Upload or download existing backups to and from the cloud.
- Rotation and retention policies for local and cloud backups, with automatic purging of expired backups.

**Reliability and recovery**

- Automatic backup verification.
- Point-in-time recovery for MySQL, MariaDB, PostgreSQL, MongoDB (via Percona Backup for MongoDB), and SQL Server.
- Restore to a temporary database server outside the cluster.
- Create a new cluster from an existing backup.
- Restore a backup taken outside ClusterControl (MySQL Replication and Galera clusters only). See *Restore External Backup*.
- Backup history.

**Operations and management**

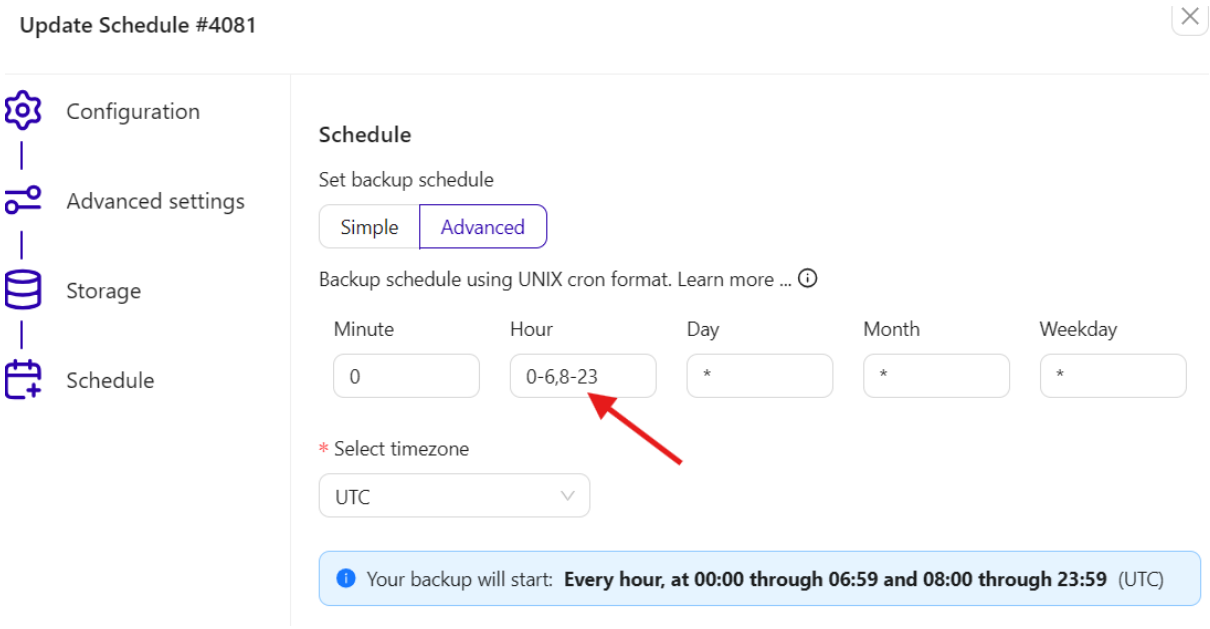
- Centralized backup management for all your database clusters, from a unified ClusterControl GUI dashboard or the ClusterControl CLI.
- Email, PagerDuty, Slack, or custom webhook alerts on backup-related events.
- Scheduled or on-demand operational reports on backup status.

### 2.59.2 Backup jobs

The backup process performed by ClusterControl is running as a background thread (RUNNING3) which doesn't block any other non-backup jobs in the queue. If the backup job takes hours to complete, other non-backup jobs can still run simultaneously via the main thread (RUNNING). You can see the job progress at ***ClusterControl GUI → Activity center → Jobs***.

Note that ClusterControl executes **a single backup job per cluster** at any given time. If multiple scheduled backups for a cluster overlap, the subsequent schedules will not start until the active backup job is finished. For instance, if a full daily backup is set for 7:00 AM, and incremental backups are scheduled hourly, the

7:00 AM incremental backup will likely wait for the full backup to complete before it begins. You may avoid this by excluding certain hours in the schedule by using the scheduler advanced editor, which follow the UNIX cron format, as shown in the following example:




### 2.59.3 Backup encryption and decryption

If the encryption option is enabled for a particular backup, ClusterControl will use OpenSSL to encrypt the backup using the AES-256 CBC algorithm. Encryption happens on the backup node. If you choose to store the backup on the controller node, the backup files are streamed over in encrypted format through `socat` or `netcat`.

If compression is enabled, the backup is first compressed and then encrypted resulting in smaller backup sizes. The encryption key will be generated automatically (if not exists) and stored inside the CMON configuration for the particular cluster under `backup_encryption_key` the option. This key is stored with base64 encoded and should be decoded first before using it as an argument to pass when decrypting the backup. The following command shows how to decode the key:

```
cat /etc/cmon.d/cmon_X.cnf | grep ^backup_encryption_key | cut -d"'" -f2 |␣
↪base64 -d > keyfile.key
```

Where X is the cluster ID. The above command will read the `backup_encryption_key` value and decode the value to a binary output. Thus, it is important to redirect the output to a file, as in the example, we redirected the output to `keyfile.key`. The key file which stores the actual encryption key can be used in the OpenSSL command to decrypt the backup, for example:

```
$ cat {BACKUPFILE}.aes256 | openssl enc -d -aes-256-cbc -pass␣
↪file:/path/to/keyfile.key > backup_file.xbstream.gz
```

Or, you can pass the stdin to the respective restore command chain, for example:

```
$ cat {BACKUPFILE}.aes256 | openssl enc -d -aes-256-cbc -pass␣
↪file:/path/to/keyfile.key | gzip -dc | xbstream -x -C /var/lib/mysql
```

### 2.59.4 Backup retention

Backup retention can be configured at ***ClusterControl GUI → {cluster} → Backups → … → Backup Settings → Default backups retention period*** (default is 31 days) to the number of days ClusterControl keeps the existing backups. Backups older than the value defined will be deleted. You can also customize the retention period per backup (default, custom or keep forever) under the **Backup Retention** field when creating or scheduling a backup.

The purging is based on the following conditions:

1. When a new backup is successfully created, and if no verified backup is requested, the older backups will be checked and removed.
2. When the verified backup is successfully created, the older backups will be checked and removed.
3. The backup housekeeping job remains executed every 24 hours. Thus, if no backups are created and no backups are verified, the backup retention still will be done every 24 hours

For cloud-storage backups, they follow the **Cloud backup retention period** setting (default is 180 days).

!!! danger

```
Backups exceeding the retention period will be deleted. Once the backups␣
↪are deleted, they can't be recovered.
```

### 2.59.5 Backup storage

Backup created and scheduled in ClusterControl can only be stored in the following location:

- ClusterControl controller host, where the backup file will be streamed over to the ClusterControl host on-the-fly.
- The database host that being backed up.
- Cloud storage, which you have the option to stream directly (physical backup only) or upload to cloud after the backup is successful.

If you have external backup storage host, you can use NFS and share the volume where your backup shall be stored on either of the ClusterControl controller host, or the database host itself. ClusterControl will respect the **Storage directory** as the base directory for all backups, suffixed by the **Backup subdirectory** as explained below.

ClickHouse is the exception: storing the backup on the ClusterControl controller is not supported, since ClickHouse's own native `BACKUP` command writes directly to a disk registered on the database node itself. See *ClickHouse native backup* below.

#### Backup subdirectory

ClusterControl will create the backup directory on the destination host, relative to the **Storage directory** path if doesn't exist. The default value is `BACKUP-%I`, which produces `BACKUP-151` for a backup with ID 151. This string may hold standard `%X` field separators. For example, `BACKUP-%06I` will be replaced by the numerical ID of the backup in 6 field-wide format that uses '0' as leading fill characters (e.g, `BACKUP-000151`).

The following table shows the supported variables:

| Variable | Description |
|---|---|
| B | The date and time when the backup creation was beginning. |
| H | The name of the backup host, the host that created the backup. |
| i | The numerical ID of the cluster. |
| I | The numerical ID of the backup. |
| J | The numerical ID of the job that created the backup. |
| M | The backup method (e.g. "mysqldump"). |
| O | The name of the user who initiated the backup job. |
| S | The name of the storage host, the host that stores the backup files. |
| % | The percent sign itself. Use two percent signs, %% the same way the standard printf() function interprets it as one percent sign. |

You can combine multiple variables and field-width specifiers in a single format string to organize backups by cluster, method, and host while keeping the numeric ID sortable. For example:

```
c%i-%M-%H-job%J-BACKUP-%06I
```

For a `xtrabackupfull` backup with ID `151`, taken by job `8831` on cluster `42` from host `192.168.40.60`, this resolves to:

```
c42-xtrabackupfull-192.168.40.60-job8831-BACKUP-000151
```

The `%06I` zero-pads the backup ID to 6 digits, so directories continue to sort correctly by ID even once you have more than 999 backups, which the plain `%I` in the default format does not guarantee.

### 2.59.6 Backup settings

The default backup settings can be configured under ***ClusterControl GUI → {cluster} → Backups → ... → Backup Settings***. These settings are bounded per database cluster. In this section, you can configure the following settings:

| Setting | Description |
|---|---|
| Default directory | The default path for the backup directory. ClusterControl will create the backup directory on the destination host if doesn't exist. |
| Default subdirectory | The default subdirectory naming format. See *Backup subdirectory*. |
| Netcat port | Specify the port number or port number range that will be used by ClusterControl to stream backup created on the database node. This port must be opened on both source and destination hosts. Only used if you choose **Store on Controller** in the **Storage Location** configuration.Default is "9999,9990-9998", which means port 9999 will be preferred if available. |
| Enable hash check on created backup files | Toggle **On** (default) to have ClusterControl perform an MD5 checksum on the created backup. For large backups, this may consume significant resources during the process. If toggled **Off**, the MD5 checksum verification is disabled. Note that there is no built-in way to verify the integrity of backup files when they are streamed over the network. |
| Default backups retention period | The number of days ClusterControl retains existing backups. Backups older than the defined value will be automatically deleted. You can also customize the retention period per backup (default, custom, or keep forever) under **Backup Retention** when creating or scheduling a backup. See *Backup retention* for how and when purging runs. |
| Default cloud backups retention period | The number of days ClusterControl retains uploaded backups in the cloud. Backups older than the defined value will be automatically deleted. |

### 2.59.7 Supported backup methods

ClusterControl offers support for various backup tools and methods, tailored to the specific database clusters in use. The table below details these options:

| Cluster type | Supported backup methods |
|---|---|
| MySQL-based clusters | mysqldumpPercona Xtrabackup (full, incremental)Binlog backup |
| MariaDB-based clusters | mariadb-dumpMariaDB Backup (full, incremental)Binlog backup |
| PostgreSQL-based clusters | pg_dumppg_dumpallpg_basebackup (full, incremental - only for v17 and later)pgBackRest (full, incremental, differential) |
| MongoDB-based clusters | mongodumpPercona Backup for MongoDB |
| Redis-based clusters | RDB (includes AOF file if AOF persistence is enabled) |
| SQL Server | FullDifferentialTransaction log |
| Elasticsearch | Elasticsearch snapshot |
| ClickHouse-based clusters | ClickHouse native backup (full, incremental) |

**Note**

Note Binlog backup is only available for scheduled backups, not for on-demand backups. See *Schedule Backup → Supported backup methods*.

Each database backup method is explained in the following sections.

### mysqldump

ClusterControl performs mysqldump against all or selected databases by using the `--single-transaction` option. It automatically performs mysqldump with `--master-data=2` if it detects binary logging is enabled on the particular node to generate a binary log file and position statement in the dump file. ClusterControl generates a set of 4 mysqldump files with the following suffixes:

- `_data.sql.gz` - Schemas' data.
- `_schema.sql.gz` - Schemas' structure.
- `_mysqldb.sql.gz` - MySQL system database.
- `_triggerseventroutines.sql.gz` - MySQL triggers, events, and routines.

### Percona Xtrabackup

Percona Xtrabackup is an open-source MySQL hot backup utility from Percona that can back up data from InnoDB, XtraDB, and MyISAM tables. Xtrabackup does not lock your database during the backup process. For large databases (100+ GB), it provides a much better restoration time as compared to mysqldump. The restoration process involves preparing MySQL data from the backup files before replacing or switching it with the current data directory on the target node.

Since its ability to create full and incremental MySQL backups, ClusterControl manages incremental backups, and groups the combination of full and incremental backups in a backup set. A backup set has an ID based on the latest full backup ID. All incremental backups after a full backup will be part of the same backup set. The backup set can then be restored as one single unit using the **Restore Backup** feature.

**Warning**

Attention

Without a full backup to start from, the incremental backups are useless.

### mariadb-dump

The mariadb-dump is a logical backup utility included with MariaDB Server. It's essentially MariaDB's fork of the original MySQL mysqldump tool and is a replacement to mysqldump since MariaDB 10.4. It supports MariaDB system-versioned tables, sequence, virtual columns and MariaDB-specific privilege grants which are different than the MySQL implementation.

On all supported versions for MariaDB, ClusterControl will default to mariadb-dump as the preferred logical backup method.

### MariaDB Backup

MariaDB Backup is a fork of Percona XtraBackup with added support for compression and data-at-rest encryption available in MariaDB, included in MariaDB 10.1.23 and later. It is an open-source tool provided by MariaDB for performing physical online backups of InnoDB, Aria, and MyISAM tables. MariaDB Backup is available on Linux and Windows.

On all supported versions for MariaDB, ClusterControl will default to MariaDB Backup as the preferred backup method and snapshot state transfer (SST) method.

### Binlog backup

**Warning**

Attention

This backup method is available only when scheduling a MySQL or MariaDB backup (not supported for on-demand backups) and requires a supported object storage destination. Successful binlog backups are not listed in the backup list, as they are designed to work in conjunction with PITR-compatible full backups. To restore, select a PITR-compatible backup and enable **Point-in-Time Recovery (PITR)** to apply the binary logs.

Binlog backup involves backing up MySQL or MariaDB binary logs, which record all data-changing events. These logs are used together with a full backup to enable point-in-time recovery (PITR). Binlog backups are not a replacement for full backups. Instead, they serve as an incremental layer that captures changes occurring after the last full backup.

Binlog backups are particularly useful for:

- Recovering from accidental `DELETE`, `UPDATE`, or `DROP` operations.
- Restoring the database to a specific point in time.
- Minimizing data loss between full backups.

Follow these steps to configure and use binlog backups effectively:

1. Perform or schedule a full PITR-compatible backup method such as:
    - mysqldump (must be PITR-compatible)
    - xtrabackup (full or incremental)
    - mariabackup (full or incremental)
2. Configure a cloud storage credential for binlog backups under ***ClusterControl GUI → Settings → Cloud storage credentials***.
3. Schedule regular binlog backups at an interval aligned with your recovery objectives (RPO/RTO), for example:
    - Every hour
    - Every 12 hours
    - Daily
4. When restoring via ClusterControl, choose a PITR-compatible backup and enable **Point-in-Time Recovery**.

5. ClusterControl will perform the following automatic restoration process:
    - Retrieve the required binary logs from object storage
    - Restore the selected full backup
    - Replay binary logs up to the specified time or position

When performing a binlog backup operation, ClusterControl performs the following actions in particular order:

1. Executes `FLUSH BINARY LOGS` to rotate the current binary log and records the new log file.
2. Uploads all completed binary logs to object storage, excluding the active (latest) log file.
3. Stores metadata about the uploaded logs in the `cmon.backup_binlog` table.

### pg_dump/pg_dumpall

Depending on the database backup target configuration, ClusterControl will automatically choose the suitable method to create the logical backup using either `pg_dump` or `pg_dumpall` backup tool. Commonly, `pg_dump` will be chosen for a partial logical backup, while `pg_dumpall` for a full logical backup.

ClusterControl performs `pg_dumpall` against all databases together with `--clean` the option, which includes SQL commands to clean (drop) databases before recreating them. DROP commands for roles and tablespaces are added as well. The output will be `.sql.gz` extension and the file name contains the timestamp of the backup.

### pg_basebackup

pg_basebackup is used to take base backups of a running PostgreSQL database cluster. These are taken without affecting other clients to the database and can be used both for point-in-time recovery and as the starting point for a log shipping or streaming replication standby server. It makes a binary copy of the database cluster files while making sure the system is put in and out of backup mode automatically. Backups are always taken of the entire database cluster; it is not possible to back up individual databases or database objects.

ClusterControl connects to the replication stream using the replication user (default is `cmon_replication`) with `--wal-method=fetch` option when creating the backup. The output will be `base.tar.gz` inside the backup directory.

Starting with ClusterControl 2.4.0 (June 2026), incremental backups using pg_basebackup are supported. This feature requires the PostgreSQL `summarize_wal` parameter, which is available only in PostgreSQL 17 and later. When enabled, `summarize_wal` starts a background process that continuously analyzes the Write-Ahead Log (WAL) and tracks which database blocks have changed. This information allows pg_basebackup to perform incremental backups by copying only modified blocks, significantly reducing backup time, storage requirements, and network usage.

To enable this feature, run the **Configure WAL** job from the node menu (see *Configuring WAL*). Before creating an incremental backup, ensure that a full `pg_basebackup` backup has been successfully completed with `summarize_wal` enabled, as incremental backups rely on this base backup.

### pgBackRest

pgBackRest is an open-source software developed to perform efficient backup on PostgreSQL databases that measure in tens of terabytes and greater. It supports per-file checksums, compression, partial/failed backup resume, high-performance parallel transfer, asynchronous archiving, tablespaces, expiration, full/differential/incremental, local/remote operation via SSH, hard-linking, restore, and more. The tool

is written in Perl and does not depend on rsync or tar but instead performs its own deltas which give it maximum flexibility.

### Installing pgBackRest

Starting from ClusterControl 1.9.0, pgBackRest can be configured as follows:

1. **Primary**: Install on the current primary node only (not on replicas). The backup repository will be hosted on the primary node. No SSH configuration is required for PgBackRest.
2. **All database nodes**: Install on all database nodes. The backup repository will be hosted on the current primary node, while backups are performed from a standby node. PgBackRest uses SSH for inter-node communication, and key-based authentication is automatically configured by ClusterControl.
3. **All database nodes and a dedicated repository host**: Install PgBackRest on all PostgreSQL nodes. The backup repository will be hosted on a separate, dedicated server (not part of the database cluster). Backups are performed from a standby node. PgBackRest uses SSH for communication, with key-based authentication automatically configured by ClusterControl.

**Note**

Note The pgBackRest backup directory cannot be reused or shared with other backup methods.

During the first attempt of making pgBackRest backup, ClusterControl will re-configure the node to install and configure pgBackRest. Take note that this operation requires a database restart and might introduce downtime to your database. A configuration file will be created at `/etc/pgbackrest.conf` and will be configured according to the version used and location of the PostgreSQL data. The pgBackRest default repository path is located at `/var/lib/pgbackrest`. Additionally, ClusterControl will configure the following lines inside postgresql.conf (which explains why it requires restart during the first run):

```
archive_mode = on
archive_command = 'pgbackrest --stanza=clustercontrol-stanza archive-push %p
↪'
```

**Warning**

Attention

PITR using `pg_basebackup` will no longer be possible if switching the WAL archiving method from “local WAL archiving” to “pgBackRest”.

**Info**

In the ClusterControl GUI, pgBackRest nodes are listed with a fake port number (200000 + cluster ID), due to the internal CmonHost object requirement for a host:port combination. As a matter of fact, the pgBackRest process does not listen, nor need a port number.

### Full backup

Full backup of pgBackRest copies the entire contents of the database cluster to the backup. The first backup of the database cluster is always a Full Backup. pgBackRest is always able to restore a full backup directly. The full backup does not depend on any files outside of the full backup for consistency.

### Differential backup

For differential backup, pgBackRest copies only those database cluster files that have changed since the last full backup. pgBackRest restores a differential backup by copying all of the files in the chosen differential backup and the appropriate unchanged files from the previous full backup. The advantage of a differential backup is that it requires less disk space than a full backup, however, the differential backup and the full backup must both be valid to restore the differential backup.

For example, if a full backup is taken on Sunday and the following daily differential backups are scheduled, the data that is backed up will be:

- Monday - data from Sunday to Monday
- Tuesday - data from Sunday to Tuesday
- Wednesday - data from Sunday to Wednesday
- Thursday - data from Sunday to Thursday

### Incremental backup

For incremental backup, pgBackRest copies only those database cluster files that have changed since the last backup (which can be another incremental backup, a differential backup, or a full backup). As an incremental backup only includes those files changed since the prior backup, they are generally much smaller than full or differential backups. As with the differential backup, the incremental backup depends on other backups to be valid to restore the incremental backup. Since the incremental backup includes only those files since the last backup, all prior incremental backups back to the prior differential, the prior differential backup, and the prior full backup must all be valid to perform a restore of the incremental backup. If no differential backup exists then all prior incremental backups back to the prior full backup, which must exist, and the full backup itself must be valid to restore the incremental backup.

For example, if a full backup is taken on Sunday and the following daily incremental backups are scheduled, the data that is backed up will be:

- Monday - data from Sunday to Monday
- Tuesday - data from Monday to Tuesday
- Wednesday - data from Tuesday to Wednesday
- Thursday - data from Wednesday to Thursday

### Write-Ahead Logging (WAL) Archiving

WAL archiving stores transaction logs continuously, allowing recovery to a specific point in time when combined with a base backup. ClusterControl provides a built-in interface to configure PostgreSQL WAL (Write-Ahead Logging) archiving for Point-in-Time Recovery (PITR) with pg_basebackup. For pgBackRest, WAL archiving is automatically configured when installing the backup tool for the first time.

### Configuring WAL

To configure WAL archiving, go to ***ClusterControl GUI → {cluster} → Backups → … → Configure WAL***. The **Configure WAL** dialog appears, where you can configure the following options:

- **Apply to**: Select which PostgreSQL nodes should receive the WAL archiving configuration. You can configure WAL archiving on all PostgreSQL nodes or on selected individual nodes. In most deployments, WAL archiving is configured on all PostgreSQL nodes.
- **Archive mode**: Controls the PostgreSQL `archive_mode` setting. Choose **Off** to disable WAL archiving. Choose **On** to enable WAL archiving during normal operation. This is the recommended setting for most environments. Choose **Always** to force WAL archiving even during recovery or standby mode.
- **Compress WAL archive**: When enabled, ClusterControl compresses archived WAL files to reduce storage usage. This option is recommended for environments with high transaction volume, long retention periods, or limited archive storage capacity.
- **Summarize WAL**: Toggle **On** to enable the PostgreSQL `summarize_wal` parameter, which starts a background process that tracks which database blocks have changed since the last backup. PostgreSQL 17 or later only. This is required before you can create an incremental `pg_basebackup`; see *pg_basebackup* for details.
- **Custom WAL archive directory**: Specify a custom directory where WAL archive files will be stored. If left empty, ClusterControl uses the default archive location configured for the cluster. Ensure that the PostgreSQL user has permission to write to the selected directory.

Click **Configure** to apply the changes. ClusterControl will:

1. Update the PostgreSQL WAL archiving settings.
2. Configure the required archive commands.
3. Reload or restart PostgreSQL if required.
4. Begin archiving WAL files automatically.

The following is an example configuration loaded into a PostgreSQL instance after the "Configure WAL" job finishes:

```
archive_mode=always
archive_command='test -f /data_volume/postgres/data/../wal_archive/%f.gz ||␣
↪gzip -c < %p > /data_volume/postgres/data/../wal_archive/%f.gz'
archive_cleanup_command='pg_archivecleanup -x .gz /
↪data_volume/postgres/data/../wal_archive %r'
```

The archived WAL files can then be used for Point-in-Time Recovery (PITR). See *Restore Backup*.

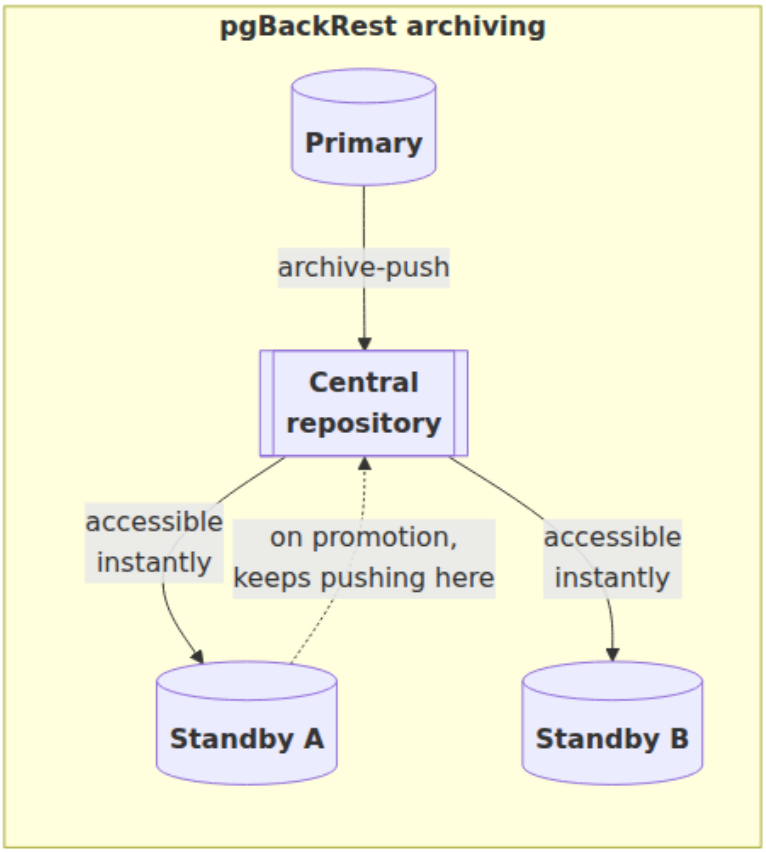
**Note**

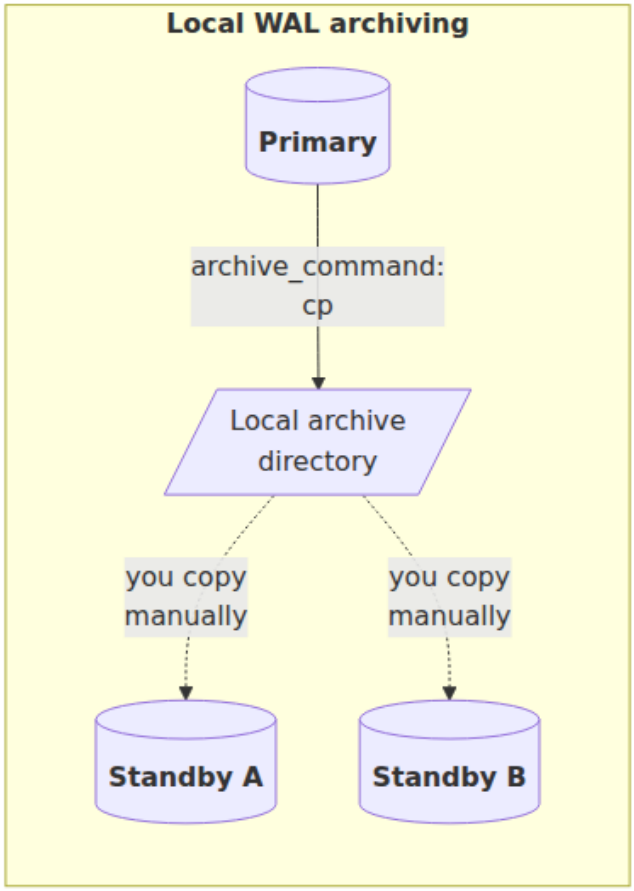
Note * Changing the WAL archive configuration may invalidate previously collected WAL archives and reduce the available PITR recovery window. * WAL archiving should be used together with regular full backups. * Sufficient storage space should be allocated for WAL retention.

### Local archiving vs pgBackRest archiving

The main difference is that local WAL archiving (like using the `cp` command) is a basic, single-threaded copy process, while pgBackRest archiving is a professional-grade, multi-featured pipeline designed for reliability and speed.

If you use simple local archiving, you have to manually ensure those files get copied to all standbys or a shared drive. With pgBackRest, the `archive-push` command automatically ships WALs to a central repository that all your nodes (primary and standbys) can access instantly. This is what makes failover possible. When a standby is promoted, it just starts "pushing" to the same central location the old master was using, keeping your backup stream unbroken.


pgBackRest archiving
Primary
archive-push
Central repository
accessible instantly
on promotion, keeps pushing here
accessible instantly
Standby A
Standby B


Local WAL archiving
Primary
archive_command: cp
Local archive directory
you copy manually
you copy manually
Standby A
Standby B

### mongodump

ClusterControl uses the standard command to perform mongodump with `--journal`, which allows mongodump operations to use the durability journal to ensure that the export is in a consistent state against shards.

**Info**

mongodump (and mongorestore) is not available for MongoDB Sharded Cluster 4 and later that have sharded transactions in progress, as backups created with mongodump do not maintain the atomicity guarantees of transactions across shards.

### Percona Backup for MongoDB

Percona Backup for MongoDB (PBM) is an open-source backup solution designed for MongoDB replica sets and sharded clusters. It provides distributed, consistent, and low-impact backup operations with support for physical backups, logical backups, point-in-time recovery (PITR), and oplog-based recovery.

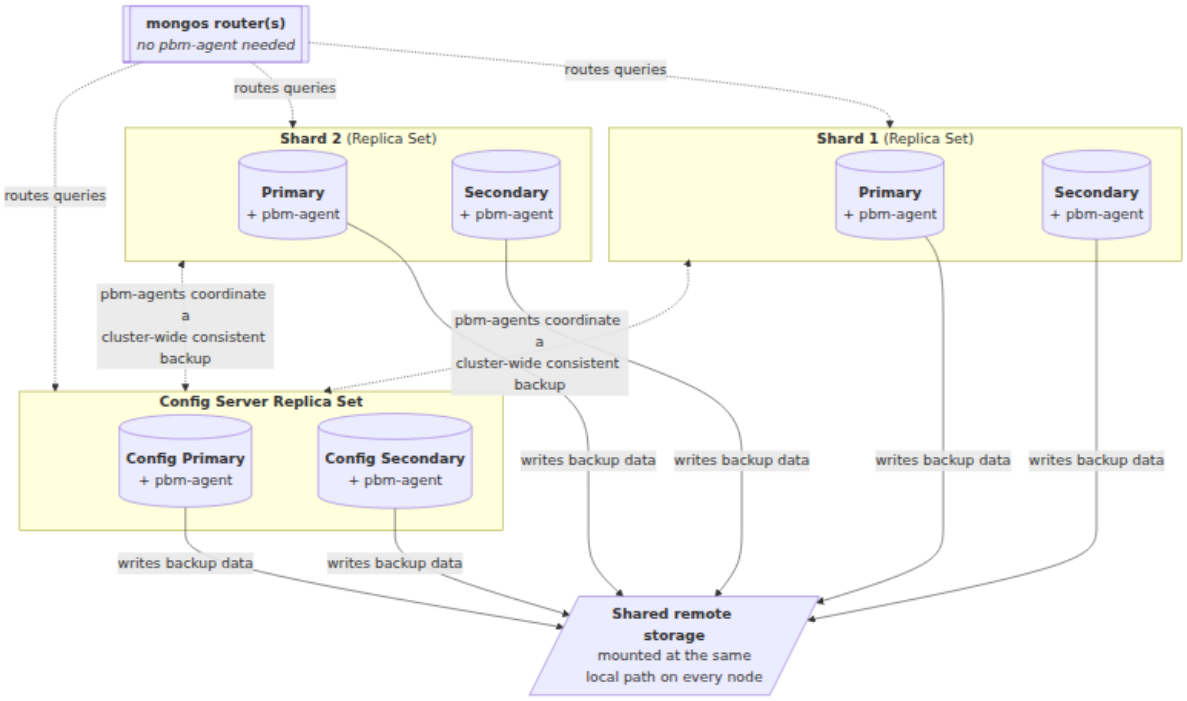
PBM supports Percona Server for MongoDB and MongoDB Community Edition deployments running with replication enabled. Since PBM relies on MongoDB replication and oplog synchronization, standalone MongoDB instances are not supported.

**Info**

ClusterControl only supports Percona Backup for MongoDB for MongoDB sharded clusters. For MongoDB replica sets, use mongodump.

Percona Backup for MongoDB is the successor to the older "mongodb-consistent-backup" project, which is no longer maintained. PBM is now the recommended backup solution for modern MongoDB environments.

Since ClusterControl only supports PBM for sharded clusters, a `pbm-agent` runs on every `mongod` in the cluster (every shard's replica set members, plus the Config Server Replica Set), so the whole cluster can be backed up as one consistent unit. `mongos` routers don't hold data, so they don't need one. Every agent must write to the same shared remote storage, mounted at an identical local path on every node; see the warning under *Installing Percona Backup for MongoDB* below.




## Installing Percona Backup for MongoDB

Percona Backup for MongoDB requires an extra step for installation and configuration, and it is not enabled by default. You can use ClusterControl to install this tool by going to ***ClusterControl GUI → Clusters → {cluster} → Action → Install Percona Backup***, or simply choose the backup method **percona-backup-mongodb** from the dropdown list when configuring a backup method for the first time. If the tool is not installed, ClusterControl will advise on installing the backup tool first before moving forward to the next step.

1. Click **Continue** button to proceed on installing the `percona-backup-mongodb` backup tool.
2. Specify the **Backup directory**. Make sure that the backup directory path exists.
3. Click **Install** button.

**Warning**

Attention

Percona Backup for MongoDB requires a remote file server mounted to a local directory. It is the responsibility of the server administrators to guarantee that the same remote directory is mounted at exactly the same local path on all servers in the MongoDB cluster or non-sharded replica set. If the path is accidentally a normal local directory, errors will eventually occur, most likely during a restore attempt.

## Redis/Valkey RDB & AOF

RDB (Redis Database Backup) is a snapshot-based persistence format used by Redis and Valkey. It stores the dataset in a compact binary file at a specific point in time and is commonly used for backups and recovery.

AOF (Append Only File) is a persistence mechanism that records every write operation received by the

server. This allows the database state to be reconstructed by replaying the logged operations.

When backing up Redis or Valkey, ClusterControl backs up the RDB file and, if AOF persistence is enabled, the corresponding AOF files from the selected database node.

### MSSQL Backup

For Microsoft SQL Server (MSSQL), ClusterControl uses the native backup capabilities provided by SQL Server and does not rely on external backup tools. Backup operations are executed through the `sqlcmd` client, which connects to the SQL Server instance and performs the requested backup tasks.

ClusterControl supports the following MSSQL backup types:

- Full Backup - Creates a complete backup of all selected databases. A full backup captures the entire database state at the time the backup operation completes.
- Differential Backup - Backs up only the data changed since the last full backup. Differential backups are smaller and faster than full backups and are grouped under their associated full backup on the **Backup** page.
- Transaction Log Backup - Backs up transaction logs to support point-in-time recovery and reduce transaction log growth. At least one full backup must exist before transaction log backups can be created.

**Warning**

Attention

Differential backups typically grow larger over time because they include all changes made since the last full backup. For example, in a write-intensive workload, the first hourly differential backup after a full backup may be relatively small, while the last differential backup before the next scheduled full backup can become significantly larger due to the accumulated changes throughout the day.

Take note of the following operational behavior performed by ClusterControl for MSSQL backups:

- Backups are always performed on the primary node, regardless of the selected backup type.
- Backup files are stored in the `/var/lib/backups` directory.
- Backup data is stored only on the primary node.
- Backup files are always encrypted. ClusterControl always adds `ENCRYPTION (ALGORITHM = AES_256, SERVER CERTIFICATE = ...)` to the generated `BACKUP DATABASE` statement, using a certificate it manages. This is SQL Server's own native backup encryption, separate from the OpenSSL-based encryption used for other backup methods, and is not a configurable option.
- Backups cannot run if the primary node is unavailable, since ClusterControl cannot identify a valid backup candidate.
- If the selected backup type is not Full (e.g, Differential or Transaction Log) and no existing full backup is found in the backup directory, ClusterControl automatically performs a full backup instead. This full backup becomes the base backup for subsequent differential or transaction log backups.

For example, a full backup runs a statement equivalent to:

```
BACKUP DATABASE [myexampledb] TO DISK = '/
↪var/lib/backups/BACKUP-838222/myexampledb_full.bak'
```

(continues on next page)

(continued from previous page)

```
WITH CHECKSUM, COMPRESSION, ENCRYPTION (ALGORITHM = AES_256, SERVER␣
↪CERTIFICATE = S9sDbmEncryptCert)
```

A common backup strategy is to perform full backups periodically (for example, weekly), differential backups more frequently (such as daily), and transaction log backups at shorter intervals throughout the day.

One limitation of MSSQL backups is version compatibility. Backups created on newer versions of SQL Server cannot be restored on older SQL Server versions.

### ClickHouse native backup

ClusterControl uses ClickHouse's own native `BACKUP/RESTORE` SQL commands rather than an external backup tool. By default, a backup is written to the `backups` disk that ClusterControl registers on the backup node itself (typically `/var/lib/clickhouse/backups/`), using ClickHouse's `Disk('backups', ...)` destination. Setting a custom storage directory switches the backup to ClickHouse's `File(...)` destination at that path instead; storing the backup on the ClusterControl controller is not supported either way.

Incremental backups automatically chain from the most recently completed backup on the same storage host/node; no parent backup ID needs to be supplied. Backups can optionally be uploaded to, or streamed directly to, S3-compatible cloud storage, the same as other physical backup methods.

ClickHouse does not support point-in-time recovery (PITR); its native backup is snapshot-based rather than log-replay-based.

Verifying a ClickHouse backup that was written to the local `backups` disk requires the backup directory to be reachable at an identical path on the *Backup Verification* server through shared storage configured outside ClusterControl; otherwise the restore fails with `BACKUP_NOT_FOUND`. This requirement does not apply to backups stored in cloud storage.

### Elasticsearch Snapshot

An Elasticsearch snapshot is a way to back up your Elasticsearch indices and data. The process involves creating a snapshot of one or more indices and storing it in a designated repository. This snapshot can then be used to restore the indices to their previous state in case of data loss or other issues. The snapshot process is incremental, meaning that only changes made since the last snapshot are stored. Snapshots can be taken manually or automatically on a schedule, and can also be used to migrate data between clusters or to create new indices.

Note that during cluster deployment, ClusterControl configures Elasticsearch snapshot storage by requiring the specification of the repository name, storage location, and file system path early in the process. The snapshot repository will then be available under the **Backups** section of the Elascticsearch cluster.

## 2.60 Create Backup

The **Create Backup** feature in ClusterControl lets you take an on-demand backup of a database cluster. Unlike *Schedule Backup*, which repeats on a cron-based recurrence, Create Backup runs once, immediately, using the options you configure in the wizard.

Compared to *Schedule Backup*, Create Backup does not include:

- Backup verification, which restores the completed backup to a standalone Backup Verification Server (BVS) to confirm it is recoverable.

- Backup failover, which automatically retries the backup on an alternative database node if the selected node is unavailable.
- Binary log (binlog) backup support for MySQL and MariaDB databases (see *Schedule Backup*).
- Recurring schedules with timezone support.
- Pausing and resuming.

Aside from these differences, Create Backup shares the same backup methods, Advanced Settings, and job_data fields as Schedule Backup for each database type.

### 2.60.1 Supported backup methods

For creating a backup, ClusterControl supports the following backup methods:

| Cluster type | Supported backup methods |
|---|---|
| MySQL-based clusters | mysqldumpPercona Xtrabackup (full, incremental) |
| MariaDB-based clusters | mariadb-dumpMariaDB Backup (full, incremental) |
| PostgreSQL-based clusters | pg_dumppg_dumpallpg_basebackup (full, incremental - only for v17 and later)pgBackRest (full, incremental, differential) |
| MongoDB-based clusters | mongodumpPercona Backup for MongoDB |
| Redis-based clusters | RDB (includes AOF file if AOF persistence is enabled) |
| SQL Server | FullDifferentialTransaction log |
| Elasticsearch | Elasticsearch snapshot |
| ClickHouse-based clusters | ClickHouse native backup (full, incremental) |

**Note**

Note Binlog backup is not available through Create Backup. It can only be scheduled; see *Schedule Backup*.

For more details on each of the supported backup methods, see *Backup methods*.

---

### 2.60.2 Prerequisites

Before creating a backup in ClusterControl, ensure that your environment meets the necessary requirements:

- The database cluster must be managed by ClusterControl.
- Only backup methods supported by ClusterControl can be used. See *Supported backup methods*.
- Ensure that the required backup tool binaries are installed and accessible on the target nodes.

- Backups created in ClusterControl can be stored either on the ClusterControl host or on the target database nodes. If you use an external backup storage host, you can mount it via NFS and share the volume for backup storage.
- If the backup destination is the ClusterControl host, ensure that port 9999 (or within the range 9990-9999) is open from the database node to the ClusterControl host for backup streaming. These ports are configurable under ***Backup settings → Netcat port***.
- Some backup methods support storing backups in cloud storage. Configure cloud credentials under ***ClusterControl GUI → Settings → Cloud storage credentials*** beforehand if you plan to use this feature.
- Ensure that system clocks are synchronized (e.g., via NTP) across all nodes.

**Tip**

Configure email or alerting integrations to receive backup status notifications. See *Notification Service*.

---

### 2.60.3 Backup jobs

A backup created on demand runs as a background thread (RUNNING3) and does not block other non-backup jobs in the queue. Even if a backup takes hours to complete, other non-backup jobs can continue to run simultaneously via the main thread (RUNNING). You can monitor job progress under ***ClusterControl GUI → Activity center → Jobs***.

ClusterControl creates a backup user (prefixed with `backupuser`) if it does not already exist. If this user encounters issues accessing the database server, it will be reset based on the `backup_user` and `backup_user_password` values defined in the CMON configuration file.

Note that ClusterControl executes **only one backup job per cluster at a time**. If a backup schedule's recurrence overlaps with an on-demand backup you create manually, the second job waits until the first one completes.

### 2.60.4 Common Wizard Steps

The create backup wizard follows the same flow and step order for every database type, though a few steps are skipped or restricted for certain backup methods (noted inline where that applies). Only the **Configuration** and **Advanced Settings** steps have database-specific options, which are documented in *Database-Specific Settings* below.

#### ClusterControl GUI

**Opening the wizard**

1. Go to the **Clusters** dashboard and click your target cluster.
2. Go to the **Backups** tab and click **Create backup**.
3. Choose **Backup on Demand**.

---

**Step 1: Configuration**

- **Cluster**: Pre-set to the cluster you selected.

- **Backup host**: Choose the node to run the backup from, or select **Auto Select** to let ClusterControl pick the least-loaded node. Not available for MongoDB Replica Set (see *MongoDB* below).
- **Backup method**: Available methods vary by database type; see *Database-Specific Settings*.
- **Upload backup to cloud**: Defaults to **Off**. Set to **On** if you have cloud credentials configured and want to upload the backup.
- **Stream backup to the cloud**: Only appears when **Upload backup to cloud** is **On**. Supported for `xtrabackupfull`, `xtrabackupincr`, `mariabackupfull`, `mariabackupincr`, `pg_basebackup`, and `pg_basebackupincr`. When **On**, the backup is written directly to cloud storage without being staged locally first, and the Local Storage step is skipped. When **Off**, the backup is stored locally first and then uploaded; you can optionally delete the local copy after upload in the **Cloud Storage** step.

Click **Continue**.

---

#### Step 2: Advanced Settings

The following fields are common across most backup methods. Additional method-specific fields are listed per database type in *Database-Specific Settings*.

- **Compression**: Defaults to **On**. When off, backup files are stored uncompressed.
- **Retention**: Defaults to `31 days`. Adjust the number of days, or set to **Off** to disable automatic deletion.
- **Compression level**: Defaults to `6 (System default)`.
- **Enable encryption**: Defaults to **Off**. When enabled, ClusterControl auto-generates an encryption key stored in `/etc/cmon.d/cmon_<cluster_id>.cnf`.

Click **Continue**.

---

#### Step 3: Local Storage

> **Note**
>
> Note pgbackrest, MongoDB Sharded Cluster, and Elasticsearch backups skip this step. When **Stream backup to the cloud** is **On** in Step 1, this step is also skipped for the methods that support it.

- **Storage location**: Choose **Store on Controller** to store the backup inside the ClusterControl node. This requires `socat` or `netcat` on the source and destination host. By default, ClusterControl uses port 9999 to stream the backup created on the database node to the ClusterControl node. This is configurable under ***Backup settings → Netcat port***. Choose **Store on Node** to store the backup inside the target database node.
- **Storage directory**: You can opt to use another backup directory as you wish. If you leave this field blank, ClusterControl will use the default backup directory specified in the ***Backup settings → Default directory***.
- **Backup subdirectory**: Defaults to `BACKUP-%I`, where `%I` is the numeric backup job ID assigned to each run. See *Backup subdirectory*.

Click **Continue**.

---

#### Step 4: Cloud Storage

!!! note "Cloud storage support"

```
pgBackRest and Elasticsearch do not support cloud storage backups.
```

This step only appears when **Upload backup to cloud** is **On** in Step 1. Otherwise, this step is skipped.

- **Credentials**: Select from credentials created in **Settings → Cloud storage credentials**. Supported providers: *Amazon Web Services*, *Google Cloud*, *Microsoft Azure*, *S3 Compatible storage provider*.
- **Bucket**: Your target storage bucket. Click **+** to create a new bucket, then confirm with **OK**. Only buckets permitted by the selected credentials will appear in the dropdown.
- **Location path**: Sub-path within the bucket for this backup. You can define multiple directory levels relative to the bucket root. Do not include a trailing slash. For example, if set to `server1/fullbackup/weekly`, the backup will be stored under `server1/fullbackup/weekly/BACKUP-151`, where `BACKUP-151` is derived from the **Backup subdirectory** value.
- **Retention period**: Defaults to **On**. Set the number of days to keep cloud backups, or **Off** to retain indefinitely.
- **Delete after upload**: Defaults to **Off**. Set to **On** to remove the local backup copy after a successful cloud upload, reclaiming local disk space. The backup will still appear in the backup list and can be restored as usual. During restoration, ClusterControl will download the backup from the cloud.

Click **Continue**.

---

#### Step 5: Preview

Review all selected options. This is the final confirmation before ClusterControl creates and runs the backup job.

### ClusterControl CLI

Backups on demand use `s9s backup --create` with `--wait --log` to print job progress while it runs. Omit `--recurrence`; including it turns the same command into a *scheduled backup* instead of a one-off job.

Specific commands are listed in each database section below.

**Manage existing backups:**

- List all backups for a cluster:

```
s9s backup \
    --list \
    --cluster-id=<ID> \
    --long
```

- Delete a backup by its backup ID:

```
s9s backup \
    --delete \
    --backup-id=<BACKUP_ID>
```

To find the backup ID, run `--list` first and note the ID column.

### ClusterControl RPC API

Create Backup does not have a dedicated convenience operation like `scheduleBackup`. A one-off backup job is created through the generic `createJobInstance` call on `POST /v2/jobs`, with `job.job_spec.command` set to `backup`. All calls require a TLS connection and, in most cases, an authenticated session or inline credentials. See *ClusterControl RPC API → Authenticating with the API* for details.

- Create a backup using the `createJobInstance` operation. The `job_data` fields differ by backup method; see the *job_data reference* below and each database section for method-specific examples.

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 1,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "mysqldump",
                "hostname": "192.168.40.54",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

- List all backups for a cluster:

```
curl -k -XPOST \
-d '{
    "operation": "getBackups",
    "cluster_id": 1,
    "ascending": true
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

The response includes a `backup_records` array and a `total` count.

- Delete a backup by its backup ID:

```
curl -k -XPOST \
-d '{
    "operation": "deleteBackupRecord",
    "backup_record": {
        "backup_id": 2
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

This removes the backup metadata from the controller; it does not delete the backup files from storage. Use `deleteBackupList` with a `backup_id_list` (comma-separated) to delete several backups in one call.

---

### 2.60.5 Database-Specific Settings

Each section below covers the supported backup methods for that database type, any **Advanced Settings** fields unique to each method beyond the common fields above, and CLI and RPC API examples.

#### MySQL & MariaDB

Covers MySQL & Percona & MariaDB standalone and primary-replica replication clusters, as well as Galera Cluster topologies (Percona XtraDB Cluster, MariaDB Galera Cluster). The backup methods, Advanced Settings, and job_data fields are identical across topologies; only the cluster/node you target differs.

**Backup methods:**

| Method | Type | Cloud upload | Notes |
|---|---|---|---|
| `mysqldump` | Logical | Yes | Dump types: *Schema only*, *MySQL DB Only*, *Data Only*, *Schema And Data*, *Complete PITR-compatible* |
| `xtrabackupfull` | Physical | Yes | Percona Server and Oracle MySQL clusters |
| `xtrabackupincr` | Physical | Yes | Percona Server and Oracle MySQL clusters |
| `mariabackupfull` | Physical | Yes | MariaDB clusters |
| `mariabackupincr` | Physical | Yes | MariaDB clusters |

#### ClusterControl GUI

For Galera clusters (Percona XtraDB Cluster, MariaDB Galera Cluster), select your Galera cluster in the **Cluster** field. All Advanced Settings below apply identically regardless of topology.

**mysqldump - unique Advanced Settings**

- **Enable partial backup**: When **On**, specify individual databases to include instead of backing up all databases.
- **Use extended insert**: Defaults to **On**. When off, `mysqldump` generates single-row `INSERT` statements.
- **One dump file per DB**: Defaults to **Off**. When on, `mysqldump` writes a separate backup file per database.

**xtrabackup / mariabackup - unique Advanced Settings**

These settings apply to both full and incremental variants. Select the specific variant in the **Backup method** field (`xtrabackup (full)`, `xtrabackup (incremental)`, `mariabackup (full)`, or `mariabackup (incremental)`).

- **Enable partial backup**: When **On**, specify individual databases to include.
- **Use PIGZ for parallel gzip**: Defaults to **Off**. Enables parallel compression, which significantly speeds up backups of large datasets.
- **Use ZSTD compression**: Defaults to **Off**. Uses Zstandard compression. For MySQL & Percona Server versions < 8.0.34, QPress is used instead.
- **Backup locks**: Defaults to **On**. When on, xtrabackup uses `LOCK TABLES FOR BACKUP`. When off (`--no-backup-locks`), uses `FLUSH NO_WRITE_TO_BINLOG TABLES` and `FLUSH TABLES WITH READ LOCK`.
- **Lock DDL per table**: Defaults to **Off**. Sets `--lock-ddl-per-table`. Enable if you encounter errors like `InnoDB: An optimized (without redo logging) DDL operation has been performed`. See Percona blog, MDEV-14095, MDEV-5336. May cause update delays on heavily loaded servers.
- **Xtrabackup parallel copy threads**: Defaults to `1`. Increase to speed up streaming during the copy phase.
- **Network streaming throttle rate (MB/s)**: Defaults to `0` (no throttle). Uses the `pv` tool when a non-zero value is set.
- **Desync node during backup** (Galera clusters only): Defaults to **Off**. When **On**, sets `wsrep_desync=ON` on the backup node before the backup starts, so it can fall behind without triggering flow control on the rest of the cluster. The node catches up and `wsrep_desync` is set back to **Off** once the backup finishes. Not shown for non-Galera (replication) clusters.

### ClusterControl CLI

- Create a backup using mysqldump:

```
s9s backup \
    --create \
    --cluster-id=127 \
    --nodes="192.168.40.54:3306" \
    --backup-method=mysqldump \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

- Create a full xtrabackup:

```
s9s backup \
    --create \
    --cluster-id=99 \
    --nodes="192.168.40.60:3306" \
    --backup-method=xtrabackupfull \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

Use `xtrabackupincr` for incremental backups.

- Create a full mariabackup:

```
s9s backup \
    --create \
    --cluster-id=126 \
    --nodes="192.168.40.52:3306" \
    --backup-method=mariabackupfull \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

Use `mariabackupincr` for incremental backups.

- Create a mysqldump backup for a Galera cluster (Percona XtraDB Cluster):

```
s9s backup \
    --create \
    --cluster-id=124 \
    --nodes="192.168.40.67:3306" \
    --backup-method=mysqldump \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

The same commands used above for replication clusters apply to Galera clusters (`xtrabackupfull`/`xtrabackupincr`, `mariabackupfull`/`mariabackupincr`). Substitute your Galera cluster ID and a Galera node in `--nodes`. **Desync node during backup** has no `s9s backup` CLI flag yet; use the GUI or RPC API to enable it for `xtrabackupfull`/`xtrabackupincr`/`mariabackupfull`/`mariabackupincr` on Galera clusters.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Create a mysqldump backup:

```
curl -k -XPOST \
-d '{
```

(continues on next page)

(continued from previous page)

```
    "operation": "createJobInstance",
    "cluster_id": 127,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "mysqldump",
                "hostname": "192.168.40.54",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "mysqldump_type": "Complete",
                "extended_insert": true
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

See `mysqldump_type` values and all available fields in the *job_data reference*.

- Create a full xtrabackup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 99,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "xtrabackupfull",
                "hostname": "192.168.40.60",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "xtrabackup_backup_locks": true,
                "xtrabackup_parallellism": 1
            }
```

(continues on next page)

(continued from previous page)

```
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

Use `"backup_method": "xtrabackupincr"` for incremental, `"mariabackupfull"` or `"mariabackupincr"` for MariaDB clusters. See all available fields in the *job_data reference*.

- Create a mysqldump backup for a Galera cluster (Percona XtraDB Cluster):

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 124,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "mysqldump",
                "hostname": "192.168.40.67",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "mysqldump_type": "Complete"
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

The same `job_data` fields used above for replication clusters apply to Galera clusters. Set `cluster_id` and `hostname` to your Galera cluster and node.

- Create a full xtrabackup for a Galera cluster with the node desynced during the backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 124,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
```

(continues on next page)

(continued from previous page)

```
                "backup_method": "xtrabackupfull",
                "hostname": "192.168.40.67",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "xtrabackup_backup_locks": true,
                "xtrabackup_parallellism": 1,
                "wsrep_desync": true
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

`wsrep_desync` only applies to Galera clusters using `xtrabackupfull`/`xtrabackupincr` or `mariabackupfull`/`mariabackupincr`. Omit it (or set `false`) for replication clusters or for `mysqldump`. See *xtrabackupfull / xtrabackupincr* for the full field reference.

**Common steps**

All other wizard steps (Local Storage, Cloud Storage, Preview) follow the *Common Wizard Steps* above.

---

### PostgreSQL & TimescaleDB

**Backup methods:**

| Method | Type | Cloud upload | Notes |
|---|---|---|---|
| `pgdumpall` | Logical | Yes | Dumps all databases in the cluster |
| `pgdump` | Logical | Yes | Dumps one or more named databases with format and content options |
| `pg_basebackup` | Physical | Yes | PITR disabled if pgbackrest manages WAL |
| `pg_basebackup` | Physical | Yes | Incremental physical backup. PostgreSQL 17+ only; requires `summarize_wal = on` |
| `pgbackrest-full` | Physical | Yes | Full backup |
| `pgbackrestd-iff` | Physical | Yes | Differential backup |
| `pgbackrestincr` | Physical | Yes | Incremental backup |

**Info**

`pg_basebackup` and `pg_basebackupincr` both support cloud upload (S3-compatible storage). Enable **Stream backup to the cloud** to write the backup directly to cloud storage as it runs, instead of staging it on local disk first. See **Stream backup to the cloud** in *Common Wizard Steps*.

### ClusterControl GUI

**pgdumpall - unique Advanced Settings**

- **Enable partial backup**: When **On**, specify individual databases to back up instead of the full cluster.

**pgdump - unique Advanced Settings**

`pgdump` backs up one or more named databases and provides additional format and content control compared to `pgdumpall`.

- **Include databases**: Specify the database names to include (required).
- **Backup format**: `Plain`, `Custom`, `Directory`, or `Tar`.
- **Schema only**: When **On**, only dumps the schema (no data).
- **Data only**: When **On**, only dumps the data (no schema).
- **No owner**: When **On**, omits `ALTER OWNER` statements.
- **No privileges**: When **On**, omits `GRANT` and `REVOKE` statements.

**pg_basebackup - unique Advanced Settings**

No method-specific fields beyond the common Advanced Settings (Compression, Retention, Compression level, Enable encryption). To upload to cloud storage, set **Upload backup to cloud** to **On** in Step 1; turn on **Stream backup to the cloud** to stream directly to S3-compatible storage without staging locally first.

**pg_basebackupincr - prerequisites and unique settings**

`pg_basebackupincr` requires PostgreSQL 17 or later and `summarize_wal = on` on the backup source node. The wizard checks this automatically and warns if the setting is off.

Before creating an incremental backup, enable WAL summarization on the node:

1. In the cluster node list, click the action menu (**…**) on the target node.
2. Select **Configure WAL**.
3. Set **Summarize WAL** to **On** and confirm.

After WAL summarization is enabled, `pg_basebackupincr` appears as a selectable backup method. Each incremental backup chains from the last full `pg_basebackup`. When restoring, ClusterControl automatically reconstructs the full chain using `pg_combinebackup`.

**pgbackrest (full, diff, incr) - installation and unique settings**

!!! note “pgbackrest installation”

```
If pgbackrest is not yet installed, the Configuration step shows an␣
↪installation panel before proceeding. Choose a **Configuration method**:

- **Primary**: Install on the primary node only. The backup repository is␣
```

(continues on next page)

(continued from previous page)

```
↪created on the primary. No SSH configuration between hosts.
- **All DB nodes**: Install on all database hosts. The backup repository is␣
↪on the primary; backups use a standby node. SSH is configured between␣
↪hosts.
- **All DB nodes and dedicated repository host**: Install on all DB nodes␣
↪and a separate repository host. Specify the **Dedicated repository host**␣
↪(IP, hostname, or FQDN).

Click **Continue** to install and verify the job completes successfully␣
↪before proceeding.
```

pgbackrest backups skip the **Local Storage** step. Backup data is managed by pgbackrest's own repository configuration.

The Advanced Settings for pgbackrest are limited to: Compression, Compression level, and Enable encryption.

### ClusterControl CLI

- Create a pgdumpall backup:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pgdumpall \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

- Create a pgdump backup of a specific database:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pgdump \
    --databases=mydb \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

- Create a pg_basebackup:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackup \
```

(continues on next page)

(continued from previous page)

```
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

To upload to cloud storage, add the same `--cloud-provider`, `--s3-bucket`, and `--credential-id` flags shown in the *xtrabackup cloud upload example*. Add `--cloud-only` to stream the backup directly to cloud storage without staging a local copy first; when using `--cloud-only`, remove `--backup-dir` and `--on-controller`, since those stage the backup on local/controller storage, which conflicts with streaming it directly to cloud storage.

- Enable WAL summarization on a node, then create an incremental pg_basebackup (PostgreSQL 17+ only):

```
# Step 1: enable summarize_wal on the target node (one-time setup)
s9s node \
    --configure-wal \
    --summarize-wal=on \
    --nodes="192.168.40.63:5432" \
    --cluster-id=125 \
    --wait \
    --log

# Step 2: create the incremental backup
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackupincr \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

`--summarize-wal=on` must be set before the first incremental backup run. Each incremental backup chains from the most recent full `pg_basebackup`. Restore is automatic: ClusterControl reconstructs the chain with `pg_combinebackup`.

- Create a pgbackrest full backup:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pgbackrestfull \
    --wait \
    --log
```

pgbackrest manages its own repository; `--backup-dir` and `--on-controller` are not used. Use `pgbackrestdiff` or `pgbackrestincr` for differential and incremental backups.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Create a pgdumpall backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 125,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "pgdumpall",
                "hostname": "192.168.40.63",
                "port": 5432,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "schema_only": false,
                "data_only": false
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

- Create a pgdump backup of a specific database:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 125,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "pgdump",
                "hostname": "192.168.40.63",
                "port": 5432,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
```

(continues on next page)

(continued from previous page)

```
                "backup_retention": 31,
                "include_databases": "mydb",
                "backup_format": "custom",
                "schema_only": false,
                "data_only": false
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

See `no_owner` and `no_privileges` in the *job_data reference*.

- Create a pgbackrest full backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 125,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "pgbackrestfull",
                "hostname": "auto",
                "port": 5432,
                "compression": true,
                "compression_level": 6
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

pgbackrest does not use `backupdir` or `cc_storage`. Use `"backup_method": "pgbackrest-diff"` or `"pgbackrestincr"` for differential and incremental.

- Create an incremental pg_basebackup (PostgreSQL 17+ only; requires `summarize_wal = on`):

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 125,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
```

(continues on next page)

(continued from previous page)

```
                "backup_method": "pg_basebackupincr",
                "hostname": "192.168.40.63",
                "port": 5432,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

Enable WAL summarization on the node before running (see the ClusterControl CLI tab or the GUI Configure WAL action). Cloud upload to S3-compatible storage is supported for `pg_basebackupincr`.

**Common steps**

Cloud Storage and Preview follow the *Common Wizard Steps* above.

---

## MongoDB

MongoDB Replica Set and Sharded Cluster topologies each use their own backup method. Unlike MySQL & MariaDB, `mongodump` and `percona-backup-mongodb` are not interchangeable between topologies.

**Backup methods:**

| Method | Type | Cloud upload | Topology | Notes |
|---|---|---|---|---|
| `mongodump` | Logical | Yes | Replica Set | Backup always runs on a replica node; the **Backup host** field is not shown |
| `percona-backup-mor` (PBM) | Physical | No | Sharded Cluster | Requires MongoDB 4.0 or later and the PBM agent installed on every node |

### ClusterControl GUI

**mongodump (Replica Set)**

Only **Retention** and **Enable encryption** are configurable in Advanced Settings. Compression settings do not apply to `mongodump`.

**percona-backup-mongodb (Sharded Cluster)**

!!! note “PBM installation”

```
If PBM is not installed, the Configuration step shows an installation␣
↪panel. Specify the **Backup directory** (the path must already exist on␣
↪the filesystem), then click **Install**. Verify the installation job␣
↪completes before continuing.
```

Only **Retention** is configurable (defaults to `31 days`). No compression or encryption options are available. This backup type skips the **Local Storage** and **Cloud Storage** steps.

### ClusterControl CLI

**mongodump (Replica Set)**

- Create a mongodump backup:

```
s9s backup \
    --create \
    --cluster-id=79 \
    --nodes="192.168.40.160:27017" \
    --backup-method=mongodump \
    --backup-dir=/storage/backups \
    --on-controller \
    --wait \
    --log
```

`--nodes="192.168.40.160:27017"` specifies the replica set member and its MongoDB port.

**percona-backup-mongodb (Sharded Cluster)**

Before creating a backup, install the PBM agents on all nodes if not already present:

```
s9s cluster \
    --add-node \
    --nodes="pbmagent://*?backup_dir=/opt/pbm/backups/" \
    --cluster-id=28 \
    --wait \
    --log
```

`--nodes="pbmagent://*?backup_dir=/opt/pbm/backups/"` targets all nodes. Change `/opt/pbm/backups/` to your preferred directory.

After PBM is installed, create the backup:

- Create a percona-backup-mongodb backup:

```
s9s backup \
    --create \
    --cluster-id=28 \
    --nodes="192.168.40.170:27019" \
    --backup-method=percona-backup-mongodb \
    --backup-retention=31 \
    --wait \
    --log
```

`--nodes="192.168.40.170:27019"` specifies a config server replica set node and its port.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

**mongodump (Replica Set)**

- Create a mongodump backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 79,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "mongodump",
                "hostname": "192.168.40.160",
                "port": 27017,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "backup_retention": 31
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

**percona-backup-mongodb (Sharded Cluster)**

Install PBM agents via the CLI before using the RPC API to create the backup (see the ClusterControl CLI tab above).

- Create a percona-backup-mongodb backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 28,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "percona-backup-mongodb",
                "hostname": "192.168.40.170",
                "port": 27019,
                "backup_retention": 31
            }
        }
```

(continues on next page)

(continued from previous page)

```
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

**Common steps**

For `mongodump`, Local Storage, Cloud Storage, and Preview follow the *Common Wizard Steps* above. For `percona-backup-mongodb`, only Preview applies. Local Storage and Cloud Storage are skipped.

---

### Redis & Valkey

Applies to Redis and Valkey, in both Sentinel and Cluster topologies.

**Backup methods:**

| Method | Type | Cloud upload |
|---|---|---|
| `rdb` | Physical | Yes |

Backups are transferred from the cluster to the ClusterControl controller using `socat` over port `9999`. The port can be changed in **Backup Settings**.

**Note**

Note If AOF (Append-Only File) persistence is enabled on the Redis or Valkey instance, ClusterControl automatically includes the AOF file in the backup.

#### ClusterControl GUI

**Backup host**: For Redis Sentinel clusters, defaults to the primary node; you can choose a replica instead.

**Unique Advanced Settings:**

- **Use PIGZ for parallel gzip**: Defaults to **Off**. Enables parallel compression for faster backup of large datasets.

#### ClusterControl CLI

- Create a backup:

```
s9s backup \
    --create \
    --cluster-id=84 \
    --nodes="192.168.40.50:6379" \
    --backup-retention=31 \
    --wait \
    --log
```

No `--backup-method` is required; ClusterControl determines the method from the cluster type.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Create a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 84,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "rdb",
                "hostname": "192.168.40.50",
                "port": 6379,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "backup_retention": 31
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

**Common steps**

Local Storage, Cloud Storage, and Preview follow the *Common Wizard Steps* above.

---

### SQL Server

ClusterControl uses SQL Server's built-in backup engine. No external tool is required.

**Backup methods:**

| Method | Notes |
|---|---|
| `mssqlfull` | Complete database backup |
| `mssqldiff` | Changes since the last full backup |
| `mssqllog` | Log backup for point-in-time recovery |

**Note**

Note Backups always run on the primary node. If the primary is unavailable, the backup cannot proceed. If no previous full backup exists in the target directory, ClusterControl automatically upgrades any Differential or Transaction Log request to a Full backup.

### ClusterControl GUI

**Unique Advanced Settings:**

- **Include system databases**: Defaults to **On**. Set to **Off** to exclude system databases from the backup scope.
- **Encryption**: Always on and not configurable. Every SQL Server backup is encrypted natively by SQL Server itself (`ENCRYPTION (ALGORITHM = AES_256, SERVER CERTIFICATE = ...)` in the generated `BACKUP DATABASE` statement), using a certificate ClusterControl manages. This is separate from the **Enable encryption** toggle used by other backup methods, which does not apply here.

**Local Storage note:** The **Storage location** field is disabled for SQL Server. Backups are always stored on the primary node. The **Storage directory** and **Backup subdirectory** fields remain configurable.

### ClusterControl CLI

- Create a full backup:

```
s9s backup \
    --create \
    --cluster-id=42 \
    --nodes="pupnode195:1433" \
    --backup-method=mssqlfull \
    --backup-retention=31 \
    --wait \
    --log
```

- Create a differential backup:

```
s9s backup \
    --create \
    --cluster-id=42 \
    --nodes="pupnode195:1433" \
    --backup-method=mssqldiff \
    --backup-retention=31 \
    --wait \
    --log
```

- Create a transaction log backup:

```
s9s backup \
    --create \
    --cluster-id=42 \
    --nodes="pupnode195:1433" \
```

(continues on next page)

(continued from previous page)

```
    --backup-method=mssqllog \
    --backup-retention=7 \
    --wait \
    --log
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

SQL Server backups always run on the primary node. Use `"hostname": "auto"` to let ClusterControl select the primary automatically, which is recommended for Always On Availability Group clusters.

- Create a full SQL Server backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 42,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "mssqlfull",
                "hostname": "auto",
                "backupdir": "/var/opt/mssql/data/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": false,
                "compression": true,
                "compression_level": -1,
                "backup_retention": 31,
                "backup_system_db": true
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

Use `"backup_method": "mssqldiff"` for differential or `"mssqllog"` for transaction log backups. `"compression_level": -1` uses the SQL Server default compression level.

**Common steps**

Cloud Storage and Preview follow the *Common Wizard Steps* above.

---

### Elasticsearch

ClusterControl uses the `elasticsearch-snapshot` tool (built-in; no external installation required).

**Backup methods:**

| Method | Notes |
|---|---|
| `elasticsearch-snapshot` | Uses the Elasticsearch snapshot API |

**Note**

Note Elasticsearch backups skip the **Local Storage** and **Cloud Storage** steps. Backup data is stored to the shared filesystem path defined by `elastic_shared_filesystem` in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf`.

### ClusterControl GUI

**Configuration step - unique fields:**

- **Repository**: Points to the repository defined by `elastic_snapshot_repository` in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf`. Additional repositories configured on the cluster appear in the drop-down.

**Advanced Settings:** Only **Retention** is configurable (defaults to `31 days`).

### ClusterControl CLI

- Create an Elasticsearch snapshot backup:

```
s9s backup \
    --create \
    --cluster-id=8 \
    --snapshot-repo-type=fs-nfs \
    --snapshot-repository=s9s-es-repo \
    --backup-retention=31 \
    --wait \
    --log
```

`--snapshot-repo-type=fs-nfs` sets the repository type. Other options are `fs` and `s3`. `--snapshot-repository=s9s-es-repo` is the repository name, found in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf` as `elastic_snapshot_repository`.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Create an Elasticsearch snapshot backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 8,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "elasticsearch-snapshot",
                "snapshot_repo_type": "fs-nfs",
                "snapshot_repository": "s9s-es-repo",
                "backup_retention": 31
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

`snapshot_repo_type` options: `fs-nfs`, `fs`, `s3`. `snapshot_repository` value comes from `elastic_snapshot_repository` in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf`.

**Common steps**

Preview follows the *Common Wizard Steps* above.

---

### ClickHouse

ClusterControl uses ClickHouse's own native `BACKUP` SQL command.

**Backup methods:**

| Method | Type | Cloud upload | Notes |
|---|---|---|---|
| `clickhouse-na` | Physical | Yes | Full backup |
| `clickhouse-na` | Physical | Yes | Incremental backup. Automatically chains from the most recent completed backup; no parent backup ID needs to be supplied |

!!! note "Storage location"

```
By default, ClickHouse native backups are written to the `backups` disk␣
↪that ClusterControl registers on the backup node itself (by default `/
↪var/lib/clickhouse/backups/`), using ClickHouse's `Disk('backups', ...)`␣
↪backup destination. Setting a custom **Storage directory** switches the␣
↪backup to ClickHouse's `File(...)` destination at that path instead.␣
↪Storing the backup on the ClusterControl controller is not supported.
```

### ClusterControl GUI

The wizard follows the same five steps as *Common Wizard Steps*: Configuration, Advanced settings, Local storage, Cloud storage, Preview.

- **Backup method dropdown**: Labeled **Clickhouse native (full)** and **Clickhouse native (incr)**.
- **Upload backup to cloud**: Same as *Common Wizard Steps* Step 1; when **On**, the **Cloud storage** step appears with the usual Credentials/Bucket/Location path/Retention period/Delete after upload fields.
- **Local storage note: Store on Controller** is disabled for ClickHouse; **Store on Node** is the only option. **Storage directory** is optional: leave it blank to use the node's pre-configured `backups` disk, or set a custom path to have ClickHouse write the backup there instead.

**Unique Advanced Settings:**

- **Enable partial backup**: When **On**, specify individual databases to include instead of backing up all databases.

### ClusterControl CLI

- Create a full backup:

```
s9s backup \
    --create \
    --cluster-id=4 \
    --nodes="10.10.15.11:9440" \
    --backup-method=clickhouse-native \
    --wait \
    --log
```

  Do not pass `--on-controller`; controller storage is not supported for ClickHouse. Omit `--backup-directory` to use the node's own `backups` disk, or set it to a custom path (see below).

- Create a full backup to a custom directory:

```
s9s backup \
    --create \
    --cluster-id=4 \
    --nodes="10.10.15.11:9440" \
    --backup-method=clickhouse-native \
    --backup-directory=/var/lib/clickhouse/backups2 \
    --wait \
    --log
```

With `--backup-directory` set, ClusterControl issues `BACKUP ... TO File('<path>/<backup name>')` instead of the default `Disk('backups', ...)`.

- Create an incremental backup:

```
s9s backup \
    --create \
    --cluster-id=4 \
    --nodes="10.10.15.11:9440" \
    --backup-method=clickhouse-native-incr \
    --wait \
    --log
```

- Create a full backup of a single database:

```
s9s backup \
    --create \
    --cluster-id=4 \
    --nodes="10.10.15.11:9440" \
    --backup-method=clickhouse-native \
    --databases=default \
    --wait \
    --log
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Create a full backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 4,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "clickhouse-native",
                "hostname": "10.10.15.11",
                "port": 9440,
                "compression": true
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

Do not include `cc_storage`; controller storage is not supported for ClickHouse. Omit `backupdir` to use the node's own `backups` disk, or set it to a custom path so ClusterControl issues

`BACKUP ... TO File(...)` instead of `Disk('backups', ...)`. Use `"backup_method": "clickhouse-native-incr"` for an incremental backup.

- Create a full backup of a single database:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 4,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "clickhouse-native",
                "hostname": "10.10.15.11",
                "port": 9440,
                "compression": true,
                "include_databases": "default"
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

See all available fields in the *job_data reference*.

**Common steps**

All five steps follow the *Common Wizard Steps* above; see the Local storage note above for the ClickHouse-specific restriction.

### 2.60.6 RPC API job_data Reference

The `job_data` object inside `job.job_spec.job_data` controls the backup configuration for the `createJobInstance` operation. Field names use the values that the RPC API accepts as input. These differ from the camelCase names that appear in backup record responses from `getBackups`.

Key naming differences between API input (`job_data`) and response (`config`):

| API input (job_data) | Response (config) | Notes |
|---|---|---|
| backup_method | backupMethod | |
| hostname | backupHost | Different name entirely |
| backupdir | backupDir | |
| backupsubdir | backupSubDir | |
| cc_storage | ccStorage | |
| compression_level | compressionLevel | |
| encrypt_backup | encrypt_backup | Same in both |
| use_pigz | usePigz | |
| use_qpress | useQpress | |
| xtrabackup_backup_locks | xtrabackup_locks | Different name |
| xtrabackup_parallellism | xtrabackupParallellism | |

### Common fields

Most backup methods accept these fields:

| Field | Type | Default | Description |
|---|---|---|---|
| backup_method | string | required | Method name (see per-method sections below) |
| hostname | string | "auto" | Node IP or hostname. "auto" lets ClusterControl select the least-loaded node |
| port | integer | varies | Database port |
| backupdir | string | /storage/backups | Directory where the backup is stored |
| backupsubdir | string | BACKUP- | Per-run subdirectory. %I expands to the numeric job ID |
| cc_storage | boole | true | true stores on the controller; false stores on the backup node |
| compression | boole | true | Enable compression |
| compression_level | integer | 6 | Level 1-9, or -1 for the tool default |
| backup_retenti | integer | 31 | -1 keeps the backup forever. 0 follows the default retention setting configured for the cluster. Any other positive value sets a custom retention in days for this backup |
| encrypt_backup | boole | false | Encrypt the backup. ClusterControl manages the key |
| cloud_only | boole | false | Stream the backup directly to cloud storage without creating a local copy first. Different from delete_after_upload, which keeps a local copy until the cloud upload succeeds |
| upload_backup_da | object | omit | See *Cloud upload* |

backup_failover and verify_backup/verify_backup_delay are not accepted here. Both backup failover and backup verification are Schedule Backup-only features; see *Schedule Backup → Backup failover* and *Schedule Backup → Backup verification*.

### Cloud upload

Include `upload_backup_data_to_cloud_storage` in `job_data` to also upload the backup to object storage.

```
"upload_backup_data_to_cloud_storage": {
    "cloud_storage_provider": "s3",
    "cloud_storage_credentials_id": 3,
    "bucket": "my-bucket/subpath",
    "backup_retention": 90,
    "delete_after_upload": false
}
```

| Field | Type | Values |
|---|---|---|
| `cloud_storag` | string | `"s3"`, `"gcs"`, `"azure"` |
| `cloud_storag` | integer | Credential ID from **Settings → Cloud storage credentials** |
| `bucket` | string | Bucket name, optionally with a sub-path |
| `backup_reter` | integer | Retention for the cloud copy. Same semantics as the top-level `backup_retention`: `-1` keeps it forever, `0` follows the cluster's default retention setting, and any other positive value is a custom retention in days |
| `delete_after` | boole | Remove the local copy after upload |

### mysqldump

```
{
    "backup_method": "mysqldump",
    "hostname": "192.168.40.54",
    "port": 3306,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "mysqldump_type": "Complete",
    "extended_insert": true,
    "include_databases": "",
    "use_pigz": false
}
```

| Field | Type | Values / Notes |
|---|---|---|
| mysql-dump_type | string | "Complete", "SchemaOnly", "DataOnly", "MySQLDbOnly", "SchemaAndData" |
| extended_insert | boolean | Use multi-row INSERT statements. Default true |
| include_databases | string | Comma-separated list of databases. Empty string backs up all databases |
| use_pigz | boolean | Use parallel gzip for faster compression |

### xtrabackupfull / xtrabackupincr

Use "backup_method": "xtrabackupfull" or "xtrabackupincr".

```
{
    "backup_method": "xtrabackupfull",
    "hostname": "192.168.40.54",
    "port": 3306,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "xtrabackup_backup_locks": true,
    "xtrabackup_parallellism": 1,
    "xtrabackup_lock_ddl_per_table": false,
    "use_pigz": false,
    "use_qpress": false,
    "throttle_rate_netbw": 0,
    "include_databases": "",
    "wsrep_desync": false
}
```

| Field | Type | Values / Notes |
|---|---|---|
| xtra-backup_b | boole | Use `LOCK TABLES FOR BACKUP`. Default `true` |
| xtra-backup_p | in-te-ger | Parallel copy threads. Default `1` |
| xtra-backup_l | boole | Lock DDL per table to avoid MDL conflicts |
| use_pigz | boole | Use parallel gzip for compression |
| use_qpre | boole | Use QPress compression (for older MariaDB versions) |
| throt-tle_rate | in-te-ger | Network throttle in MB/s. `0` means no throttle |
| in-clude_da | string | Comma-separated list for partial backup. Empty string backs up all databases |
| ws-rep_desy | boole | **Galera clusters only.** When `true`, sets `wsrep_desync=ON` on the backup node before the backup starts, so its large write-sets don't trigger flow control on the rest of the cluster. ClusterControl sets `wsrep_desync=OFF` again once the backup finishes and the node has caught up. Omit or leave `false` for non-Galera (replication) clusters |

### mariabackupfull / mariabackupincr

Use `"backup_method": "mariabackupfull"` or `"mariabackupincr"`. Field names are identical to xtrabackup.

```
{
    "backup_method": "mariabackupfull",
    "hostname": "192.168.40.52",
    "port": 3306,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "cloud_only": false,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "xtrabackup_backup_locks": true,
    "xtrabackup_parallellism": 1,
    "xtrabackup_lock_ddl_per_table": false,
    "use_pigz": false,
    "use_qpress": false,
    "throttle_rate_netbw": 0,
    "wsrep_desync": false
}
```

All xtrabackup field descriptions apply identically to mariabackup, including `wsrep_desync` for MariaDB Galera Cluster.

### pgdumpall

```
{
    "backup_method": "pgdumpall",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "schema_only": false,
    "data_only": false
}
```

| Field | Type | Notes |
|---|---|---|
| `schema_only` | boolean | Dump only the schema, not data |
| `data_only` | boolean | Dump only the data, not schema |

### pgdump

`pgdump` backs up one or more named databases. Unlike `pgdumpall`, it supports format selection and content filters.

```
{
    "backup_method": "pgdump",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "include_databases": "mydb",
    "backup_format": "custom",
    "schema_only": false,
    "data_only": false,
    "no_owner": false,
    "no_privileges": false
}
```

| Field | Type | Values / Notes |
|---|---|---|
| include_databases | string | Comma-separated list of databases to dump (required for pgdump) |
| backup_format | string | "plain", "custom", "directory", "tar" |
| schema_only | boolean | Dump only the schema |
| data_only | boolean | Dump only the data |
| no_owner | boolean | Omit ALTER OWNER statements |
| no_privileges | boolean | Omit GRANT and REVOKE statements |

### pg_basebackup

```
{
    "backup_method": "pg_basebackup",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false
}
```

pg_basebackup supports cloud upload. Include *upload_backup_data_to_cloud_storage* to specify the destination (provider, credentials, bucket); add "cloud_only": true alongside it to stream directly to S3-compatible storage instead of uploading after the backup completes locally.

Example: stream a pg_basebackup directly to S3 without staging it on local disk first.

### ClusterControl CLI

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackup \
    --cloud-only \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --wait \
    --log
```

--cloud-only cannot be combined with --backup-dir or --on-controller. Those flags stage the backup on local/controller storage, which conflicts with streaming it directly to cloud storage.

### ClusterControl RPC API

```
{
    "backup_method": "pg_basebackup",
    "hostname": "192.168.40.63",
    "port": 5432,
    "cloud_only": true,
    "upload_backup_data_to_cloud_storage": {
        "cloud_storage_provider": "s3",
        "cloud_storage_credentials_id": 2,
        "bucket": "my-aws-bucket-eu-west-1/postgresql"
    }
}
```

Omit `backupdir` and `cc_storage` when `cloud_only` is `true`; they specify local/controller storage, which isn't used for a direct-to-cloud stream.

### pg_basebackupincr

PostgreSQL 17 or later only. Requires `summarize_wal = on` on the backup source node. Enable it before creating the backup:

```
s9s node --configure-wal --summarize-wal=on \
    --nodes="<HOST>:<PORT>" \
    --cluster-id=<ID> \
    --wait --log
```

Each incremental backup chains from the most recent full `pg_basebackup` in the same directory. ClusterControl reconstructs the chain automatically using `pg_combinebackup` on restore.

```
{
    "backup_method": "pg_basebackupincr",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false
}
```

Like `pg_basebackup`, `pg_basebackupincr` supports cloud upload. Include *`upload_backup_data_to_cloud_storage`* to specify the destination; add `"cloud_only": true` alongside it to stream directly to S3-compatible storage without requiring local disk capacity for the full backup size.

Example: stream an incremental `pg_basebackupincr` directly to S3 without requiring local disk capacity for the full backup size.

### ClusterControl CLI

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackupincr \
    --cloud-only \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --wait \
    --log
```

`--cloud-only` cannot be combined with `--backup-dir` or `--on-controller`. Those flags stage the backup on local/controller storage, which conflicts with streaming it directly to cloud storage.

### ClusterControl RPC API

```
{
    "backup_method": "pg_basebackupincr",
    "hostname": "192.168.40.63",
    "port": 5432,
    "cloud_only": true,
    "upload_backup_data_to_cloud_storage": {
        "cloud_storage_provider": "s3",
        "cloud_storage_credentials_id": 2,
        "bucket": "my-aws-bucket-eu-west-1/postgresql"
    }
}
```

Omit `backupdir` and `cc_storage` when `cloud_only` is `true`; they specify local/controller storage, which isn't used for a direct-to-cloud stream.

### pgbackrestfull / pgbackrestdiff / pgbackrestincr

Use `"backup_method": "pgbackrestfull"`, `"pgbackrestdiff"`, or `"pgbackrestincr"`. pgbackrest manages its own repository so `backupdir`, `backupsubdir`, and `cc_storage` are not used.

```
{
    "backup_method": "pgbackrestfull",
    "hostname": "auto",
    "port": 5432,
    "compression": true,
    "compression_level": 6
}
```

`"hostname": "auto"` is recommended. pgbackrest selects the backup node based on its own configuration (standby preference and repository host settings).

### mongodump

```
{
    "backup_method": "mongodump",
    "hostname": "192.168.40.160",
    "port": 27017,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "backup_retention": 31
}
```

Compression is not configurable for `mongodump`. Specify a replica set member in `hostname`, not the primary.

### percona-backup-mongodb

PBM must be installed on all nodes before creating a backup. See *MongoDB*.

```
{
    "backup_method": "percona-backup-mongodb",
    "hostname": "192.168.40.170",
    "port": 27019,
    "backup_retention": 31
}
```

`hostname` is a config server replica set node. PBM manages its own backup directory; `backupdir` and `cc_storage` are not used.

### rdb (Redis & Valkey)

Used for all Redis and Valkey cluster types. Always specify `"backup_method": "rdb"` regardless of the underlying cluster type (Sentinel or Cluster mode). If AOF persistence is enabled on the instance, ClusterControl includes the AOF file in the backup automatically.

```
{
    "backup_method": "rdb",
    "hostname": "192.168.40.50",
    "port": 6379,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "backup_retention": 31
}
```

### mssqlfull / mssqldiff / mssqllog

Use `"backup_method": "mssqlfull"`, `"mssqldiff"`, or `"mssqllog"`.

```
{
    "backup_method": "mssqlfull",
```

(continues on next page)

(continued from previous page)

```
    "hostname": "auto",
    "backupdir": "/var/opt/mssql/data/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": false,
    "compression": true,
    "compression_level": -1,
    "backup_retention": 31,
    "backup_system_db": true
}
```

| Field | Type | Values / Notes |
|---|---|---|
| `hostname` | string | `"auto"` (recommended) lets ClusterControl target the current primary automatically |
| `backup_system` | boolean | Include system databases (`master`, `model`, `msdb`). Default `true` |
| `compression_level` | integer | `-1` uses SQL Server's built-in compression default |
| `cc_storage` | boolean | SQL Server backups are stored on the primary node; the controller cannot act as storage host. Use `false` |

`port` is not required. ClusterControl resolves the SQL Server port from the cluster configuration. `encrypt_backup` is not accepted for these methods; encryption is always applied natively by SQL Server (`AES_256` via a ClusterControl-managed server certificate) and cannot be disabled. For example, ClusterControl runs a statement equivalent to:

```
BACKUP DATABASE [myexampledb] TO DISK = '/
↪var/lib/backups/BACKUP-838222/myexampledb_full.bak'
WITH CHECKSUM, COMPRESSION, ENCRYPTION (ALGORITHM = AES_256, SERVER␣
↪CERTIFICATE = S9sDbmEncryptCert)
```

### elasticsearch-snapshot

```
{
    "backup_method": "elasticsearch-snapshot",
    "snapshot_repo_type": "fs-nfs",
    "snapshot_repository": "s9s-es-repo",
    "backup_retention": 31
}
```

| Field | Type | Values / Notes |
|---|---|---|
| `snapshot_repo_typ` | string | `"fs-nfs"` (NFS-backed filesystem), `"fs"` (local filesystem), `"s3"` (S3-compatible object storage) |
| `snapshot_reposito` | string | Repository name. Matches the value of `elastic_snapshot_repository` in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf` |

`hostname`, `backupdir`, and `cc_storage` are not used. Snapshot storage is configured at cluster setup time via the repository settings.

#### clickhouse-native / clickhouse-native-incr

Use `"backup_method": "clickhouse-native"` for a full backup or `"clickhouse-native-incr"` for an incremental backup.

```
{
    "backup_method": "clickhouse-native",
    "hostname": "10.10.15.11",
    "port": 9440,
    "compression": true,
    "backup_retention": 31,
    "include_databases": ""
}
```

| Field | Type | Notes |
|---|---|---|
| `include_dat` | string | Comma-separated list of databases to include. Empty string backs up all databases |
| `backupdir` | string | Optional. Omit to write to the `backups` disk registered on the backup node (`Disk('backups', ...)`). Set a custom path to have ClickHouse write there instead (`File(...)`) |

`cc_storage` is not used; controller storage is not supported for ClickHouse. Cloud upload uses the same `upload_backup_data_to_cloud_storage` object as other methods; see *Cloud upload*.

---

### 2.60.7 Advanced Example

The examples throughout this page each highlight a single option at a time. The example below combines many of them into one backup, to give you a sense of how the advanced CLI and RPC API options work together in practice.

The scenario: create a full xtrabackup for a MySQL Replication cluster with tuned compression and parallelism, DDL-per-table locking, network throttling, encryption, and cloud upload to AWS S3 with its own retention and automatic local cleanup.

#### ClusterControl CLI

```
s9s backup \
    --create \
    --cluster-id=99 \
    --nodes="192.168.40.60:3306" \
    --backup-method=xtrabackupfull \
    --backup-dir=/storage/backups \
    --subdirectory="BACKUP-%H-%I" \
    --on-controller \
    --title="Full xtrabackup with cloud upload" \
    --backup-retention=45 \
    --compression-level=9 \
    --use-pigz \
    --parallelism=4 \
```

(continues on next page)

(continued from previous page)

```
    --encrypt-backup \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --cloud-retention=90 \
    --delete-after-upload \
    --wait \
    --log
```

- `--title` labels the backup so it is easy to identify later in `--list` output.
- `--compression-level=9` with `--use-pigz` maximizes compression while using parallel gzip to offset the added CPU cost on large datasets.
- `--parallelism=4` uses 4 threads during the xtrabackup copy phase.
- `--encrypt-backup` enables AES-256 encryption; ClusterControl auto-generates and stores the key.
- `--cloud-provider`, `--s3-bucket`, and `--credential-id` upload the completed backup to AWS S3.
- `--cloud-retention=90` sets a retention period for the cloud copy that is independent of `--backup-retention`.
- `--delete-after-upload` removes the local copy once the cloud upload succeeds, reclaiming disk space on the backup host.

### ClusterControl RPC API

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 99,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "backup",
            "job_data": {
                "backup_method": "xtrabackupfull",
                "hostname": "192.168.40.60",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%H-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 9,
                "backup_retention": 45,
                "encrypt_backup": true,
                "xtrabackup_backup_locks": true,
                "xtrabackup_parallellism": 4,
                "xtrabackup_lock_ddl_per_table": true,
```

(continues on next page)

(continued from previous page)

```
                "use_pigz": true,
                "throttle_rate_netbw": 50,
                "upload_backup_data_to_cloud_storage": {
                    "cloud_storage_provider": "s3",
                    "cloud_storage_credentials_id": 2,
                    "bucket": "my-aws-bucket-eu-west-1/mysql-replication",
                    "backup_retention": 90,
                    "delete_after_upload": true
                }
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

- `xtrabackup_lock_ddl_per_table: true` avoids long DDL locks on heavily loaded servers. See *MySQL & MariaDB* for background.
- `throttle_rate_netbw: 50` caps network streaming at 50 MB/s to avoid saturating the link during business hours.
- `upload_backup_data_to_cloud_storage` uploads the completed local backup to S3 with its own 90-day retention, then deletes the local copy. See *Cloud upload* for all supported fields.

## 2.61 Schedule Backup

The **Schedule Backup** feature in ClusterControl allows you to automate regular database backups and backup verification to ensure data protection and recovery readiness. By configuring scheduled backups, you can define backup frequency, retention policies, storage locations, and backup methods according to your operational requirements.

Compared to the *Create Backup* feature, **Schedule Backup** provides additional capabilities, including:

- Backup verification to validate backup integrity and ensure recoverability.
- Backup failover, which automatically performs backups on an alternative database node if the selected node is unavailable.
- Binary log (binlog) backup support for MySQL and MariaDB databases.
- Flexible scheduling with timezone support to accommodate different deployment locations and maintenance windows.
- Pausing and resuming backup schedules.

### 2.61.1 Supported backup methods

For schedule backup, ClusterControl supports the following backup methods:

| Cluster type | Supported backup methods |
|---|---|
| MySQL-based clusters | mysqldumpPercona Xtrabackup (full, incremental)Binlog backup |
| MariaDB-based clusters | mariadb-dumpMariaDB Backup (full, incremental)Binlog backup |
| PostgreSQL-based clusters | pg_dumppg_dumpallpg_basebackup (full, incremental - only for v17 and later)pgBackRest (full, incremental, differential) |
| MongoDB-based clusters | mongodumpPercona Backup for MongoDB |
| Redis-based clusters | RDB (includes AOF file if AOF persistence is enabled) |
| SQL Server | FullDifferentialTransaction log |
| Elasticsearch | Elasticsearch snapshot |
| ClickHouse-based clusters | ClickHouse native backup (full, incremental) |

**Info**

Binlog backup is only available for scheduled backups, and the backup destination is limited to cloud storage.

For more details on each of the supported backup methods, see *Backup methods*.

### 2.61.2 Prerequisites

Before configuring scheduled backups in ClusterControl, ensure that your environment meets the necessary requirements:

- The database cluster must be managed by ClusterControl.
- Only backup methods supported by ClusterControl can be used. See *Supported backup methods*.
- Ensure that the required backup tool binaries are installed and accessible on the target nodes.
- Backups created and scheduled in ClusterControl can be stored either on the ClusterControl host or on the target database nodes. If you use an external backup storage host, you can mount it via NFS and share the volume for backup storage.
- If the backup destination is the ClusterControl host, ensure that port 9999 (or within the range 9990-9999) is open from the database node to the ClusterControl host for backup streaming. These ports are configurable under ***Backup settings → Netcat port***.
- Some backup methods support storing backups in cloud storage. Configure cloud credentials under ***ClusterControl GUI → Settings → Cloud storage credentials*** beforehand if you plan to use this feature.
- Ensure that system clocks are synchronized (e.g., via NTP) across all nodes.

**Tip**

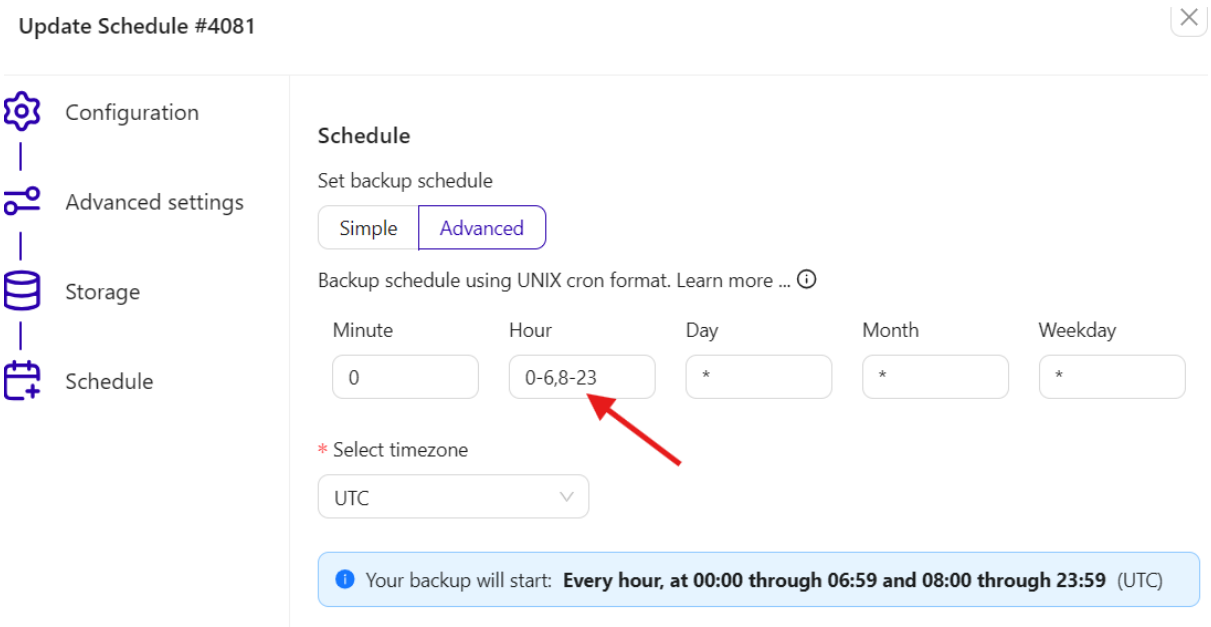
Configure email or alerting integrations to receive backup status notifications. See *Notification Service*.

### 2.61.3 Backup jobs & scheduling

The scheduled backup job in ClusterControl runs as a background thread (RUNNING3) and does not block other non-backup jobs in the queue. Even if a backup takes hours to complete, other non-backup jobs can continue to run simultaneously via the main thread (RUNNING). You can monitor job progress under ***ClusterControl GUI → Activity center → Jobs***.

ClusterControl creates a backup user (prefixed with `backupuser`) if it does not already exist. If this user encounters issues accessing the database server, it will be reset based on the `backup_user` and `backup_user_password` values defined in the CMON configuration file.

Note that ClusterControl executes **only one backup job per cluster at a time**. If multiple scheduled backups overlap, subsequent jobs will wait until the active backup completes. For example, if a full backup is scheduled at 7:00 AM and incremental backups run hourly, the 7:00 AM incremental backup will most likely wait for the full backup to finish before starting. To avoid this, you can exclude specific hours using the scheduler's advanced editor, which follows the UNIX cron format, as shown in the example below:




### 2.61.4 Common Wizard Steps

The schedule backup wizard follows the same flow and step order for every database type, though a few steps are skipped or restricted for certain backup methods (noted inline where that applies). Only the **Configuration** and **Advanced Settings** steps have database-specific options, which are documented in *Database-Specific Settings* below.

### ClusterControl GUI

**Opening the wizard**

1. Go to the **Clusters** dashboard and click your target cluster.
2. Go to the **Backups** tab and click **Create backup**.
3. Choose **Schedule a Backup**.

---

**Step 1: Configuration**

- **Schedule name**: A label to identify this scheduled backup job in the backup list.
- **Cluster**: Pre-set to the cluster you selected.
- **Backup host**: Choose the node to run the backup from, or select **Auto Select** to let ClusterControl pick the least-loaded node. Not available for MongoDB Replica Set (see *MongoDB* below).
- **Backup method**: Available methods vary by database type; see *Database-Specific Settings*.
- **Upload backup to cloud**: Defaults to **Off**. Set to **On** if you have cloud credentials configured and want to upload the backup.
- **Stream backup to the cloud**: Only appears when **Upload backup to cloud** is **On**. Supported for `xtrabackupfull`, `xtrabackupincr`, `mariabackupfull`, `mariabackupincr`, `pg_basebackup`, and `pg_basebackupincr`. When **On**, the backup is written directly to cloud storage without being staged locally first, and the Local Storage step is skipped. When **Off**, the backup is stored locally first and then uploaded; you can optionally delete the local copy after upload in the **Cloud Storage** step.

Click **Continue**.

---

**Step 2: Advanced Settings**

The following fields are common across most backup methods. Additional method-specific fields are listed per database type in *Database-Specific Settings*.

- **Compression**: Defaults to **On**. When off, backup files are stored uncompressed.
- **Retention**: Defaults to `31 days`. Adjust the number of days, or set to **Off** to disable automatic deletion.
- **Compression level**: Defaults to `6 (System default)`.
- **Enable encryption**: Defaults to **Off**. When enabled, ClusterControl auto-generates an encryption key stored in `/etc/cmon.d/cmon_<cluster_id>.cnf`.

Click **Continue**.

---

**Step 3: Verify Backup**

!!! note “Backup verification support”

```
Backup verification is not limited to one backup method. It is available␣
↪for MySQL-based, MariaDB-based, PostgreSQL-based, and ClickHouse-based␣
↪clusters (all listed backup methods), as well as Redis-based clusters. It␣
↪is not available for MongoDB-based clusters, SQL Server, or␣
↪Elasticsearch. See [Backup Verification](./backup-verification.md) for␣
↪setup requirements and per-method restore steps.
```

The **Verify backup** section is only visible if **Verify backup** is enabled in the **Advanced settings** section. Here, you can configure the following options:

- **Restore backup on**: Specify the FQDN, hostname or IP address of the standalone host. The host must not be part of the cluster. Press ++enter++ to add the node, where ClusterControl will perform a pre-deployment check to verify if the node is reachable via SSH key-based authentication.
- **Install database software**: Toggle **On** to instruct ClusterControl to install a new MySQL or MariaDB server during restoration. If a MySQL server is already installed or running, it will be stopped and removed before ClusterControl reinstalls it. If toggled **Off**, ClusterControl will skip the database installation step and proceed directly with the backup verification process.
- **Disable firewall**: Check the box to disable the firewall (recommended).
- **Disable SELinux/AppArmor**: Check the box to disable SELinux or AppArmor during the database deployment phase.
- **Shut down the server after backup restored**: Toggle **On** to shut down the server after the restoration completes. Toggle **Off** to keep the server running after restoration. The node will then be listed under ***ClusterControl GUI → Nodes*** as *Backup Verification Server (BVS)*. You will be responsible for removing the database server afterward.
- **Start backup verification (after completion) in**: Specify how many hours to wait before starting the backup verification job after the backup has completed successfully. Verification can be scheduled up to 24 hours after completion.

Click **Continue**.

---

#### Step 4: Local Storage

> **Note**
>
> Note pgbackrest, MongoDB Sharded Cluster, and Elasticsearch backups skip this step. When **Stream backup to the cloud** is **On** in Step 1, this step is also skipped for the methods that support it.

- **Storage location**: Choose **Store on Controller** to store the backup inside the ClusterControl node. This requires `socat` or `netcat` on the source and destination host. By default, ClusterControl uses port 9999 to stream the backup created on the database node to the ClusterControl node. This is configurable under ***Backup settings → Netcat port***. Choose **Store on Node** to store the backup inside the target database node.
- **Storage directory**: You can opt to use another backup directory as you wish. If you leave this field blank, ClusterControl will use the default backup directory specified in the ***Backup settings → Default directory***.
- **Backup subdirectory**: Defaults to `BACKUP-%I`, where `%I` is the numeric backup job ID assigned to each run. See *Backup subdirectory*.

Click **Continue**.

---

#### Step 5: Cloud Storage

!!! note "Cloud storage support"

```
While pgBackRest and Elasticsearch do not support cloud storage backups,␣
↪MySQL & MariaDB binlog backups strictly require it.
```

This step only appears when **Upload backup to cloud** is **On** in Step 1. Otherwise, this step is skipped.

- **Credentials**: Select from credentials created in **Settings → Cloud storage credentials**. Supported providers: *Amazon Web Services*, *Google Cloud*, *Microsoft Azure*, *S3 Compatible storage provider*.
- **Bucket**: Your target storage bucket. Click **+** to create a new bucket, then confirm with **OK**. Only buckets permitted by the selected credentials will appear in the dropdown.
- **Location path**: Sub-path within the bucket for this backup. You can define multiple directory levels relative to the bucket root. Do not include a trailing slash. For example, if set to `server1/fullbackup/weekly`, the backup will be stored under `server1/fullbackup/weekly/BACKUP-151`, where `BACKUP-151` is derived from the **Backup subdirectory** value.
- **Retention period**: Defaults to **On**. Set the number of days to keep cloud backups, or **Off** to retain indefinitely.
- **Delete after upload**: Defaults to **Off**. Set to **On** to remove the local backup copy after a successful cloud upload, reclaiming local disk space. The backup will still appear in the backup list and can be restored as usual. During restoration, ClusterControl will download the backup from the cloud.

Click **Continue**.

---

#### Step 6: Schedule

This step is unique to scheduled backups. Choose between the **Simple** and **Advanced** tabs:

=== "Simple"

```
:::{tip}

The configuration defined here translates to the same output as the␣
↪**Advanced** editor. You can start with the **Simple** editor and switch␣
↪to the **Advanced** tab to fine-tune the schedule.
:::
* Configure the schedule periodically by minute (default). Other options␣
↪include hour, day, week, month, and year.
* Each option (except minute) provides additional fields to refine the␣
↪schedule. For example, selecting hour allows you to specify the exact␣
↪hour and minute for the backup.
```

=== "Advanced"

```
* You will be presented with a cron editor. You can set backup schedule␣
↪using UNIX cron format: `* * * * *`. Each field represents:
```

(continues on next page)

(continued from previous page)

```
    - minute (0-59)
    - hour (0-23)
    - day of the month (1-31)
    - month (1-12)
    - day of the week (0-7, with 0 and 7 representing Sunday).

* Use `*` for any value.
* Step values are also supported. For example to run the backup every other␣
↪hour, set the hour field to `*/2`, which is equivalent to␣
↪`0,2,4,6,8,10,12,14,16,18,20,22`.

- **Select timezone**: Choose the appropriate timezone, especially if the␣
↪server is in a different timezone. This is important to ensure backups do␣
↪not run during peak hours.
```

Click **Continue**.

---

#### Step 7: Preview

Review all selected options. This is the final confirmation before ClusterControl saves the schedule.

### ClusterControl CLI

Scheduled backups via the CLI use `s9s backup --create` with the `--recurrence` flag instead of `--wait` and `--log`. The `--recurrence` option accepts a standard cron expression and makes the job repeat on that schedule.

- Cron format: `'MINUTE HOUR DAY MONTH WEEKDAY'`
- Example: `'0 2 * * *'` runs daily at 2:00 AM
- Example: `'0 2 * * 0'` runs every Sunday at 2:00 AM

Specific commands with `--recurrence` are listed in each database section below.

**Manage existing schedules:**

- List all backup schedules for a cluster:

```
s9s backup \
    --list-schedules \
    --cluster-id=<ID>
```

- Delete a backup schedule by its job ID:

```
s9s backup \
    --delete-schedules \
    --job-id=<SCHEDULE_JOB_ID>
```

  To find the job ID, run `--list-schedules` first and note the ID column.

### ClusterControl RPC API

All RPC API calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`. All calls require a TLS connection and, in most cases, an authenticated session or inline credentials. See *ClusterControl RPC API → Authenticating with the API* for details.

- Create a backup schedule using the `scheduleBackup` operation. The `schedule.schedule` field is a standard cron expression. The `job_data` fields differ by backup method; see the *job_data reference* below and each database section for method-specific examples.

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 1,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mysqldump",
                "hostname": "192.168.40.54",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

- List all backup schedules:

```
curl -k -XPOST \
-d '{
    "operation": "getBackupSchedules"
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

  The response includes a `backup_schedules` array. Each entry has an `id` field that identifies the schedule.

### 2.61.5 Database-Specific Settings

Each section below covers the supported backup methods for that database type, any **Advanced Settings** fields unique to each method beyond the common fields above, and CLI and RPC API examples.

#### MySQL & MariaDB

Covers MySQL & Percona & MariaDB standalone and primary-replica replication clusters, as well as Galera Cluster topologies (Percona XtraDB Cluster, MariaDB Galera Cluster). The backup methods, Advanced Settings, and job_data fields are identical across topologies; only the cluster/node you target differs.

**Backup methods:**

| Method | Type | Cloud upload | Notes |
|---|---|---|---|
| `mysqldump` | Logical | Yes | Dump types: *Schema only*, *MySQL DB Only*, *Data Only*, *Schema And Data*, *Complete PITR-compatible* |
| `xtrabackupfull` | Physical | Yes | Percona Server and Oracle MySQL clusters |
| `xtrabackupincr` | Physical | Yes | Percona Server and Oracle MySQL clusters |
| `mariabackupfull` | Physical | Yes | MariaDB clusters |
| `mariabackupincr` | Physical | Yes | MariaDB clusters |

#### ClusterControl GUI

For Galera clusters (Percona XtraDB Cluster, MariaDB Galera Cluster), select your Galera cluster in the **Cluster** field. All Advanced Settings below apply identically regardless of topology.

**mysqldump - unique Advanced Settings**

- **Enable partial backup**: When **On**, specify individual databases to include instead of backing up all databases.
- **Use extended insert**: Defaults to **On**. When off, `mysqldump` generates single-row `INSERT` statements.
- **One dump file per DB**: Defaults to **Off**. When on, `mysqldump` writes a separate backup file per database.

**xtrabackup / mariabackup - unique Advanced Settings**

These settings apply to both full and incremental variants. Select the specific variant in the **Backup method** field (`xtrabackup (full)`, `xtrabackup (incremental)`, `mariabackup (full)`, or `mariabackup (incremental)`).

- **Enable partial backup**: When **On**, specify individual databases to include.
- **Use PIGZ for parallel gzip**: Defaults to **Off**. Enables parallel compression, which significantly speeds up backups of large datasets.

- **Use ZSTD compression**: Defaults to **Off**. Uses Zstandard compression. For MySQL & Percona Server versions < 8.0.34, QPress is used instead.
- **Backup locks**: Defaults to **On**. When on, xtrabackup uses `LOCK TABLES FOR BACKUP`. When off (`--no-backup-locks`), uses `FLUSH NO_WRITE_TO_BINLOG TABLES` and `FLUSH TABLES WITH READ LOCK`.
- **Lock DDL per table**: Defaults to **Off**. Sets `--lock-ddl-per-table`. Enable if you encounter errors like `InnoDB: An optimized (without redo logging) DDL operation has been performed`. See Percona blog, MDEV-14095, MDEV-5336. May cause update delays on heavily loaded servers.
- **Xtrabackup parallel copy threads**: Defaults to `1`. Increase to speed up streaming during the copy phase.
- **Network streaming throttle rate (MB/s)**: Defaults to `0` (no throttle). Uses the `pv` tool when a non-zero value is set.
- **Desync node during backup** (Galera clusters only): Defaults to **Off**. When **On**, sets `wsrep_desync=ON` on the backup node before the backup starts, so it can fall behind without triggering flow control on the rest of the cluster. The node catches up and `wsrep_desync` is set back to **Off** once the backup finishes. Not shown for non-Galera (replication) clusters.

**Backup failover (all methods)**

- **Failover backup if node is down**: Defaults to **Off**. When **On**, ClusterControl automatically retries the backup on another eligible node if the selected node is unreachable when the job starts, instead of failing the job.

**Backup verification (all methods)**

- **Enable backup verification**: Defaults to **Off**. When **On**, additional fields appear to configure the Backup Verification Server (BVS) that ClusterControl restores the backup to after it completes, to confirm it is recoverable. Available for `mysqldump`, `xtrabackup`, and `mariabackup`. See *Backup Verification* for BVS setup requirements and field descriptions.

### ClusterControl CLI

- Schedule a backup using mysqldump, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=127 \
    --nodes="192.168.40.54:3306" \
    --backup-method=mysqldump \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 1 * * *'
```

- Schedule a mysqldump backup with automatic verification on a standalone Backup Verification Server, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=127 \
    --nodes="192.168.40.54:3306" \
```

(continues on next page)

(continued from previous page)

```
    --backup-method=mysqldump \
    --backup-dir=/storage/backups \
    --on-controller \
    --test-server=192.168.73.88 \
    --recurrence='0 1 * * *'
```

`--test-server` triggers a follow-up verify job against the given host after each scheduled backup completes.

- Schedule a full xtrabackup, running daily at 2:00 AM:

```
s9s backup \
    --create \
    --cluster-id=99 \
    --nodes="192.168.40.60:3306" \
    --backup-method=xtrabackupfull \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 2 * * *'
```

Use `xtrabackupincr` for incremental backups.

- Schedule a full mariabackup, running daily at 2:00 AM:

```
s9s backup \
    --create \
    --cluster-id=126 \
    --nodes="192.168.40.52:3306" \
    --backup-method=mariabackupfull \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 2 * * *'
```

Use `mariabackupincr` for incremental backups.

- Schedule a mysqldump backup for a Galera cluster (Percona XtraDB Cluster), running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=124 \
    --nodes="192.168.40.67:3306" \
    --backup-method=mysqldump \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 1 * * *'
```

The same commands used above for replication clusters apply to Galera clusters (`xtrabackupfull`/`xtrabackupincr`, `mariabackupfull`/`mariabackupincr`). Substitute your Galera cluster ID and a Galera node in `--nodes`. **Desync node during backup** has no `s9s backup` CLI flag yet; use the GUI or RPC API to enable it for `xtrabackupfull`/`xtrabackupincr`/`mariabackupfull`/`mariabackupincr` on Galera clusters.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`.

- Schedule a mysqldump backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 127,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mysqldump",
                "hostname": "192.168.40.54",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "mysqldump_type": "Complete",
                "extended_insert": true
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

  See `mysqldump_type` values and all available fields in the *job_data reference*.

- Schedule a mysqldump backup with automatic verification on a standalone Backup Verification Server, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 127,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mysqldump",
                "hostname": "192.168.40.54",
```

(continues on next page)

(continued from previous page)

```
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "mysqldump_type": "Complete",
                "extended_insert": true,
                "verify_backup_delay": 0,
                "verify_backup": {
                    "server_address": "192.168.73.88",
                    "install_software": true,
                    "disable_firewall": true,
                    "disable_selinux": true,
                    "terminate_db_server": false
                }
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

See *Backup verification* for all `verify_backup` fields. The same fields apply to `xtrabackupfull`, `xtrabackupincr`, `mariabackupfull`, and `mariabackupincr`.

- Schedule a full xtrabackup, running daily at 2:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 99,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "xtrabackupfull",
                "hostname": "192.168.40.60",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "xtrabackup_backup_locks": true,
```

(continues on next page)

(continued from previous page)

```
                "xtrabackup_parallellism": 1
            }
        },
        "schedule": "0 2 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

Use `"backup_method": "xtrabackupincr"` for incremental, `"mariabackupfull"` or `"mariabackupincr"` for MariaDB clusters. See all available fields in the *job_data reference*.

- Schedule a mysqldump backup for a Galera cluster (Percona XtraDB Cluster), running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 124,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mysqldump",
                "hostname": "192.168.40.67",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "mysqldump_type": "Complete"
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

The same `job_data` fields used above for replication clusters apply to Galera clusters. Set `cluster_id` and `hostname` to your Galera cluster and node.

- Schedule a full xtrabackup for a Galera cluster with the node desynced during the backup, running daily at 2:00 AM:

```
curl -k -XPOST \
-d '{
```

(continues on next page)

(continued from previous page)

```
    "operation": "scheduleBackup",
    "cluster_id": 124,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "xtrabackupfull",
                "hostname": "192.168.40.67",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "xtrabackup_backup_locks": true,
                "xtrabackup_parallellism": 1,
                "wsrep_desync": true
            }
        },
        "schedule": "0 2 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

`wsrep_desync` only applies to Galera clusters using `xtrabackupfull`/`xtrabackupincr` or `mariabackupfull`/`mariabackupincr`. Omit it (or set `false`) for replication clusters or for `mysqldump`. See *xtrabackupfull / xtrabackupincr* for the full field reference.

**Common steps**

All other wizard steps (Local Storage, Cloud Storage, Schedule, Preview) follow the *Common Wizard Steps* above.

---

### PostgreSQL & TimescaleDB

**Backup methods:**

| Method | Type | Cloud upload | Notes |
|---|---|---|---|
| `pgdumpall` | Logical | Yes | Dumps all databases in the cluster |
| `pgdump` | Logical | Yes | Dumps one or more named databases with format and content options |
| `pg_basebackup` | Physical | Yes | PITR disabled if pgbackrest manages WAL |
| `pg_basebackup` | Physical | Yes | Incremental physical backup. PostgreSQL 17+ only; requires `summarize_wal = on` |
| `pgbackrest-full` | Physical | Yes | Full backup |
| `pgbackrestd-iff` | Physical | Yes | Differential backup |
| `pgbackrestincr` | Physical | Yes | Incremental backup |

**Info**

`pg_basebackup` and `pg_basebackupincr` both support cloud upload (S3-compatible storage). Enable **Stream backup to the cloud** to write the backup directly to cloud storage as it runs, instead of staging it on local disk first.

### ClusterControl GUI

**pgdumpall - unique Advanced Settings**

- **Enable partial backup**: When **On**, specify individual databases to back up instead of the full cluster.

**pgdump - unique Advanced Settings**

`pgdump` backs up one or more named databases and provides additional format and content control compared to `pgdumpall`.

- **Include databases**: Specify the database names to include (required).
- **Backup format**: `Plain`, `Custom`, `Directory`, or `Tar`.
- **Schema only**: When **On**, only dumps the schema (no data).
- **Data only**: When **On**, only dumps the data (no schema).

- **No owner**: When **On**, omits `ALTER OWNER` statements.
- **No privileges**: When **On**, omits `GRANT` and `REVOKE` statements.

**pg_basebackup - unique Advanced Settings**

No method-specific fields beyond the common Advanced Settings (Compression, Retention, Compression level, Enable encryption). To upload to cloud storage, set **Upload backup to cloud** to **On** in Step 1; turn on **Stream backup to the cloud** to stream directly to S3-compatible storage without staging locally first.

**pg_basebackupincr - prerequisites and unique settings**

`pg_basebackupincr` requires PostgreSQL 17 or later and `summarize_wal = on` on the backup source node. The wizard checks this automatically and warns if the setting is off.

Before creating the first incremental schedule, enable WAL summarization on the node:

1. In the cluster node list, click the action menu (**...**) on the target node.
2. Select **Configure WAL**.
3. Set **Summarize WAL** to **On** and confirm.

After WAL summarization is enabled, `pg_basebackupincr` appears as a selectable backup method. Incremental backups chain from the last full `pg_basebackup`. When restoring, ClusterControl automatically reconstructs the full chain using `pg_combinebackup`.

**pgbackrest (full, diff, incr) - installation and unique settings**

!!! note "pgbackrest installation"

```
If pgbackrest is not yet installed, the Configuration step shows an␣
↪installation panel before proceeding. Choose a **Configuration method**:

- **Primary**: Install on the primary node only. The backup repository is␣
↪created on the primary. No SSH configuration between hosts.
- **All DB nodes**: Install on all database hosts. The backup repository is␣
↪on the primary; backups use a standby node. SSH is configured between␣
↪hosts.
- **All DB nodes and dedicated repository host**: Install on all DB nodes␣
↪and a separate repository host. Specify the **Dedicated repository host**␣
↪(IP, hostname, or FQDN).

Click **Continue** to install and verify the job completes successfully␣
↪before proceeding.
```

pgbackrest backups skip the **Local Storage** step. Backup data is managed by pgbackrest's own repository configuration.

The Advanced Settings for pgbackrest are limited to: Compression, Compression level, and Enable encryption.

**Backup failover (`pgdumpall`, `pgdump`, `pg_basebackup`, `pg_basebackupincr` only)**

- **Failover backup if node is down**: Defaults to **Off**. When **On**, ClusterControl automatically retries the backup on another eligible node if the selected node is unreachable when the job starts, instead of failing the job. Not available for `pgbackrest`, which selects its backup node from its own repository configuration instead.

**Backup verification (all methods)**

- **Enable backup verification**: Defaults to **Off**. When **On**, additional fields appear to configure the Backup Verification Server (BVS) that ClusterControl restores the backup to after it completes, to confirm it is recoverable. Available for `pgdumpall`, `pgdump`, `pg_basebackup`, and `pgbackrest` (full, differential, incremental). See *Backup Verification* for BVS setup requirements and field descriptions.

### ClusterControl CLI

- Schedule a pgdumpall backup, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pgdumpall \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 1 * * *'
```

- Schedule a pgdump backup of a specific database, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pgdump \
    --databases=mydb \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 1 * * *'
```

- Schedule a pgdumpall backup with automatic verification on a standalone Backup Verification Server, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pgdumpall \
    --backup-dir=/storage/backups \
    --on-controller \
    --test-server=192.168.73.90 \
    --recurrence='0 1 * * *'
```

`--test-server` also applies when scheduling `pgdump`, `pg_basebackup`, and `pgbackrest-full`/`pgbackrestdiff`/`pgbackrestincr`.

- Schedule a pg_basebackup, running daily at 1:00 AM:

```
s9s backup \
    --create \
```

(continues on next page)

(continued from previous page)

```
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackup \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 1 * * *'
```

To upload to cloud storage, add the same `--cloud-provider`, `--s3-bucket`, and `--credential-id` flags shown in the *xtrabackup cloud upload example*. Add `--cloud-only` to stream the backup directly to cloud storage without staging a local copy first; when using `--cloud-only`, remove `--backup-dir` and `--on-controller`, since those stage the backup on local/controller storage, which conflicts with streaming it directly to cloud storage.

- Enable WAL summarization on a node, then schedule an incremental pg_basebackup (PostgreSQL 17+ only):

```
# Step 1: enable summarize_wal on the target node (one-time setup)
s9s node \
    --configure-wal \
    --summarize-wal=on \
    --nodes="192.168.40.63:5432" \
    --cluster-id=125 \
    --wait \
    --log

# Step 2: schedule the incremental backup, running every 6 hours
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackupincr \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 */6 * * *'
```

`--summarize-wal=on` must be set before the first incremental backup run. Each incremental backup chains from the most recent full `pg_basebackup`. Restore is automatic: ClusterControl reconstructs the chain with `pg_combinebackup`.

- Schedule a pgbackrest full backup, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pgbackrestfull \
    --recurrence='0 1 * * *'
```

pgbackrest manages its own repository; `--backup-dir` and `--on-controller` are not used. Use `pgbackrestdiff` or `pgbackrestincr` for differential and incremental backups.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`.

- Schedule a pgdumpall backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 125,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "pgdumpall",
                "hostname": "192.168.40.63",
                "port": 5432,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "schema_only": false,
                "data_only": false
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

- Schedule a pgdumpall backup with automatic verification on a standalone Backup Verification Server, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 125,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "pgdumpall",
                "hostname": "192.168.40.63",
                "port": 5432,
                "backupdir": "/storage/backups",
```

(continues on next page)

(continued from previous page)

```
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "schema_only": false,
                "data_only": false,
                "verify_backup_delay": 0,
                "verify_backup": {
                    "server_address": "192.168.73.90",
                    "install_software": true,
                    "disable_firewall": true,
                    "disable_selinux": true,
                    "terminate_db_server": false
                }
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

See *Backup verification* for all `verify_backup` fields. The same fields apply to `pgdump`, `pg_basebackup`, and `pgbackrestfull`/`pgbackrestdiff`/`pgbackrestincr`.

- Schedule a pgdump backup of a specific database, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 125,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "pgdump",
                "hostname": "192.168.40.63",
                "port": 5432,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31,
                "include_databases": "mydb",
                "backup_format": "custom",
                "schema_only": false,
```

(continues on next page)

(continued from previous page)

```
                "data_only": false
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

See `no_owner` and `no_privileges` in the *job_data reference*. `pgdump` also supports the `verify_backup` fields described in *Backup verification*.

- Schedule a pgbackrest full backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 125,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "pgbackrestfull",
                "hostname": "auto",
                "port": 5432,
                "compression": true,
                "compression_level": 6
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

pgbackrest does not use `backupdir` or `cc_storage`. Use `"backup_method": "pgbackrestdiff"` or `"pgbackrestincr"` for differential and incremental.

- Schedule an incremental pg_basebackup (PostgreSQL 17+ only; requires `summarize_wal = on`):

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 125,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
```

(continues on next page)

(continued from previous page)

```
            "job_data": {
                "backup_method": "pg_basebackupincr",
                "hostname": "192.168.40.63",
                "port": 5432,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 6,
                "backup_retention": 31
            }
        },
        "schedule": "0 */6 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

Enable WAL summarization on the node before scheduling (see the ClusterControl CLI tab or the GUI Configure WAL action). Cloud upload to S3-compatible storage is supported for `pg_basebackupincr`.

**Common steps**

The Schedule and Preview steps follow the *Common Wizard Steps* above.

---

## MongoDB

MongoDB Replica Set and Sharded Cluster topologies each use their own backup method. Unlike MySQL & MariaDB, `mongodump` and `percona-backup-mongodb` are not interchangeable between topologies.

**Backup methods:**

| Method | Type | Cloud upload | Topology | Notes |
|---|---|---|---|---|
| `mongodump` | Logical | Yes | Replica Set | Backup always runs on a replica node; the **Backup host** field is not shown |
| `percona-backup-mor` (PBM) | Physical | No | Sharded Cluster | Requires MongoDB 4.0 or later and the PBM agent installed on every node |

### ClusterControl GUI

**mongodump (Replica Set)**

Only **Retention** and **Enable encryption** are configurable in Advanced Settings. Compression settings do not apply to `mongodump`.

**percona-backup-mongodb (Sharded Cluster)**

!!! note “PBM installation”

```
If PBM is not installed, the Configuration step shows an installation␣
↪panel. Specify the **Backup directory** (the path must already exist on␣
↪the filesystem), then click **Install**. Verify the installation job␣
↪completes before continuing.
```

Only **Retention** is configurable (defaults to `31 days`). No compression or encryption options are available. This backup type skips the **Local Storage** and **Cloud Storage** steps.

### ClusterControl CLI

**mongodump (Replica Set)**

- Schedule a mongodump backup, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=79 \
    --nodes="192.168.40.160:27017" \
    --backup-method=mongodump \
    --backup-dir=/storage/backups \
    --on-controller \
    --recurrence='0 1 * * *'
```

`--nodes="192.168.40.160:27017"` specifies the replica set member and its MongoDB port.

**percona-backup-mongodb (Sharded Cluster)**

Before scheduling, install the PBM agents on all nodes if not already present:

```
s9s cluster \
    --add-node \
    --nodes="pbmagent://*?backup_dir=/opt/pbm/backups/" \
    --cluster-id=28 \
    --wait \
    --log
```

`--nodes="pbmagent://*?backup_dir=/opt/pbm/backups/"` targets all nodes. Change `/opt/pbm/backups/` to your preferred directory.

After PBM is installed, schedule the backup:

- Schedule a percona-backup-mongodb backup, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=28 \
    --nodes="192.168.40.170:27019" \
    --backup-method=percona-backup-mongodb \
    --backup-retention=31 \
    --recurrence='0 1 * * *'
```

`--nodes="192.168.40.170:27019"` specifies a config server replica set node and its port.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`.

**mongodump (Replica Set)**

- Schedule a mongodump backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 79,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mongodump",
                "hostname": "192.168.40.160",
                "port": 27017,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "backup_retention": 31
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

**percona-backup-mongodb (Sharded Cluster)**

Install PBM agents via the CLI before using the RPC API to schedule the backup (see the ClusterControl CLI tab above).

- Schedule a percona-backup-mongodb backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 28,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "percona-backup-mongodb",
                "hostname": "192.168.40.170",
                "port": 27019,
```

(continues on next page)

(continued from previous page)

```
                "backup_retention": 31
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

**Common steps**

For `mongodump`, Local Storage, Cloud Storage, Schedule, and Preview follow the *Common Wizard Steps* above. For `percona-backup-mongodb`, only Schedule and Preview apply. Local Storage and Cloud Storage are skipped.

---

### Redis & Valkey

Applies to Redis and Valkey, in both Sentinel and Cluster topologies.

**Backup methods:**

| Method | Type | Cloud upload |
|---|---|---|
| `rdb` | Physical | Yes |

Backups are transferred from the cluster to the ClusterControl controller using `socat` over port `9999`. The port can be changed in **Backup Settings**.

**Note**

Note If AOF (Append-Only File) persistence is enabled on the Redis or Valkey instance, ClusterControl automatically includes the AOF file in the backup.

### ClusterControl GUI

**Backup host**: For Redis Sentinel clusters, defaults to the primary node; you can choose a replica instead.

**Unique Advanced Settings:**

- **Use PIGZ for parallel gzip**: Defaults to **Off**. Enables parallel compression for faster backup of large datasets.
- **Enable backup verification**: Defaults to **Off**. Available for Redis & Valkey Sentinel only. When **On**, additional fields appear to configure the Backup Verification Server (BVS) that ClusterControl restores the backup to after it completes, to confirm it is recoverable. See *Backup Verification* for BVS setup requirements and field descriptions.

### ClusterControl CLI

- Schedule a backup, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=84 \
    --nodes="192.168.40.50:6379" \
    --backup-retention=31 \
    --recurrence='0 1 * * *'
```

No `--backup-method` is required; ClusterControl determines the method from the cluster type.

- Schedule a backup with automatic verification on a standalone Backup Verification Server (Redis Sentinel only), running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=84 \
    --nodes="192.168.40.50:6379" \
    --backup-retention=31 \
    --test-server=192.168.73.92 \
    --recurrence='0 1 * * *'
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`.

- Schedule a backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 84,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "rdb",
                "hostname": "192.168.40.50",
                "port": 6379,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "backup_retention": 31
            }
        },
        "schedule": "0 1 * * *"
    }
```

(continues on next page)

(continued from previous page)

```
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

- Schedule a backup with automatic verification on a standalone Backup Verification Server (Redis Sentinel only), running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 84,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "rdb",
                "hostname": "192.168.40.50",
                "port": 6379,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": true,
                "compression": true,
                "backup_retention": 31,
                "verify_backup_delay": 0,
                "verify_backup": {
                    "server_address": "192.168.73.92",
                    "install_software": true,
                    "disable_firewall": true,
                    "disable_selinux": true,
                    "terminate_db_server": false
                }
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

See *Backup verification* for all `verify_backup` fields.

**Common steps**

Local Storage, Cloud Storage, Schedule, and Preview follow the *Common Wizard Steps* above.

### SQL Server

ClusterControl uses SQL Server's built-in backup engine. No external tool is required.

**Backup methods:**

| Method | Notes |
|---|---|
| `mssqlfull` | Complete database backup |
| `mssqldiff` | Changes since the last full backup |
| `mssqllog` | Log backup for point-in-time recovery |

**Note**

Note Backups always run on the primary node. If the primary is unavailable, the backup cannot proceed. If no previous full backup exists in the target directory, ClusterControl automatically upgrades any Differential or Transaction Log request to a Full backup.

### ClusterControl GUI

**Unique Advanced Settings:**

- **Include system databases**: Defaults to **On**. Set to **Off** to exclude system databases from the backup scope.
- **Encryption**: Always on and not configurable. Every SQL Server backup is encrypted natively by SQL Server itself (`ENCRYPTION (ALGORITHM = AES_256, SERVER CERTIFICATE = ...)` in the generated `BACKUP DATABASE` statement), using a certificate ClusterControl manages. This is separate from the **Enable encryption** toggle used by other backup methods, which does not apply here.

**Local Storage note:** The **Storage location** field is disabled for SQL Server. Backups are always stored on the primary node. The **Storage directory** and **Backup subdirectory** fields remain configurable.

### ClusterControl CLI

- Schedule a full backup, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=42 \
    --nodes="pupnode195:1433" \
    --backup-method=mssqlfull \
    --backup-retention=31 \
    --recurrence='0 1 * * *'
```

- Schedule a differential backup, running nightly at 3:00 AM:

```
s9s backup \
    --create \
    --cluster-id=42 \
    --nodes="pupnode195:1433" \
```

(continues on next page)

(continued from previous page)

```
    --backup-method=mssqldiff \
    --backup-retention=31 \
    --recurrence='0 3 * * *'
```

- Schedule a transaction log backup, running every hour:

```
s9s backup \
    --create \
    --cluster-id=42 \
    --nodes="pupnode195:1433" \
    --backup-method=mssqllog \
    --backup-retention=7 \
    --recurrence='0 * * * *'
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`.

SQL Server backups always run on the primary node. Use `"hostname": "auto"` to let ClusterControl select the primary automatically, which is recommended for Always On Availability Group clusters.

- Schedule a full SQL Server backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 42,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mssqlfull",
                "hostname": "auto",
                "backupdir": "/var/opt/mssql/data/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": false,
                "compression": true,
                "compression_level": -1,
                "backup_retention": 31,
                "backup_system_db": true
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

Use `"backup_method": "mssqldiff"` for differential or `"mssqllog"` for transaction log back-

ups. `"compression_level": -1` uses the SQL Server default compression level.

**Common steps**

Cloud Storage, Schedule, and Preview follow the *Common Wizard Steps* above.

---

### Elasticsearch

ClusterControl uses the `elasticsearch-snapshot` tool (built-in; no external installation required).

**Backup methods:**

| Method | Notes |
|---|---|
| `elasticsearch-snapshot` | Uses the Elasticsearch snapshot API |

**Note**

Note Elasticsearch backups skip the **Local Storage** and **Cloud Storage** steps. Backup data is stored to the shared filesystem path defined by `elastic_shared_filesystem` in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf`.

#### ClusterControl GUI

**Configuration step - unique fields:**

- **Repository**: Points to the repository defined by `elastic_snapshot_repository` in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf`. Additional repositories configured on the cluster appear in the drop-down.

**Advanced Settings:** Only **Retention** is configurable (defaults to `31 days`).

#### ClusterControl CLI

- Schedule an Elasticsearch snapshot backup, running daily at 1:00 AM:

```
s9s backup \
    --create \
    --cluster-id=8 \
    --snapshot-repo-type=fs-nfs \
    --snapshot-repository=s9s-es-repo \
    --backup-retention=31 \
    --recurrence='0 1 * * *'
```

`--snapshot-repo-type=fs-nfs` sets the repository type. Other options are `fs` and `s3`. `--snapshot-repository=s9s-es-repo` is the repository name, found in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf` as `elastic_snapshot_repository`.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`.

- Schedule an Elasticsearch snapshot backup, running daily at 1:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 8,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "elasticsearch-snapshot",
                "snapshot_repo_type": "fs-nfs",
                "snapshot_repository": "s9s-es-repo",
                "backup_retention": 31
            }
        },
        "schedule": "0 1 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

`snapshot_repo_type` options: `fs-nfs`, `fs`, `s3`. `snapshot_repository` value comes from `elastic_snapshot_repository` in `/etc/cmon.d/cmon_<CLUSTER_ID>.cnf`.

**Common steps**

The Schedule and Preview steps follow the *Common Wizard Steps* above.

---

### ClickHouse

ClusterControl uses ClickHouse's own native `BACKUP` SQL command.

**Backup methods:**

| Method | Type | Cloud upload | Notes |
|---|---|---|---|
| `clickhouse-na` | Physical | Yes | Full backup |
| `clickhouse-na` | Physical | Yes | Incremental backup. Automatically chains from the most recent completed backup; no parent backup ID needs to be supplied |

!!! note "Storage location"

```
By default, ClickHouse native backups are written to the `backups` disk␣
↪that ClusterControl registers on the backup node itself (by default `/
↪var/lib/clickhouse/backups/`), using ClickHouse's `Disk('backups', ...)`␣
↪backup destination. Setting a custom **Storage directory** switches the␣
↪backup to ClickHouse's `File(...)` destination at that path instead.␣
↪Storing the backup on the ClusterControl controller is not supported.
```

**Warning**

Backup verification prerequisite

Verifying a ClickHouse backup that was written to the `backups` disk requires the backup directory to be reachable at the same path on the Backup Verification Server (BVS) through shared storage configured outside ClusterControl; otherwise the restore fails with `BACKUP_NOT_FOUND`. See *Backup Verification → ClickHouse* for details. This requirement does not apply to backups uploaded to cloud storage.

### ClusterControl GUI

- **Backup method dropdown**: Labeled **Clickhouse native (full)** and **Clickhouse native (incr)**.
- **Local storage note: Store on Controller** is disabled for ClickHouse; **Store on Node** is the only option. **Storage directory** is optional: leave it blank to use the node's pre-configured `backups` disk, or set a custom path to have ClickHouse write the backup there instead.

**Unique Advanced Settings:**

- **Enable partial backup**: When **On**, specify individual databases to include instead of backing up all databases.

**Backup verification:**

- **Enable backup verification**: Defaults to **Off**. When **On**, additional fields appear to configure the Backup Verification Server (BVS) that ClusterControl restores the backup to after it completes, to confirm it is recoverable. See the prerequisite note above, and *Backup Verification* for BVS setup requirements and field descriptions.

### ClusterControl CLI

Use `s9s backup --create-schedule` for ClickHouse schedules.

- Schedule a full backup, running daily at 2:00 AM:

```
s9s backup \
    --create-schedule \
    --cluster-id=4 \
    --nodes="10.10.15.11:9440" \
    --backup-method=clickhouse-native \
    --recurrence='0 2 * * *'
```

  Do not pass `--on-controller`; controller storage is not supported for ClickHouse. Omit `--backup-directory` to use the node's own `backups` disk, or set it to a custom path so ClusterControl issues `BACKUP ... TO File(...)` instead of `Disk('backups', ...)`.

- Schedule an incremental backup, running every 6 hours:

```
s9s backup \
    --create-schedule \
    --cluster-id=4 \
    --nodes="10.10.15.11:9440" \
    --backup-method=clickhouse-native-incr \
    --recurrence='0 */6 * * *'
```

- Schedule a full backup with automatic verification on a standalone Backup Verification Server, running daily at 2:00 AM:

```
s9s backup \
    --create-schedule \
    --cluster-id=4 \
    --nodes="10.10.15.11:9440" \
    --backup-method=clickhouse-native \
    --test-server=192.168.73.88 \
    --recurrence='0 2 * * *'
```

  See the backup verification prerequisite note above; without shared storage between the backup node and the verification server, this fails with `BACKUP_NOT_FOUND`.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/backup`.

- Schedule a full backup, running daily at 2:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 4,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "clickhouse-native",
                "hostname": "10.10.15.11",
                "port": 9440,
                "compression": true
            }
        },
        "schedule": "0 2 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

  Do not include `cc_storage`; controller storage is not supported for ClickHouse. Omit `backupdir` to use the node's own `backups` disk, or set it to a custom path so ClusterControl issues `BACKUP ... TO File(...)` instead of `Disk('backups', ...)`. Use `"backup_method":`

`"clickhouse-native-incr"` for an incremental schedule.

- Schedule a full backup with automatic verification on a standalone Backup Verification Server, running daily at 2:00 AM:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 4,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "clickhouse-native",
                "hostname": "10.10.15.11",
                "port": 9440,
                "compression": true,
                "verify_backup_delay": 0,
                "verify_backup": {
                    "server_address": "192.168.73.88",
                    "install_software": true,
                    "disable_firewall": true,
                    "disable_selinux": true,
                    "terminate_db_server": false
                }
            }
        },
        "schedule": "0 2 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

See the backup verification prerequisite note above; without shared storage between the backup node and the verification server, this fails with `BACKUP_NOT_FOUND`. See *Backup verification* for all `verify_backup` fields, and see all backup-specific fields in the *job_data reference*.

**Common steps**

All seven steps follow the *Common Wizard Steps* above; see the Local storage note above for the ClickHouse-specific restriction.

### 2.61.6 Manage Existing Schedules

After a backup schedule is created, you can view, edit, pause, resume, or delete it at any time.

### View backup schedules

#### ClusterControl GUI

1. Go to the **Clusters** dashboard and click your target cluster.
2. Go to the **Backups** tab and select the **Schedules** sub-tab.

The list shows all schedules for the cluster, including their status (**Active** or **Paused**), next run time, and last execution time.

#### ClusterControl CLI

Backup schedules are job components. Use `s9s job --list --show-scheduled` to list them:

- List all scheduled backup jobs across all clusters:

```
s9s job --list --show-scheduled
```

- List scheduled jobs for a specific cluster:

```
s9s job --list --show-scheduled --cluster-id=<ID>
```

  Output columns: **ID**, **CID** (cluster ID), **REPEAT** (cron expression with timezone), **STATUS** (ENABLED/DISABLED), **DESCRIPTION**.

#### ClusterControl RPC API

```
curl -k -XPOST \
-d '{
    "operation": "getBackupSchedules"
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

The response includes a `backup_schedules` array. Each entry contains the schedule `id`, `enabled` state, `schedule` (cron with timezone), and the full `job_data` block used when creating the schedule.

---

### Edit a schedule

Editing updates the cron expression, backup method, storage location, retention period, or any other `job_data` parameter. The schedule ID does not change.

> **Warning**
>
> RPC API replaces job_data entirely
>
> When updating a schedule via the RPC API, the `job_data` you send **replaces** the entire existing configuration. It is not merged. Always retrieve the current schedule with `getBackupSchedules` first, modify the fields you need, and send the full `job_data` back.

### ClusterControl GUI

1. Go to **Backups → Schedules** for the cluster.
2. Click the action menu (**…**) on the schedule row and select **Edit**.
3. The schedule wizard reopens with all current values pre-filled. Change any field and click through to **Preview**, then confirm.

### ClusterControl CLI

The s9s CLI does not support editing an existing backup schedule. Use the GUI or RPC API to change an existing schedule.

To achieve the same result via CLI, delete the existing schedule and create a new one with the updated parameters:

```
# Delete the old schedule
s9s backup --delete-schedules --job-id=<OLD_SCHEDULE_ID>

# Create a replacement with updated settings
s9s backup \
    --create \
    --cluster-id=<ID> \
    --nodes="<HOST>:<PORT>" \
    --backup-method=<METHOD> \
    --backup-dir=<DIR> \
    --on-controller \
    --recurrence='<NEW_CRON>'
```

### ClusterControl RPC API

Use `scheduleBackup` with the existing schedule `id` in the `schedule` object. The request replaces the entire schedule configuration.

**Step 1**: Retrieve the current schedule to get its `id` and existing `job_data`:

```
curl -k -XPOST \
-d '{
    "operation": "getBackupSchedules"
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

**Step 2**: Send the updated schedule, including the `id` and the complete `job_data`:

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 50,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "id": 27271,
        "enabled": true,
```

(continues on next page)

(continued from previous page)

```
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mssqldiff",
                "hostname": "auto",
                "backupdir": "/var/opt/mssql/data/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": false,
                "compression": true,
                "compression_level": -1,
                "backup_retention": 60,
                "backup_system_db": true
            }
        },
        "schedule": "0 */8 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

The response returns the full updated schedule under the `schedule` key. The `id` remains the same.

---

### Pause and resume a schedule

Pausing a schedule sets it to disabled so it skips future runs without being deleted. Resuming re-enables it.

#### ClusterControl GUI

1. Go to **Backups → Schedules** for the cluster.
2. Use the **Enable/Disable** toggle on the schedule row, or open the action menu (**…**) and select **Disable** or **Enable**.

#### ClusterControl CLI

Backup schedules are job components. Use `s9s job --disable` to pause and `s9s job --enable` to resume:

- Pause a schedule:

```
s9s job --disable --job-id=<SCHEDULE_ID>
```

- Resume a schedule:

```
s9s job --enable --job-id=<SCHEDULE_ID>
```

To find the schedule ID, run `s9s job --list --show-scheduled --cluster-id=<ID>` and note the **ID** column.

### ClusterControl RPC API

Use `scheduleBackup` with the existing schedule `id` and set `"enabled": false` to pause or `"enabled": true` to resume. The complete `job_data` must be included because the API replaces it entirely.

**Pause** (disable without deleting):

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 50,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "id": 27271,
        "enabled": false,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "mssqldiff",
                "hostname": "auto",
                "backupdir": "/var/opt/mssql/data/backups",
                "backupsubdir": "BACKUP-%I",
                "cc_storage": false,
                "compression": true,
                "compression_level": -1,
                "backup_retention": 31,
                "backup_system_db": true
            }
        },
        "schedule": "0 */6 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

**Resume** (re-enable): Send the same request with `"enabled": true`.

---

### Delete a schedule

Deleting a schedule removes it permanently. Existing backup files created by the schedule are not affected.

### ClusterControl GUI

1. Go to **Backups → Schedules** for the cluster.
2. Click the action menu (**…**) on the schedule row and select **Delete**.
3. Confirm the deletion in the dialog.

### ClusterControl CLI

```
s9s backup \
    --delete-schedules \
    --job-id=<SCHEDULE_ID>
```

To find the schedule ID, run `s9s job --list --show-scheduled --cluster-id=<ID>` and note the **ID** column.

### ClusterControl RPC API

Delete a schedule by calling `deleteJobInstance` on the `/v2/jobs` endpoint. The `job_id` is the schedule ID returned by `getBackupSchedules`.

```
curl -k -XPOST \
-d '{
    "operation": "deleteJobInstance",
    "job_id": 27271
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

Note the endpoint is `/v2/jobs`, not `/v2/backup`.

---

## 2.61.7 RPC API job_data Reference

The `job_data` object inside `schedule.job.job_data` controls the backup configuration for the `scheduleBackup` operation. Field names use the values that the RPC API accepts as input. These differ from the camelCase names that appear in backup record responses from `getBackups`.

Key naming differences between API input (`job_data`) and response (`config`):

| API input (`job_data`) | Response (`config`) | Notes |
|---|---|---|
| `backup_method` | `backupMethod` | |
| `hostname` | `backupHost` | Different name entirely |
| `backupdir` | `backupDir` | |
| `backupsubdir` | `backupSubDir` | |
| `cc_storage` | `ccStorage` | |
| `compression_level` | `compressionLevel` | |
| `encrypt_backup` | `encrypt_backup` | Same in both |
| `use_pigz` | `usePigz` | |
| `use_qpress` | `useQpress` | |
| `xtrabackup_backup_locks` | `xtrabackup_locks` | Different name |
| `xtrabackup_parallellism` | `xtrabackupParallellism` | |

### Common fields

Most backup methods accept these fields:

| Field | Type | Default | Description |
|---|---|---|---|
| `backup_method` | string | required | Method name (see per-method sections below) |
| `hostname` | string | `"auto"` | Node IP or hostname. `"auto"` lets ClusterControl select the least-loaded node |
| `port` | integer | varies | Database port |
| `backupdir` | string | `/storage/backups` | Directory where the backup is stored |
| `backupsubdir` | string | `BACKUP-` | Per-run subdirectory. `%I` expands to the numeric job ID |
| `cc_storage` | boole | `true` | `true` stores on the controller; `false` stores on the backup node |
| `compression` | boole | `true` | Enable compression |
| `compression_level` | integer | `6` | Level 1-9, or `-1` for the tool default |
| `backup_retenti` | integer | `31` | `-1` keeps the backup forever. `0` follows the default retention setting configured for the cluster. Any other positive value sets a custom retention in days for this backup |
| `encrypt_backup` | boole | `false` | Encrypt the backup. ClusterControl manages the key |
| `cloud_only` | boole | `false` | Stream the backup directly to cloud storage without creating a local copy first. Different from `delete_after_upload`, which keeps a local copy until the cloud upload succeeds |
| `upload_backup_da` | object | omit | See *Cloud upload* |

`backup_failover` is not accepted by every method; see *Backup failover* below for its exact scope.

### Cloud upload

Include `upload_backup_data_to_cloud_storage` in `job_data` to also upload the backup to object storage.

```
"upload_backup_data_to_cloud_storage": {
    "cloud_storage_provider": "s3",
    "cloud_storage_credentials_id": 3,
    "bucket": "my-bucket/subpath",
    "backup_retention": 90,
    "delete_after_upload": false
}
```

| Field | Type | Values |
|---|---|---|
| cloud_storag | string | "s3", "gcs", "azure" |
| cloud_storag | integer | Credential ID from **Settings → Cloud storage credentials** |
| bucket | string | Bucket name, optionally with a sub-path |
| backup_reten | integer | Retention for the cloud copy. Same semantics as the top-level backup_retention: -1 keeps it forever, 0 follows the cluster's default retention setting, and any other positive value is a custom retention in days |
| delete_after | boole | Remove the local copy after upload |

### Backup verification

Include verify_backup_delay and verify_backup in job_data to have ClusterControl automatically restore the backup to a standalone Backup Verification Server (BVS) after it completes, to confirm it is recoverable. See *Backup Verification* for BVS prerequisites and the restore steps used for each backup method.

This is supported for mysqldump, xtrabackupfull, xtrabackupincr, mariabackupfull, mariabackupincr, pgdumpall, pgdump, pg_basebackup, pgbackrestfull, pgbackrestdiff, pgbackrestincr, clickhouse-native, clickhouse-native-incr, and rdb (Redis Sentinel only). It is not supported for mongodump, percona-backup-mongodb, mssqlfull/mssqldiff/mssqllog, or elasticsearch-snapshot.

```
"verify_backup_delay": 0,
"verify_backup": {
    "server_address": "192.168.73.88",
    "install_software": true,
    "disable_firewall": true,
    "disable_selinux": true,
    "terminate_db_server": false
}
```

| Field | Type | Values / Notes |
|---|---|---|
| verify_backup_del | integer | Seconds to wait after the backup completes before starting verification. 0 verifies immediately. Set to 86400 or more to verify on an hour/day basis instead of right away |
| verify_backup | object | Omit both fields to skip verification |
| verify_backup.server_address | string | FQDN or IP address of the BVS |
| verify_backup.install_softwa | boole | Install the database software and its dependencies on the BVS before restoring. Set to false only if the BVS already has the software installed |
| verify_backup.disable_firewa | boole | Disable the firewall on the BVS |
| verify_backup.disable_selinu | boole | Disable SELinux/AppArmor on the BVS |
| verify_backup.terminate_db_s | boole | When true, the BVS is terminated and its datadir and configuration files are removed once verification finishes. install_software must be true on the next verification job against the same BVS |

### Backup failover

Include backup_failover in job_data to have ClusterControl automatically retry the backup on another eligible node if the selected node is unreachable when the job starts, instead of failing the job.

This is only supported for MySQL & MariaDB and PostgreSQL & TimescaleDB clusters: mysqldump, xtrabackupfull, xtrabackupincr, mariabackupfull, mariabackupincr, pgdumpall, pgdump, pg_basebackup, and pg_basebackupincr. It is not supported for pgbackrestfull/pgbackrestdiff/pgbackrestincr (pgbackrest selects its backup node from its own repository configuration instead), or any MongoDB, Redis, SQL Server, or Elasticsearch backup method.

```
"backup_failover": false
```

| Field | Type | Values / Notes |
|---|---|---|
| backup_failover | boolea | Default false. Set true to retry on another node automatically |
| backup_failover | string | Optional. Pins the failover attempt to a specific node instead of letting ClusterControl pick any available node |

### mysqldump

```
{
    "backup_method": "mysqldump",
    "hostname": "192.168.40.54",
    "port": 3306,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
```

(continues on next page)

(continued from previous page)

```
    "encrypt_backup": false,
    "backup_failover": false,
    "mysqldump_type": "Complete",
    "extended_insert": true,
    "include_databases": "",
    "use_pigz": false
}
```

| Field | Type | Values / Notes |
|---|---|---|
| `mysql-dump_type` | string | `"Complete"`, `"SchemaOnly"`, `"DataOnly"`, `"MySQLDbOnly"`, `"SchemaAndData"` |
| `ex-tended_insert` | boolean | Use multi-row `INSERT` statements. Default `true` |
| `in-clude_databases` | string | Comma-separated list of databases. Empty string backs up all databases |
| `use_pigz` | boolean | Use parallel gzip for faster compression |

### xtrabackupfull / xtrabackupincr

Use `"backup_method": "xtrabackupfull"` or `"xtrabackupincr"`.

```
{
    "backup_method": "xtrabackupfull",
    "hostname": "192.168.40.54",
    "port": 3306,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "backup_failover": false,
    "xtrabackup_backup_locks": true,
    "xtrabackup_parallellism": 1,
    "xtrabackup_lock_ddl_per_table": false,
    "use_pigz": false,
    "use_qpress": false,
    "throttle_rate_netbw": 0,
    "include_databases": "",
    "wsrep_desync": false
}
```

| Field | Type | Values / Notes |
|---|---|---|
| xtra-backup_b | boole | Use `LOCK TABLES FOR BACKUP`. Default `true` |
| xtra-backup_p | in-te-ger | Parallel copy threads. Default `1` |
| xtra-backup_l | boole | Lock DDL per table to avoid MDL conflicts |
| use_pigz | boole | Use parallel gzip for compression |
| use_qpre | boole | Use QPress compression (for older MariaDB versions) |
| throt-tle_rate | in-te-ger | Network throttle in MB/s. `0` means no throttle |
| in-clude_da | string | Comma-separated list for partial backup. Empty string backs up all databases |
| ws-rep_desy | boole | **Galera clusters only.** When `true`, sets `wsrep_desync=ON` on the backup node before the backup starts, so its large write-sets don't trigger flow control on the rest of the cluster. ClusterControl sets `wsrep_desync=OFF` again once the backup finishes and the node has caught up. Omit or leave `false` for non-Galera (replication) clusters |

### mariabackupfull / mariabackupincr

Use `"backup_method": "mariabackupfull"` or `"mariabackupincr"`. Field names are identical to xtrabackup.

```
{
    "backup_method": "mariabackupfull",
    "hostname": "192.168.40.52",
    "port": 3306,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "cloud_only": false,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "backup_failover": false,
    "xtrabackup_backup_locks": true,
    "xtrabackup_parallellism": 1,
    "xtrabackup_lock_ddl_per_table": false,
    "use_pigz": false,
    "use_qpress": false,
    "throttle_rate_netbw": 0,
    "wsrep_desync": false
}
```

All xtrabackup field descriptions apply identically to mariabackup, including `wsrep_desync` for MariaDB Galera Cluster.

### pgdumpall

```
{
    "backup_method": "pgdumpall",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "backup_failover": false,
    "schema_only": false,
    "data_only": false
}
```

| Field | Type | Notes |
|---|---|---|
| `schema_only` | boolean | Dump only the schema, not data |
| `data_only` | boolean | Dump only the data, not schema |

### pgdump

`pgdump` backs up one or more named databases. Unlike `pgdumpall`, it supports format selection and content filters. Like the other PostgreSQL methods, it also supports backup verification (see *Backup verification*).

```
{
    "backup_method": "pgdump",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "backup_failover": false,
    "backup_failover_host": "",
    "include_databases": "mydb",
    "backup_format": "custom",
    "schema_only": false,
    "data_only": false,
    "no_owner": false,
    "no_privileges": false,
    "verify_backup_delay": 0,
    "verify_backup": {
```

(continues on next page)

(continued from previous page)

```
        "server_address": "192.168.73.88",
        "install_software": false,
        "disable_firewall": false,
        "disable_selinux": false,
        "terminate_db_server": false
    }
}
```

| Field | Type | Values / Notes |
|---|---|---|
| include_databases | string | Comma-separated list of databases to dump (required for pgdump) |
| backup_format | string | "plain", "custom", "directory", "tar" |
| schema_only | boolean | Dump only the schema |
| data_only | boolean | Dump only the data |
| no_owner | boolean | Omit ALTER OWNER statements |
| no_privileges | boolean | Omit GRANT and REVOKE statements |

verify_backup_delay and verify_backup are documented once in *Backup verification* since the same fields apply across all PostgreSQL backup methods. backup_failover and backup_failover_host are documented once in *Backup failover*.

### pg_basebackup

```
{
    "backup_method": "pg_basebackup",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "backup_failover": false
}
```

pg_basebackup supports cloud upload. Include *upload_backup_data_to_cloud_storage* to specify the destination (provider, credentials, bucket); add "cloud_only": true alongside it to stream directly to S3-compatible storage instead of uploading after the backup completes locally.

Example: schedule a pg_basebackup that streams directly to S3 without staging on local disk first, running daily at 1:00 AM.

### ClusterControl CLI

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackup \
    --cloud-only \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --recurrence='0 1 * * *'
```

`--cloud-only` cannot be combined with `--backup-dir` or `--on-controller`. Those flags stage the backup on local/controller storage, which conflicts with streaming it directly to cloud storage.

### ClusterControl RPC API

```
{
    "backup_method": "pg_basebackup",
    "hostname": "192.168.40.63",
    "port": 5432,
    "cloud_only": true,
    "upload_backup_data_to_cloud_storage": {
        "cloud_storage_provider": "s3",
        "cloud_storage_credentials_id": 2,
        "bucket": "my-aws-bucket-eu-west-1/postgresql"
    }
}
```

Omit `backupdir` and `cc_storage` when `cloud_only` is `true`; they specify local/controller storage, which isn't used for a direct-to-cloud stream.

### pg_basebackupincr

PostgreSQL 17 or later only. Requires `summarize_wal = on` on the backup source node. Enable it before scheduling:

```
s9s node --configure-wal --summarize-wal=on \
    --nodes="<HOST>:<PORT>" \
    --cluster-id=<ID> \
    --wait --log
```

Each incremental backup chains from the most recent full `pg_basebackup` in the same directory. ClusterControl reconstructs the chain automatically using `pg_combinebackup` on restore.

```
{
    "backup_method": "pg_basebackupincr",
    "hostname": "192.168.40.63",
    "port": 5432,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
```

(continues on next page)

(continued from previous page)

```
    "cc_storage": true,
    "compression": true,
    "compression_level": 6,
    "backup_retention": 31,
    "encrypt_backup": false,
    "backup_failover": false
}
```

Like `pg_basebackup`, `pg_basebackupincr` supports cloud upload. Include *`upload_backup_data_to_cloud_storage`* to specify the destination; add `"cloud_only": true` alongside it to stream directly to S3-compatible storage without requiring local disk capacity for the full backup size.

Example: schedule an incremental `pg_basebackupincr` that streams directly to S3 without requiring local disk capacity for the full backup size, running every 6 hours.

### ClusterControl CLI

```
s9s backup \
    --create \
    --cluster-id=125 \
    --nodes="192.168.40.63:5432" \
    --backup-method=pg_basebackupincr \
    --cloud-only \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --recurrence='0 */6 * * *'
```

`--cloud-only` cannot be combined with `--backup-dir` or `--on-controller`. Those flags stage the backup on local/controller storage, which conflicts with streaming it directly to cloud storage.

### ClusterControl RPC API

```
{
    "backup_method": "pg_basebackupincr",
    "hostname": "192.168.40.63",
    "port": 5432,
    "cloud_only": true,
    "upload_backup_data_to_cloud_storage": {
        "cloud_storage_provider": "s3",
        "cloud_storage_credentials_id": 2,
        "bucket": "my-aws-bucket-eu-west-1/postgresql"
    }
}
```

Omit `backupdir` and `cc_storage` when `cloud_only` is `true`; they specify local/controller storage, which isn't used for a direct-to-cloud stream.

### pgbackrestfull / pgbackrestdiff / pgbackrestincr

Use `"backup_method": "pgbackrestfull"`, `"pgbackrestdiff"`, or `"pgbackrestincr"`. pgbackrest manages its own repository so `backupdir`, `backupsubdir`, and `cc_storage` are not used.

```
{
    "backup_method": "pgbackrestfull",
    "hostname": "auto",
    "port": 5432,
    "compression": true,
    "compression_level": 6
}
```

`"hostname": "auto"` is recommended. pgbackrest selects the backup node based on its own configuration (standby preference and repository host settings).

### mongodump

```
{
    "backup_method": "mongodump",
    "hostname": "192.168.40.160",
    "port": 27017,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "backup_retention": 31
}
```

Compression is not configurable for `mongodump`. Specify a replica set member in `hostname`, not the primary.

### percona-backup-mongodb

PBM must be installed on all nodes before scheduling a backup. See *MongoDB*.

```
{
    "backup_method": "percona-backup-mongodb",
    "hostname": "192.168.40.170",
    "port": 27019,
    "backup_retention": 31
}
```

`hostname` is a config server replica set node. PBM manages its own backup directory; `backupdir` and `cc_storage` are not used.

### rdb (Redis & Valkey)

Used for all Redis and Valkey cluster types. Always specify `"backup_method": "rdb"` regardless of the underlying cluster type (Sentinel or Cluster mode). If AOF persistence is enabled on the instance, ClusterControl includes the AOF file in the backup automatically.

```
{
    "backup_method": "rdb",
    "hostname": "192.168.40.50",
    "port": 6379,
    "backupdir": "/storage/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": true,
    "compression": true,
    "backup_retention": 31
}
```

### mssqlfull / mssqldiff / mssqllog

Use `"backup_method": "mssqlfull"`, `"mssqldiff"`, or `"mssqllog"`.

```
{
    "backup_method": "mssqlfull",
    "hostname": "auto",
    "backupdir": "/var/opt/mssql/data/backups",
    "backupsubdir": "BACKUP-%I",
    "cc_storage": false,
    "compression": true,
    "compression_level": -1,
    "backup_retention": 31,
    "backup_system_db": true
}
```

| Field | Type | Values / Notes |
|---|---|---|
| `hostname` | string | `"auto"` (recommended) lets ClusterControl target the current primary automatically |
| `backup_system` | boolean | Include system databases (`master`, `model`, `msdb`). Default `true` |
| `compression_level` | integer | `-1` uses SQL Server's built-in compression default |
| `cc_storage` | boolean | SQL Server backups are stored on the primary node; the controller cannot act as storage host. Use `false` |

`port` is not required. ClusterControl resolves the SQL Server port from the cluster configuration. `encrypt_backup` is not accepted for these methods; encryption is always applied natively by SQL Server (`AES_256` via a ClusterControl-managed server certificate) and cannot be disabled. For example, ClusterControl runs a statement equivalent to:

```
BACKUP DATABASE [myexampledb] TO DISK = '/
↪var/lib/backups/BACKUP-838222/myexampledb_full.bak'
WITH CHECKSUM, COMPRESSION, ENCRYPTION (ALGORITHM = AES_256, SERVER␣
↪CERTIFICATE = S9sDbmEncryptCert)
```

### elasticsearch-snapshot

```
{
    "backup_method": "elasticsearch-snapshot",
    "snapshot_repo_type": "fs-nfs",
    "snapshot_repository": "s9s-es-repo",
    "backup_retention": 31
}
```

| Field | Type | Values / Notes |
|---|---|---|
| snapshot_repo_typ | string | "fs-nfs" (NFS-backed filesystem), "fs" (local filesystem), "s3" (S3-compatible object storage) |
| snapshot_reposito | string | Repository name. Matches the value of elastic_snapshot_repository in /etc/cmon.d/cmon_<CLUSTER_ID>.cnf |

hostname, backupdir, and cc_storage are not used. Snapshot storage is configured at cluster setup time via the repository settings.

### clickhouse-native / clickhouse-native-incr

Use "backup_method": "clickhouse-native" for a full backup or "clickhouse-native-incr" for an incremental backup.

```
{
    "backup_method": "clickhouse-native",
    "hostname": "10.10.15.11",
    "port": 9440,
    "compression": true,
    "backup_retention": 31,
    "include_databases": ""
}
```

| Field | Type | Notes |
|---|---|---|
| include_dat | string | Comma-separated list of databases to include. Empty string backs up all databases |
| backupdir | string | Optional. Omit to write to the backups disk registered on the backup node (Disk('backups', ...)). Set a custom path to have ClickHouse write there instead (File(...)) |

cc_storage is not used; controller storage is not supported for ClickHouse. Cloud upload uses the same upload_backup_data_to_cloud_storage object as other methods; see *Cloud upload*. Backup verification uses the same verify_backup/verify_backup_delay fields as other supported methods; see *Backup verification* and the shared-storage prerequisite in *Backup Verification → ClickHouse*.

### 2.61.8 Advanced Example

The examples throughout this page each highlight a single option at a time. The example below combines many of them into one schedule, to give you a sense of how the advanced CLI and RPC API options work together in practice.

The scenario: schedule a full xtrabackup for a MySQL Replication cluster with a custom title, tuned compression and parallelism, DDL-per-table locking, network throttling, encryption, backup failover, and cloud upload to AWS S3 with its own retention and automatic local cleanup, running nightly at 2:30 AM.

#### ClusterControl CLI

```
s9s backup \
    --create \
    --cluster-id=99 \
    --nodes="192.168.40.60:3306" \
    --backup-method=xtrabackupfull \
    --backup-dir=/storage/backups \
    --subdirectory="BACKUP-%H-%I" \
    --on-controller \
    --title="Nightly full xtrabackup with cloud upload" \
    --backup-retention=45 \
    --compression-level=9 \
    --use-pigz \
    --parallelism=4 \
    --encrypt-backup \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --cloud-retention=90 \
    --delete-after-upload \
    --recurrence='30 2 * * *'
```

- `--title` labels the schedule so it is easy to identify later in `--list-schedules` output.
- `--compression-level=9` with `--use-pigz` maximizes compression while using parallel gzip to offset the added CPU cost on large datasets.
- `--parallelism=4` uses 4 threads during the xtrabackup copy phase.
- `--encrypt-backup` enables AES-256 encryption; ClusterControl auto-generates and stores the key.
- `--cloud-provider`, `--s3-bucket`, and `--credential-id` upload the completed backup to AWS S3.
- `--cloud-retention=90` sets a retention period for the cloud copy that is independent of `--backup-retention`.
- `--delete-after-upload` removes the local copy once the cloud upload succeeds, reclaiming disk space on the backup host.

### ClusterControl RPC API

```
curl -k -XPOST \
-d '{
    "operation": "scheduleBackup",
    "cluster_id": 99,
    "schedule": {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job": {
            "command": "backup",
            "job_data": {
                "backup_method": "xtrabackupfull",
                "hostname": "192.168.40.60",
                "port": 3306,
                "backupdir": "/storage/backups",
                "backupsubdir": "BACKUP-%H-%I",
                "cc_storage": true,
                "compression": true,
                "compression_level": 9,
                "backup_retention": 45,
                "encrypt_backup": true,
                "backup_failover": true,
                "xtrabackup_backup_locks": true,
                "xtrabackup_parallellism": 4,
                "xtrabackup_lock_ddl_per_table": true,
                "use_pigz": true,
                "throttle_rate_netbw": 50,
                "upload_backup_data_to_cloud_storage": {
                    "cloud_storage_provider": "s3",
                    "cloud_storage_credentials_id": 2,
                    "bucket": "my-aws-bucket-eu-west-1/mysql-replication",
                    "backup_retention": 90,
                    "delete_after_upload": true
                }
            }
        },
        "schedule": "30 2 * * *"
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/backup
```

- `backup_failover: true` retries the job on another eligible node if `192.168.40.60` is unreachable when the schedule fires.
- `xtrabackup_lock_ddl_per_table: true` avoids long DDL locks on heavily loaded servers. See *MySQL & MariaDB* for background.
- `throttle_rate_netbw: 50` caps network streaming at 50 MB/s to avoid saturating the link during business hours.
- `upload_backup_data_to_cloud_storage` uploads the completed local backup to S3 with its

own 90-day retention, then deletes the local copy. See *Cloud upload* for all supported fields.

## 2.62 Restore Backup

The **Restore Backup** feature in ClusterControl restores a previously created backup back onto its source cluster, to recover from data loss, corruption, or a bad transaction.

There are two ways to use a backup once it exists:

- Restore it onto the cluster it came from, to recover the cluster itself. This page covers that flow.
- Restore it onto a standalone host outside the cluster to confirm it is recoverable, without touching production. See *Backup Verification*.

Restoring can also undo a bad transaction or operation via point-in-time recovery (PITR), or serve as the starting point for testing and cloning environments. See *Create Cluster from Backup* to build a brand-new cluster from an existing backup instead of restoring in place.

Restoring only works on backups created using a ClusterControl-supported backup method, and only back onto the cluster where the backup originated. For backups taken outside of ClusterControl (not using a supported backup method), see *Restore External Backup*; that flow currently only supports MySQL Replication and MySQL Galera clusters.

### 2.62.1 Supported backup methods

Restore Backup supports every method listed in *Supported backup methods*, including binlog-chained backups when restoring with PITR.

### 2.62.2 Prerequisites

Before restoring a backup in ClusterControl, ensure that your environment meets the necessary requirements:

- The backup must have completed successfully and have been created using ClusterControl. Failed or incomplete backups cannot be restored.
- Sufficient disk space on the target node for both the restored data and the temporary directory used during restore.
- Network connectivity between the ClusterControl controller and the source cluster where the backup is stored.
- Appropriate user permissions and privileges on the target node. If SELinux or AppArmor is enabled, ensure it is configured to allow the restore operation to complete.
- The cluster's topology should be otherwise healthy going into the restore. Restoring is meant to recover from a failure, not to fix an already-unstable topology; nodes that are down should come back to a normal state once the restore finishes.

**Warning**

Warning!

Restoring a backup overwrites the existing data on the target node. Make sure you have selected the correct backup and target before confirming the restore.

### 2.62.3 Restoring vs. Verifying

The **How to restore** wizard step (see *Common Wizard Steps* below) offers two paths:

- **Restore on node**: Restores the backup in place, onto a node in the source cluster. This is the recovery path covered on this page.
- **Restore and verify on standalone host**: Restores the backup onto a separate Backup Verification Server (BVS) outside the cluster instead, to confirm it is recoverable without touching production. This path is fully covered in *Backup Verification* and is not duplicated here.

This choice is only offered for methods that support backup verification: `mysqldump`, `xtrabackup`, `mariabackup`, `pgdumpall`, `pgdump`, `pg_basebackup`, `pgbackrest`, `clickhouse-native`/`clickhouse-native-incr`, and `rdb` (Redis Sentinel only). For MongoDB, SQL Server, and Elasticsearch, the wizard goes straight from **Configuration** to **Settings** since there is no standalone-host verification path for these methods.

### 2.62.4 Point-in-Time Recovery (PITR)

Point-in-time recovery lets you restore a database to any moment between a backup and the present, rather than only to the exact point the backup was taken. Instead of accepting data loss back to the last full backup, you can replay the transaction log forward to just before the bad event, such as a dropped table, a botched `UPDATE`, or an accidental `DELETE`, and stop there.

#### Supported databases and backup methods

| Database | Backup method | PITR mechanism | Stop point |
|---|---|---|---|
| MySQL & Percona | `mysqldump` (`Complete` type) | Binary log replay on restore target | Timestamp or binlog position |
| MariaDB | `mysqldump` or `mariadb-dump` (`Complete` type, depending on version) | Binary log replay on restore target | Timestamp or binlog position |
| MySQL & Percona | `xtrabackupfull` / `xtrabackupincr` | Binary log replay on restore target | Timestamp or binlog position |
| MariaDB | `mariabackupfull` / `mariabackupincr` | Binary log replay on restore target | Timestamp or binlog position |
| PostgreSQL & TimescaleDB | `pg_basebackup` / `pg_basebackupincr` | WAL replay (requires WAL archiving enabled) | Timestamp |
| PostgreSQL & TimescaleDB | `pgbackrestfull` / `pgbackrestdiff` / `pgbackrestincr` | WAL replay (archiving managed by pgbackrest) | Timestamp |

PITR is not available for `pgdumpall`, `pgdump`, MongoDB, Redis & Valkey, ClickHouse, or Elasticsearch.

!!! note “SQL Server transaction log restores”

```
SQL Server supports point-in-time recovery through its native backup chain:␣
↪a full backup (`mssqlfull`) chained with differential (`mssqldiff`) and␣
↪transaction log (`mssqllog`) backups. In the restore wizard, you select␣
```

(continues on next page)

(continued from previous page)

```
↪which log backup to restore up to. This is handled through chain␣
↪selection rather than a freeform timestamp, and is described in the [SQL␣
↪Server](#sql-server) section below.
```

## Setup requirements

### MySQL & MariaDB

PITR requires a full backup taken with a PITR-compatible method: `mysqldump`/`mariadb-dump` with dump type `Complete`, `xtrabackupfull`/`xtrabackupincr`, or `mariabackup-full`/`mariabackupincr`. During restore, ClusterControl replays binary logs up to the specified stop time or position.

- **Without a scheduled binlog backup**: ClusterControl replays the binary logs directly from the restore target server. They must still be present and unmodified on the target node at the time of restore. If they have been purged or rotated away, the PITR window shrinks or disappears entirely.
- **With a scheduled binlog backup (cloud only)**: ClusterControl downloads the archived binary logs from cloud storage and replays them first, then replays the latest binary log still present on the restore target server to reach the stop point. This extends the PITR window beyond what local binary log retention alone would allow. See *Schedule Backup* for setting up a binlog backup.

**Warning**

Warning!

Without a scheduled binlog backup, PITR availability depends entirely on binary log retention on the restore target. Ensure `expire_logs_days` (MySQL) or `binlog_expire_logs_seconds` is set to cover your intended recovery window.

### PostgreSQL

For `pg_basebackup` and `pg_basebackupincr`, WAL archiving must be enabled on the PostgreSQL server (`archive_mode = on` with a configured `archive_command`). During restore, ClusterControl restores the base backup, stops the PostgreSQL server, configures the PITR recovery parameters and recovery mode, then starts the server so PostgreSQL replays the archived WAL segments to reach the requested recovery point.

pgbackrest manages WAL archiving automatically as part of its backup process, so no separate WAL configuration is needed. The earliest allowed recovery point for pgbackrest is the backup creation time, or the current time minus the `PITR_RETENTION_HOURS` setting, whichever is more recent.

## Operational notes

- **The recovery target must be after the backup's creation time.** You cannot replay backward past the backup itself.
- **For MySQL & MariaDB, choose Time or Position:**
  - **Time**: Stops binlog replay at a specific date and time in the host's timezone. Use this when you know roughly when the problem occurred.
  - **Position**: Stops at an exact binlog file and byte offset. Use this when you have identified the exact transaction to stop before (for example, from `mysqlbinlog` output).

- **After PITR restore, replicas must be rebuilt.** The primary will be at a different point in the transaction history than the replicas. ClusterControl will resync replicas from the restored primary as part of the restore job, but verify the topology is healthy before resuming writes.
- **The backup must remain available.** If the backup was stored only locally and the local copy has been deleted, or if cloud download fails, the restore job will not start.
- **Test PITR regularly.** Run PITR restores against a *Backup Verification* host to confirm that binary logs or WAL archives are intact and cover the expected recovery window before you need it in production.

---

### 2.62.5 Common Wizard Steps

The restore wizard follows the same flow for every database type, though the **How to restore** step and some **Settings** fields are skipped for methods that don't support them (noted inline where that applies).

#### ClusterControl GUI

**Opening the wizard**

1. Go to the **Clusters** dashboard and click your target cluster.
2. Go to the **Backups** tab, keeping the view set to **All Backups** (default).
3. Click the action menu (**…**) on the backup you want to restore and select **Restore**.

---

**Step 1: Configuration**

- **Backup**: The backup selected for restore.
- **Incremental backups**: Only shown for backups that have incrementals chained to them (`xtrabackup`, `mariabackup`, `pg_basebackupincr`, or a full SQL Server backup with differential/log backups chained to it). Lists the available incremental backups linked to the selected full backup.
- **Restore this backup from**: The storage path or location of the backup.
- **Method**: Read-only panel showing the backup method, size, creation time, and the host where it was taken.

Click **Continue**.

---

**Step 2: How to restore**

Only shown for methods that support backup verification (see *Restoring vs. Verifying* above).

- **Restore on node**: Restores the backup onto the source cluster, to recover it. Select this option for the recovery flow covered on this page.
- **Restore and verify on standalone host**: Restores the backup to an external Backup Verification Server instead. See *Backup Verification*.

Click **Continue**.

---

**Step 3: Settings**

Fields vary by backup method; see *Database-Specific Settings* below for the exact fields shown for each method.

Click **Continue**.

---

#### Step 4: Summary

Review the selected backup and restore options. This is the final confirmation before ClusterControl restores the backup. Click **Finish** to run the job.

### ClusterControl CLI

Restores use `s9s backup --restore` with `--cluster-id` and `--backup-id`, plus `--wait --log` to print job progress while it runs.

- Restore backup ID 3 on cluster ID 2:

```
s9s backup \
    --restore \
    --cluster-id=2 \
    --backup-id=3 \
    --wait \
    --log
```

If the backup is encrypted, it is automatically decrypted during restore.

Method-specific flags (`--memory`, `--pitr-stop-time`, `--psql-immediate`) are listed in each database section below.

"Restore and verify on standalone host" has no CLI equivalent under `--restore`. Use `s9s backup --verify --backup-id=<ID> --test-server=<HOST> --cluster-id=<ID>` instead, which restores the backup to the given host to confirm it is recoverable. See *Backup Verification*.

### ClusterControl RPC API

Restore Backup does not have a dedicated convenience operation. A restore job is created through the generic `createJobInstance` call on `POST /v2/jobs`, with `job.job_spec.command` set to `restore`. All calls require a TLS connection and, in most cases, an authenticated session or inline credentials. See *ClusterControl RPC API → Authenticating with the API* for details.

- Restore backup ID 3 on cluster ID 2:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 2,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 3
            }
        }
```

(continues on next page)

(continued from previous page)

```
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

### 2.62.6 Database-Specific Settings

Each section below covers the **Settings** step fields unique to that database type, plus CLI and RPC API examples.

#### MySQL & MariaDB

Covers MySQL & Percona & MariaDB standalone and primary-replica replication clusters, as well as Galera Cluster topologies (Percona XtraDB Cluster, MariaDB Galera Cluster). The Settings fields are identical across topologies; only the cluster you target differs.

##### ClusterControl GUI

**mysqldump - unique Settings**

- **Restore backup on**: Defaults to the primary node. Select a replica instead if you only need to recover or inspect data outside the primary (for example, checking dropped data), rather than bring the whole cluster back to health.
- **Temporary directory**: Staging area for the backup files during restore. Must have enough free space to hold a full copy of the datadir.
- **Restore system database**: Defaults to **Off**. Turn **On** to also restore MySQL's system database, `mysql`. Leave this off if your existing data directory isn't corrupted and you only need to restore user data for testing or verification.
- **Bootstrap cluster from restored node**: Defaults to **On**. Starts the restored node first so the rest of the cluster bootstraps from it, avoiding data drift between nodes.
- **Point in time recovery (PITR)**: Defaults to **Off**. Only shown for PITR-compatible `mysqldump` backups. When **On**, choose one of:
  - **Time**: Set **Restore time in host TZ** to stop replaying binary logs at a specific date and time.
  - **Position**: Set **Binary log name** and **Log stop position** to stop replaying at an exact binlog coordinate instead of a timestamp.

**xtrabackup / mariabackup - unique Settings**

- **Incremental backups**: Only shown when the selected full backup (`xtrabackupfull` or `mariabackupfull`) has incremental backups chained to it. Select which incremental to restore up to.
- **Restore backup on**: Same behavior as `mysqldump` above.
- **Temporary directory**: Same behavior as `mysqldump` above.
- **xtrabackup –use-memory (MiB)**: xtrabackup only. Memory allocated for the `xtrabackup --prepare` step, analogous to `innodb_buffer_pool_size`. Defaults to 100 MiB; 1024-2048 MiB is a reasonable value if you have the memory available.

- **Make a copy of the datadir before restoring the backup**: mariabackup only. Defaults to **Off**. Turn **On** to keep a copy of the current datadir before it's replaced, in case you need to fall back.
- **Bootstrap cluster from restored node**: Same behavior as `mysqldump` above.
- **Point in time recovery (PITR)**: Available for `xtrabackup` and `mariabackup`. Same **Time/Position** behavior as `mysqldump` above.

### ClusterControl CLI

- Restore a full backup:

```
s9s backup \
    --restore \
    --cluster-id=127 \
    --backup-id=3 \
    --wait \
    --log
```

  Works for `mysqldump`, `xtrabackupfull`, and `mariabackupfull` backups the same way. If the backup has chained incrementals, ClusterControl restores the full chain automatically.

- Restore an xtrabackup with a custom memory allocation for the prepare step:

```
s9s backup \
    --restore \
    --cluster-id=99 \
    --backup-id=12 \
    --memory=2048 \
    --wait \
    --log
```

- Restore using a custom temporary directory to stage the backup files, if the default staging path doesn't have enough free space:

```
s9s backup \
    --restore \
    --cluster-id=99 \
    --backup-id=12 \
    --temp-dir-path=/data/restore-tmp \
    --wait \
    --log
```

- Restore a PITR-compatible `mysqldump` or `mariabackup` backup up to a specific timestamp:

```
s9s backup \
    --restore \
    --cluster-id=127 \
    --backup-id=3 \
    --pitr-stop-time="2020-07-14 14:27:04" \
    --log
```

  `--pitr-stop-time` requires binlog backups to be scheduled alongside the PITR-compatible full backup; see *Schedule Backup → Binlog backup* for setup.

"Restore system database", "Make a copy of the datadir before restoring", and "Bootstrap cluster from restored node" have no dedicated CLI flags; the CLI restore follows ClusterControl's default behavior for these (bootstrap on, system database and datadir copy off).

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Restore a full backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 127,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 3
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

- Restore an xtrabackup with a custom memory allocation:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 99,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 12,
                "memory": 2048
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

- Restore using a custom temporary directory to stage the backup files:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
```

(continues on next page)

(continued from previous page)

```
    "cluster_id": 99,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 12,
                "temp_dir_path": "/data/restore-tmp"
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

- Restore a PITR-compatible backup up to a specific timestamp:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 127,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 3,
                "pitr_stop_time": "2020-07-14 14:27:04"
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

---

### PostgreSQL & TimescaleDB

### ClusterControl GUI

- **Restore backup on**: Disabled; always restores to the primary node.
- **Incremental backups**: Only shown for `pg_basebackupincr` backups with a chain. ClusterControl reconstructs the full chain automatically using `pg_combinebackup`.
- **Make a copy of the datadir before restoring the backup**: pgbackrest only. Defaults to **Off**. Turn **On** to keep a copy of the current datadir before it's replaced.
- **Point in time recovery (PITR)**: Available for `pg_basebackup`/`pg_basebackupincr` (when WAL archiving is enabled) and all `pgbackrest` methods. Defaults to **Off**. When **On**, set **Restore Time** to how far PostgreSQL should replay forward. For pgbackrest, the earliest allowed value is

either the backup's creation time, or the current time minus the **PITR_RETENTION_HOURS** setting (whichever is more recent, if that setting is greater than 1 hour).

- **Bootstrap cluster from restored node**: Defaults to **On**. Starts the restored node first so the rest of the cluster bootstraps from it.

### ClusterControl CLI

- Restore a backup:

```
s9s backup \
    --restore \
    --cluster-id=125 \
    --backup-id=8 \
    --wait \
    --log
```

Works for `pgdumpall`, `pgdump`, `pg_basebackup`, `pg_basebackupincr`, and `pgbackrest-full`/`pgbackrestdiff`/`pgbackrestincr` the same way.

- Restore without replaying the full WAL stream (finish recovery as soon as consistency is reached):

```
s9s backup \
    --restore \
    --cluster-id=125 \
    --backup-id=8 \
    --psql-immediate \
    --log
```

- Restore a pgbackrest backup up to a specific point in time:

```
s9s backup \
    --restore \
    --cluster-id=125 \
    --backup-id=8 \
    --pitr-stop-time="2020-07-14 14:27:04" \
    --log
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Restore a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 125,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
```

(continues on next page)

(continued from previous page)

```
                "backup_id": 8
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

- Restore without replaying the full WAL stream:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 125,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 8,
                "psql_immediate": true
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

---

### MongoDB

MongoDB Replica Set and Sharded Cluster restores each accept only the method used to create the backup: `mongodump` for Replica Set, `percona-backup-mongodb` (PBM) for Sharded Cluster.

#### ClusterControl GUI

**mongodump (Replica Set)**

No Settings step. After **Configuration**, click **Finish** to restore.

**percona-backup-mongodb (Sharded Cluster)**

- **Make a copy of the datadir before restoring the backup**: Defaults to **Off**. Turn **On** to keep a copy of the current datadir before it's replaced.

#### ClusterControl CLI

- Restore a mongodump backup:

```
s9s backup \
    --restore \
```

(continues on next page)

(continued from previous page)

```
    --cluster-id=79 \
    --backup-id=5 \
    --wait \
    --log
```

- Restore a percona-backup-mongodb backup:

```
s9s backup \
    --restore \
    --cluster-id=28 \
    --backup-id=9 \
    --wait \
    --log
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Restore a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 79,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 5
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

The same shape works for `percona-backup-mongodb` backups; only `cluster_id` and `backup_id` change.

---

## Redis & Valkey

Applies to Redis and Valkey, in both Sentinel and Cluster topologies.

### ClusterControl GUI

**Sentinel**

- **How to restore** step is shown (see *Restoring vs. Verifying*).
- **Restore backup on**: Disabled; always restores to the primary node.

- **Make a copy of the datadir before restoring the backup**: Defaults to **Off**. Turn **On** to keep a copy of the current datadir before it's replaced.

**Cluster**

- No **How to restore** step; backup verification is not supported for Cluster mode.
- **Restore backup on**: Disabled; always restores to the primary node.
- **Make a copy of the datadir before restoring the backup**: Same as Sentinel above.

### ClusterControl CLI

- Restore a backup:

```
s9s backup \
    --restore \
    --cluster-id=84 \
    --backup-id=6 \
    --wait \
    --log
```

  Works the same way for both Sentinel and Cluster topologies.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Restore a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 84,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 6
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

---

## SQL Server

### ClusterControl GUI

- **Incremental backups**: Only shown when the selected `mssqlfull` backup has differential or transaction log backups chained to it. Select which one to restore up to.

- No **How to restore** or **Settings** step; backup verification is not supported for SQL Server. After **Configuration**, click **Finish** to restore.

### ClusterControl CLI

- Restore a full backup:

```
s9s backup \
    --restore \
    --cluster-id=42 \
    --backup-id=15 \
    --wait \
    --log
```

  To restore up to a differential or transaction log backup, pass that backup's ID instead; ClusterControl restores the full backup it is chained to first.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Restore a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 42,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 15
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

---

## ClickHouse

ClickHouse restores use ClickHouse's own native `RESTORE` SQL command, always onto the node the backup was taken from.

### ClusterControl GUI

- **How to restore** step is shown (see *Restoring vs. Verifying* above).
- **Restore backup on**: The ClickHouse node the backup was taken from.
- No other unique Settings fields. A banner warns if the selected backup has not been through *Backup Verification*: "Restoring a **non-verified** backup may cause server failures and unexpected results."

### ClusterControl CLI

- Restore a backup:

```
s9s backup \
    --restore \
    --cluster-id=4 \
    --backup-id=13 \
    --wait \
    --log
```

Works the same way for `clickhouse-native` and `clickhouse-native-incr`, and for backups stored locally or in cloud storage. If the backup was written to a custom `backupdir` (see *Create Backup → ClickHouse*) or to cloud storage, ClusterControl (re)configures the matching ClickHouse disk on the target node before restoring, which briefly restarts ClickHouse there.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Restore a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 4,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore_backup",
            "job_data": {
                "backupid": 13
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

**Note**

Note For ClickHouse, `command` is `restore_backup` and the backup ID field is `backupid` (not `restore/backup_id` as used elsewhere on this page), confirmed live via `s9s backup --restore --print-request` and a real restore job's job specification.

---

## Elasticsearch

### ClusterControl GUI

- No **How to restore** step; Elasticsearch verifies snapshots via the Elasticsearch snapshot API instead of a standalone host restore. See *Backup Verification → Elasticsearch* for that flow.

- **Make a copy of the datadir before restoring the backup**: Defaults to **Off**. Turn **On** to keep a copy of the current datadir before it's replaced.

### ClusterControl CLI

- Restore a backup:

```
s9s backup \
    --restore \
    --cluster-id=8 \
    --backup-id=4 \
    --wait \
    --log
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Restore a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 8,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "restore",
            "job_data": {
                "backup_id": 4
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

## 2.63 Restore External Backup

**Restore External Backup** lets you restore a backup that was created outside of ClusterControl onto a cluster it manages. You provide the storage host and path to the backup file, and ClusterControl handles the rest: staging the file, stopping the database, applying the backup, and bringing the node back online.

This feature is only available for MySQL Replication and MySQL Galera clusters.

### 2.63.1 Restoring internal vs. external backups

An **internal backup** is one ClusterControl itself created: it already knows the backup's method, storage location, and any incremental chain it belongs to, so restoring it is mostly a matter of picking it from a list. An **external backup** was taken independently, so you have to tell ClusterControl what it is and where it lives.

| | Internal backup | External backup |
|---|---|---|
| Created by | ClusterControl | Any tool, outside ClusterControl |
| Selecting a backup | Pick it from the backup list | Specify its storage host and absolute path |
| Restore target | Primary or replica node | Primary node only |
| Backup methods | Every method in *Schedule Backup → Supported backup methods* | `mysqldump`, `xtrabackup`, `mariabackup` |
| Verify on a standalone host | Yes, see *Backup Verification* | No |
| Point-in-time recovery (PITR) | Yes, for PITR-compatible backups | No |
| Restore system database | Yes (`mysqldump`) | No |

### 2.63.2 Supported backup methods

| Method | Notes |
|---|---|
| `mysqldump` | Supported file extensions: `.sql.gz`, `.sql.bz2`, `.aes`, `.aes256`, `.aes128` |
| `xtrabackup` | Percona Server and Oracle MySQL clusters |
| `mariabackup` | MariaDB clusters |

### 2.63.3 Prerequisites

- The backup must have been created with one of the methods above; ClusterControl cannot restore an external backup taken with anything else.
- Sufficient disk space on the target node for both the restored data and the temporary directory used during restore.
- The backup file must be reachable over the filesystem from the host you specify, with enough permissions to read it if it sits on a shared network path.
- Network connectivity between the ClusterControl controller and the source cluster.
- Appropriate user permissions and privileges on the target node. If SELinux or AppArmor is enabled, ensure it is configured to allow the restore operation to complete.
- The cluster's topology should be otherwise healthy going into the restore, the same as for *Restore Backup → Prerequisites*.

**Warning**

Warning!

Restoring a backup overwrites the existing data on the target node. Make sure you have selected the correct backup file and target before confirming the restore.

### 2.63.4 Common Wizard Steps

#### ClusterControl GUI

**Opening the wizard**

1. Go to the **Clusters** dashboard and click your target cluster.
2. Go to the **Backups** tab and click **More…**, next to **Create backup**.
3. Choose **Restore backup**, then **Restore an external backup**.

---

**Step 1: Configuration**

| Field | Required | Purpose and guidance |
|---|---|---|
| **Cluster** | Yes | The database cluster that will receive the restored data. Confirm the cluster name and ID before proceeding. |
| **Restore backup on** | Yes | The target database host the backup will be imported onto. External backups can only be restored to the primary node. |
| **Backup Method** | Yes | The restore engine matching your backup type: `mysqldump` or `xtrabackup`/`mariabackup`. |
| **Storage Host** | Yes | The host from which ClusterControl can access the external backup file. |
| **Backup Path (absolute path)** | Yes | Full path to the backup file on the storage host. Supported `mysqldump` extensions: `.sql.gz`, `.sql.bz2`, `.aes`, `.aes256`, `.aes128`. |
| **Temporary directory** | No | Staging area for the backup files during restore. Choose a location with sufficient free space and suitable permissions. |
| **Use database root/admin credentials** | No | Defaults to **On**. When enabled, provide the root or admin credentials that are stored in the backup. After the restore completes, ClusterControl uses these credentials to reconnect to the database and finalize the restoration. Reveals the two fields below. |
| **Database user** | Yes, if the option above is **On** | The root or admin username stored in the backup (for example, `root`). |
| **Database password** | Yes, if the option above is **On** | Password for the database user. |
| **The dump file sets the database to restore the data into** | No | For mysqldump-created backups only. Defaults to **On**. Uses the database selection already contained in the dump file. Review the dump file first and look for `USE <databasename>` if you have any doubt about its target database before turning this **Off**. If **Off**, specify the target database name in the **Database** textbox. |
| **Reset master before restore** | No | Defaults to **Off**. Requests a reset of the binary log state before the import (for example, when the backup file contains GTID information that would otherwise conflict with the target's history). This can affect replication and recovery history, so use it only when explicitly required. |
| **Bootstrap cluster from the restored node** | No | Defaults to **On**. Uses the restored node as the starting point for cluster recovery. Confirm that this node contains the authoritative data set before proceeding. |

Click **Continue**.

---

#### Step 2: Summary

Review the selected backup path and restore options. Click **Finish** to run the job.

### ClusterControl CLI

No `s9s` CLI flags are currently documented for restoring an external backup. Use the ClusterControl GUI for this operation.

### ClusterControl RPC API

No RPC operation is currently documented for restoring an external backup. Use the ClusterControl GUI for this operation.

## 2.64 Backup Verification

The **Backup Verification** feature in ClusterControl restores a backup onto a standalone Backup Verification Server (BVS), outside the source cluster, to confirm the backup is actually recoverable rather than just "successful." This is the standalone-host path introduced in *Restore Backup → Restoring vs. Verifying*; this page covers it in full.

> **Note**
>
> Note Backup Verification requires an Enterprise license.

There are two ways to trigger verification:

- **On demand**, against an existing backup. Use this to spot-check a specific backup, recover a specific piece of data onto an isolated host, or confirm a configuration change doesn't break restorability.
- **Automatically**, as part of a backup schedule, either immediately after the scheduled backup completes or on a delay of up to 24 hours. This is the preferred approach for most environments, since it continuously proves your backups are usable rather than checking them only when you happen to remember to. See *Schedule Backup → Backup verification* for the `verify_backup`/`verify_backup_delay` job_data fields used to configure this.

### 2.64.1 Supported backup methods

| Cluster type | Supported backup methods |
|---|---|
| MySQL-based clusters | mysqldumpPercona Xtrabackup (full, incremental) |
| MariaDB-based clusters | mariadb-dumpMariaDB Backup (full, incremental) |
| PostgreSQL-based clusters | pg_dumppg_dumpallpg_basebackup (full, incremental)pgBackRest (full, incremental, differential) |
| ClickHouse-based clusters | ClickHouse native backup (full, incremental) - see *ClickHouse* below for a storage prerequisite |
| Redis-based clusters | RDB - Sentinel topology only, not Cluster mode |
| Elasticsearch | Elasticsearch snapshot - verified differently; see *Elasticsearch* below |

> **Note**

Note MongoDB (`mongodump`, `percona-backup-mongodb`) and SQL Server (`mssqlfull`/`mssqldiff`/`mssqllog`) do not support backup verification.

### 2.64.2 Prerequisites

- The backup must have completed successfully. Failed or incomplete backups cannot be verified.
- The target Backup Verification Server (BVS) must be reachable by the ClusterControl controller, with the same OS-level access your database nodes use.
- Sufficient disk space on the BVS to hold a full copy of the restored data.
- Network connectivity between the ClusterControl controller and the source cluster where the backup is stored.
- Appropriate user permissions and privileges on the BVS. If SELinux or AppArmor is enabled, either configure it to allow the restore or use the **Disable SELinux/AppArmor** option described below.
- For best compatibility, use a BVS running the same OS and version as the source database server the backup was taken from.

---

### 2.64.3 Common Wizard Steps

Backup Verification uses the same wizard as *Restore Backup*, choosing **Restore and verify on standalone host** at Step 2 instead of **Restore on node**.

#### ClusterControl GUI

**Opening the wizard**

1. Go to the **Clusters** dashboard and click your target cluster.
2. Go to the **Backups** tab, keeping the view set to **All Backups** (default).
3. Click the action menu (**…**) on the backup you want to verify and select **Restore**.
4. Complete **Step 1: Configuration** as described in *Restore Backup*.

---

**Step 2: How to restore**

- Choose **Restore and verify on standalone host**.

Click **Continue**.

---

**Step 3: Settings**

- **Restore backup on**: FQDN or IP address of the target BVS. The host must not already be part of the cluster.
- **Temporary directory**: Staging area for the backup files during restore. Allow a bit more space than the size of a full datadir.

- **Install software**: Defaults to **On**. Installs the database software and its dependencies on the BVS from the package repository before restoring. Turn **Off** only if the BVS already has the correct database software installed (for example, a custom build, or an offline environment without a package repository).
- **Disable firewall**: Defaults to **On** (recommended).
- **Disable SELinux/AppArmor**: Defaults to **On**. If you turn this **Off**, make sure SELinux/AppArmor is configured to allow the database to create files and open sockets.
- **Shutdown the server after the backup is restored**: Defaults to **Off** (recommended). When **Off**, the BVS stays up and online after verification, with its data and packages intact; it's then listed under **ClusterControl GUI → Nodes** as a Backup Verification Server, and you're responsible for decommissioning it yourself. When **On**, ClusterControl terminates the BVS once verification finishes and removes its `datadir` and configuration files; **Install software** must be **On** the next time you verify against the same BVS.
- **Start backup verification (after completion) in**: Only shown when verification is configured as part of a backup schedule, not for on-demand verification. Number of hours to wait after the backup completes before starting verification, up to 24.

Click **Continue**.

---

**Step 4: Summary**

Review the selected backup and BVS settings, then click **Finish** to run the job.

### ClusterControl CLI

On-demand verification of an existing backup uses `s9s backup --verify` with `--backup-id` and `--test-server`:

- Verify backup ID 1 on cluster ID 1 against a standalone host:

```
s9s backup \
    --verify \
    --cluster-id=1 \
    --backup-id=1 \
    --test-server=192.168.0.55 \
    --log
```

  ClusterControl installs the database software on the test server using the same settings as the source cluster, restores the backup there, and the job succeeds only if the restore completes successfully.

To verify a backup automatically right after it's created, add `--test-server` to `s9s backup --create`; see *Schedule Backup → MySQL & MariaDB* for examples. There is no dedicated CLI flag for **Install software**, **Disable firewall**, **Disable SELinux/AppArmor**, or **Shutdown the server after the backup is restored**; the CLI verify job uses ClusterControl's defaults for these (install software and disable firewall/SELinux on, shutdown off).

### ClusterControl RPC API

Backup Verification does not have a dedicated convenience operation. A verify job is created through the generic `createJobInstance` call on `POST /v2/jobs`, with `job.job_spec.command` set to `verify`. All calls require a TLS connection and, in most cases, an authenticated session or inline credentials. See *ClusterControl RPC API → Authenticating with the API* for details.

- Verify backup ID 1 on cluster ID 1 against a standalone host:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 1,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "verify",
            "job_data": {
                "backup_id": 1,
                "test_server": "192.168.0.55",
                "install_software": true,
                "disable_firewall": true,
                "disable_selinux": true,
                "terminate_db_server": false
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

**Note**

Note The public RPC reference does not document the `job_data` fields accepted by the `verify` command. `backup_id` and `test_server` are inferred from their identically-named `s9s backup --verify` CLI flags. `install_software`, `disable_firewall`, `disable_selinux`, and `terminate_db_server` are carried over from the confirmed `verify_backup` object used by `scheduleBackup` (see *Schedule Backup → Backup verification*), since they configure the same BVS settings screen; the ad-hoc `verify` job may or may not nest them the same way. Verify against your controller version, or run `s9s metatype --list-properties --type=CmonJobInstance --long` for the authoritative property list, before relying on this in production. For ClickHouse specifically, this has been confirmed live; see *ClickHouse* below.

### 2.64.4 Database-Specific Settings

#### MySQL & MariaDB

Covers MySQL & Percona & MariaDB standalone and primary-replica replication clusters, as well as Galera Cluster topologies (Percona XtraDB Cluster, MariaDB Galera Cluster). Settings are identical across topologies.

#### ClusterControl GUI

- **Incremental backups**: Only shown when the selected full backup (`xtrabackupfull` or `mariabackupfull`) has incrementals chained to it. Select which incremental to verify up to.

- **Install software**: Installs MySQL, Percona Server, or MariaDB on the BVS, matching the source cluster's vendor and version.

All other Settings fields are common to every method; see *Common Wizard Steps* above.

### ClusterControl CLI

- Verify a full xtrabackup:

```
s9s backup \
    --verify \
    --cluster-id=99 \
    --backup-id=12 \
    --test-server=192.168.73.88 \
    --log
```

  Works the same way for `mysqldump`, `mariabackupfull`, and their incrementals.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Verify a full xtrabackup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 99,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "verify",
            "job_data": {
                "backup_id": 12,
                "test_server": "192.168.73.88",
                "install_software": true,
                "disable_firewall": true,
                "disable_selinux": true,
                "terminate_db_server": false
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

---

### PostgreSQL & TimescaleDB

### ClusterControl GUI

- **Incremental backups**: Only shown when the selected full backup (`pgbackrestfull` or `pg_basebackupincr`) has incrementals chained to it. Select which incremental to verify up to.

- **Install software**: Installs PostgreSQL (or TimescaleDB, or EDB Postgres) on the BVS, matching the source cluster's vendor and version.

All other Settings fields are common to every method; see *Common Wizard Steps* above.

### ClusterControl CLI

- Verify a pgBackRest full backup:

```
s9s backup \
    --verify \
    --cluster-id=125 \
    --backup-id=20 \
    --test-server=192.168.73.90 \
    --log
```

  Works the same way for `pgdumpall`, `pgdump`, `pg_basebackup`, `pg_basebackupincr`, and `pg-backrestdiff`/`pgbackrestincr`.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Verify a pgBackRest full backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 125,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "verify",
            "job_data": {
                "backup_id": 20,
                "test_server": "192.168.73.90",
                "install_software": true,
                "disable_firewall": true,
                "disable_selinux": true,
                "terminate_db_server": false
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

### Redis Sentinel

Backup verification is only available for Redis Sentinel clusters. Redis Cluster and Valkey (Sentinel or Cluster) are not supported.

### ClusterControl GUI

- **Install software**: Installs Redis on the BVS, matching the source cluster's version.

All other Settings fields are common to every method; see *Common Wizard Steps* above.

### ClusterControl CLI

- Verify a backup:

```
s9s backup \
    --verify \
    --cluster-id=84 \
    --backup-id=6 \
    --test-server=192.168.73.92 \
    --log
```

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Verify a backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 84,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "verify",
            "job_data": {
                "backup_id": 6,
                "test_server": "192.168.73.92",
                "install_software": true,
                "disable_firewall": true,
                "disable_selinux": true,
                "terminate_db_server": false
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

### ClickHouse

#### ClusterControl GUI

- **Install software**: Installs ClickHouse on the Backup Verification Server (BVS), matching the source cluster's version. The BVS is configured as a single-node standalone server (embedded Keeper, no SSL, not part of the cluster topology).

All other Settings fields are common to every method; see *Common Wizard Steps* above.

#### ClusterControl CLI

- Verify a ClickHouse native backup:

```
s9s backup \
    --verify \
    --cluster-id=4 \
    --backup-id=11 \
    --test-server=192.168.73.88 \
    --log
```

  Works the same way for `clickhouse-native-incr`.

#### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Verify a ClickHouse native backup:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 4,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "verify_backup",
            "job_data": {
                "backupid": 11,
                "server_address": "192.168.73.88",
                "install_software": true,
                "disable_firewall": true,
                "disable_selinux": true,
                "terminate_db_server": false
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

A ClickHouse native backup written to the default `backups` disk (`Disk('backups', ...)`, see *Create Backup → ClickHouse*) is not copied to the Backup Verification Server (BVS); there is no file-transfer step, so the backup directory must resolve to the same backup data at an identical path on both the storage

node and the BVS host, made available on both (for example, via a shared or network filesystem) as an administrator task performed outside ClusterControl, or verification fails with `BACKUP_NOT_FOUND`; this requirement does not apply to backups written to S3-compatible cloud storage, since the BVS restores those directly from the object store using the cluster's stored credentials.

---

### Elasticsearch

Elasticsearch verifies snapshots differently from every other backup method above. Instead of restoring the backup to a standalone host, ClusterControl sends an HTTP request to the Elasticsearch snapshot API's `_snapshot/_verify` endpoint. This confirms that:

- The snapshot repository (shared filesystem, S3 bucket, etc.) is reachable.
- The repository is readable and writable.
- Every node in the cluster can access it, if it's a shared location.
- The repository is usable for snapshot and restore operations.

This does **not** verify that a specific snapshot's data is intact or restorable, only that the repository backing it is healthy. Elasticsearch clusters are often too large for a full restore-and-verify cycle to be practical, which is why this method is repository-level rather than snapshot-level.

There is no BVS, no **Settings** step, and no `verify_backup`/`verify_backup_delay` scheduling option for Elasticsearch: the check either passes or fails immediately.

#### ClusterControl GUI

1. Go to the **Clusters** dashboard and click your target cluster.
2. Go to the **Backups** tab, keeping the view set to **All Backups** (default).
3. Click the action menu (**…**) on the backup you want to verify and select **Restore**.
4. Complete **Step 1: Configuration** as described in *Restore Backup*.
5. Click **Continue**.
6. **Step 2: How to restore**: Choose **Verify snapshot**.
7. Click **Continue** to proceed to **Step 3: Summary**.
8. Click **Finish** to run the job.

#### ClusterControl CLI

- Verify an Elasticsearch snapshot:

```
s9s backup \
    --verify \
    --cluster-id=8 \
    --backup-id=4 \
    --log
```

`--test-server` is not used; there is no standalone host involved.

### ClusterControl RPC API

All calls go to `POST https://<CONTROLLER_HOST>:9501/v2/jobs`.

- Verify an Elasticsearch snapshot:

```
curl -k -XPOST \
-d '{
    "operation": "createJobInstance",
    "cluster_id": 8,
    "job": {
        "class_name": "CmonJobInstance",
        "job_spec": {
            "command": "verify",
            "job_data": {
                "backup_id": 4
            }
        }
    }
}' \
-b cookies.jar \
https://localhost:9501/v2/jobs
```

## 2.65 Create Cluster from Backup

The **Create Cluster from Backup** feature is a feature in ClusterControl with relevant approach to how you will deploy using *Create Database Cluster* method. With this method, users are given the option to easily create a new cluster from a backup, eliminating hassle, reducing time consumption, and minimizing errors. It also enables easy scalability and ensures idempotency with just a few clicks. This feature provides an intuitive workflow for configuring high-availability setups, replication, load balancing, and more.

Similar to *Create Database Cluster*, you can:

- Provisioning of a new cluster.
- Clone an exising cluster and re-deploy to a new cluster. Helpful especially you need to build a quick QA, staging, or dev environment.
- Create a copy of an existing cluster and migrate to a new cloud environment.
- Built-in validation to test on a different environment or operating system to achieve a rapid outcome.

This guide provides step-by-step instructions on how to create a cluster from backup using ClusterControl.

### 2.65.1 Support Matrix

Currently, using the GUI, there's only one suupported cluster in ClusterControl. See below:

| Database | Vendor | Topology |
|---|---|---|
| Galera Cluster | MariaDB, Percona | Galera certification-based replication |

### 2.65.2 Prerequisites

- Backup should be created using ClusterControl and from an existing cluster.
- Backup should be from a Galera Cluster.
- Only MariaDB Galera and Percona XtraDB Cluster are the types of cluster supported.
- You have sufficient storage space for the target environment. Make sure that the new environment or target nodes has enough disk space to copy and restore the backup.
- Network interoperability between the source of backup (either from an existing cluster node or from the ClusterControl controller host) down to your target environment.
- Ensure that you have appropriate user permissions and privileges on the target environment. If security kernel modules such as AppArmor or SELinux must be enabled, make sure you have appropriate configuration to allow the operation successful.

### 2.65.3 Creating the Cluster from Backup

There is only one method to create a cluster from backup using ClusterControl. As mentioned above, only Galera Cluster is supported to achieve this job.

#### Galera Cluster

There is only one method to accomplish the job. The step-by-step in the following section will guide you on hwo to achieve this.

#### Deployment Steps

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster then click.
2. Go to the **Backups** tab.
3. By default, the **All Backups** view shall be displayed. Otherwise, make sure you click the **All Backups** view.
4. In the list of backups, select the backup you desire to be the source of data to stage the cluster and click the ellipsis button (...).
5. Select and click **Create cluster**.

```
6. First step is **Cluster details**. This provides you the backup␣
↪information, fields to define the name of your cluster, and tags.

   1. **Backup**. This field shows the selected backup that was created␣
↪successfully.

   2. **Method**. This is a panel view where it displays the information␣
↪of the backup such as, the backup method used, size, host taken, and when␣
↪it was created.

   3. Under **Name your cluster** panel, there are two fields you can␣
↪specify:
```

(continues on next page)

(continued from previous page)

```
        1. Fill-in the **Name** of the cluster you want to assign. (Optional)

        2. Add **Tags** to search or group your database clusters. (Optional)

7. Click **Continue**.

8. Second step is **SSH Configuration**. This is where you have to define the SSH connection details to have network interoparibility between the ClusterControl controller, existing cluster, and the target environment for your new cluster.

    1. **SSH user**. Specify root if you have root credentials if you use sudo to execute system commands, specify the username that you wish to use here. The user must exists on all nodes.

    2. **SSH user key path**. Specify the full path of SSH key (the key must exist in ClusterControl node) that will be used by SSH User to perform passwordless SSH.

    3. **SSH port**. Specify the SSH port for target nodes. ClusterControl assumes SSH is running on the same port on all nodes.

    4. **SSH sudo password**. Specify the password if the SSH user that you specified under SSH User requires sudo password to run super-privileged commands. Ignore this if SSH User is root or have no sudo password.

    5. **Install software** is set to **On**. You can disable this if you do not want to install the software if the package is already installed. If set to **On**, ClusterControl will uninstall any existing package found and install the right package and version of the database instead.

    6. **Security configuration** to either disable firewall or SELinux/AppArmor. Leave check if you need to **Disable firewall** (default). Leave check if you need to **Disable SELinux/AppArmor** (default). Otherwise
        uncheck if any of the two needs not be disabled. If you have uncheck **Disable SELinux/AppArmor**, make sure that you have configured properly to allow creating files and opening sockets for the database to run properly.

9. Click **Continue**.

10. Third step is **Node configuration**.

    1. Specify the **Server Port** of your MySQL service. Default is set to 3306.
    2. Specify the **Server data directory** is set to `/var/lib/mysql`. If
```

(continues on next page)

(continued from previous page)

```
↪you have prepared a separate path or on a different volume, replace the␣
↪value with your custom path.
    3. **Admin/Root user**. The database admin username that ClusterControl␣
↪shall configure on all database nodes. This user will be granted global␣
↪SUPER privilege and the GRANT option for localhost only.
    4. **Admin/Root password**. The password for **Admin/Root user**.
    5. **Repository**. Choose "Use vendor repositories" (default) will let␣
↪ClusterControl provision software by setting up and using the database␣
↪vendor's preferred software repository. ClusterControl will always␣
↪install the latest version of what is provided at that moment. Choose "Do␣
↪not setup vendor repositories" if you have a special configuration from␣
↪the vendor (commonly for enterprise databases) and ClusterControl will␣
↪skip the repository configuration part.
        6. **Version**. Make sure that the cluster of the source backup␣
↪should match the version specified here.
    7. **Configuration template**. The dropdown loads all MySQL␣
↪configuration template files under `/etc/cmon/templates` or `/
↪usr/share/cmon/templates` of the ClusterControl node for this cluster␣
↪type. Choose one configuration template for the deployment. See [Configu-
ration␣
↪Template](../management/configuration-management.md#configuration-template).
    8. **Enable SSL encryption**. Toggle on (default) will instruct␣
↪ClusterControl to configure database SSL encryption for both␣
↪client-server and replication. Toggle off will make ClusterControl skip␣
↪this step.

11. Fourth step is **Add nodes**.

    1. **Galera node**. Type the IP address/hostname or valid FQDN of the␣
↪nodes you want to assign for this new cluster. Please note that an odd␣
↪number of nodes is recommended, i.e., 3, 5, 7, etc.

12. Click **Continue** to proceed the last step **Preview**.

13. The last step is **Preview**, this is the last step where it shows all␣
↪the values and choosen options for the ClusterControl to use for setting␣
↪up the backup.
```

### ClusterControl CLI

- Create a three-node Percona XtraDB Cluster using version 8.0 from an existing backup.

```
    s9s cluster \
        --create \
        --cluster-type=galera \
        --
↪nodes="192.168.40.67?master;192.168.40.68?master;192.168.40.69?master" \
```

(continues on next page)

(continued from previous page)

```
        --vendor=percona \
        --cluster-name=pxc_stage_env \
        --provider-version=8.0 \
        --backup-id=466 \
        --os-user=vagrant \
        --os-key-file=/home/vagrant/.ssh/id_rsa \
        --wait \
        --log
    ```

    - where `--
↪nodes="192.168.40.67?master;192.168.40.68?master;192.168.40.69?master"`␣
↪are the ip address of the target nodes.
    - `--cluster-name=pxc_stage_env` parameter is the name of the cluster.
    - `--provider-version=8.0` defines the version of the PXC to be␣
↪installed.
    - `--backup-id=466` defines the selected backup ID for the source␣
↪backup to be used to create the cluster. The backup ID can be located in␣
↪the **All Backups** list view.
    - `--os-user=vagrant` and `--os-key-file=/home/vagrant/.ssh/id_rsa`␣
↪defines the OS user and its path of its public key
    - where the `--wait` and `--log` as the name suggest to print the job␣
↪logs while waiting.
```

## 2.66 Upload Backup to Cloud

Automated backups created using ClusterControl are supported to be offloaded from own premise to external cloud storage. By offloading your database backups to the cloud, you are ensured of redundancy, data durability, and potentially easier disaster recovery.

Supported clouds available are the following:

- Amazon S3
- Google Cloud Storage (GCS)
- Azure Blob Storage (through an S3-compatible interface)
- Other S3-compatible storage (for example, MinIO)

Enabling you to upload your backups to your cloud environment requires that you must have done integrating your *Cloud Providers*. If you have not done yet, please make sure you have added and setup integration for your *cloud storage credentials*.

### 2.66.1 Uploading backups to the cloud

All supported database clusters by ClusterControl shares the same approach when dealing with uploading of backups to the cloud. These procedures are tailored by ClusterControl to make it straightforward with usability through GUI clicks replacing those automated playbooks that DevOps or SysOps might have dealt with.

With that said, the step-by-step guide are provided thru the succeeding sections. Before moving forward thru the steps, there are important pointers for uploading backups to the cloud using ClusterControl as it
```

has its limits. There are certain backup methods that are created using ClusterControl that does not provide the feature and this is tabulated in the next section.

### 2.66.2 Support Matrix

Most database clusters supported by ClusterControl include the backup verification feature. For the complete list, checkout below:

| Database | Vendor | Topology | Backup Methods |
|---|---|---|---|
| MySQL | Percona, Oracle | Standalone, replication | `mysqldump`, `xtrabackup` |
| MariaDB | MariaDB | Standalone, replication | `mysqldump`, `mariabackup` |
| Galera Cluster | MariaDB, Percona | Galera certification-based replication | `mysqldump`, `xtrabackup`/`mariabackup` |
| PostgreSQL | PostgreSQL, EnterpriseDB | Standalone, streaming replication | `pgdumpall`, `pg_basebackup` |
| TimescaleDB | TimescaleDB | Standalone, streaming replication | `pgdumpall`, `pg_basebackup` |
| MongoDB | MongoDB, Percona, MongoDB Enterprise | Replica Set | `rdb, aof` |
| Redis | Redis, Valkey | Sentinel, cluster | `rdb, aof` |
| Microsoft SQL Server for Linux | Microsoft | Standalone, Always On availability group | `mssqlfull` |

### 2.66.3 Prerequisites

- Backup should be created using ClusterControl and from an existing cluster.
- Must have done setting up your *cloud storage credentials*
- supported cloud providers for object storage services are only AWS S3, Google Cloud Storage (GCS), Azure Blob Storage, and other S3-compatible storage (MinIO, DigitalOcean Spaces, etc.)
- Ensure proper network connectivity and interoperability between the controller, your database clusters, and your cloud providers.
- Ensure that you have appropriate user permissions and privileges to your bucket or target object storage location.

### 2.66.4 Default Configuration

- Retention period is set to 180 days if not specified.

### 2.66.5 Steps to Upload a Backup

As mentioned above, all supported database clusters shares the same approach and the steps to do it are applicable for all of them based on the *Support Matrix*. Although, if a backup is uploaded to the cloud, the UI applies a slight change.

This slight change shows a different approach in the step-by-step guide which is tackled in the next section of uploading a backup.

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster then click.
2. Go to the **Backups** tab and make sure you click or set the view to **All Backups** (default).
3. Choose a backup you would like to restore and click the ellipsis button (…).
4. Select and click **Upload** option. This will display a prompt with all the procedures in one display to upload the backup.
5. In the prompt, the following elements are displayed:
    - **Backup ID**: This displays the backup ID assigned for the selected backup and also when this was created.
    - **Cloud Storage**: In this section, it will cover the elements required for all cloud storage information
        - **Credentials**: Select your named cloud credentials profile to use for authentication. This is the credentials you have already setup during setting up your integrations in *Cloud storage credentials* section.
        - **Bucket**: Select your existing bucket or type to create new.

```
                - **Location path**: Directory under bucket where backup␣
↪will be stored. Backup will be stored in bucket root directory if␣
↪location path is empty
                - **Retention period**: By default, it will set to 180␣
↪days. Specify a number to set the number of days you want your backup to␣
↪retain.
        - **Delete after upload**: This is **Off** by default. When␣
↪enabled, this means that delete local backup copy after it has been␣
↪uploaded to the cloud.

6. Click **Upload** button to execute and upload the backup to the cloud.
```

### ClusterControl CLI

The `s9s` CLI has no dedicated command to upload an already-created backup to the cloud. The cloud storage options (`--cloud-provider`, `--s3-bucket`, `--credential-id`, `--delete-after-upload`) can only be set when the backup is first created, not applied afterwards to an existing backup:

```
s9s backup \
    --create \
    --cluster-id=1 \
    --nodes=192.168.0.209 \
    --backup-method=xtrabackupfull \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --delete-after-upload \
    --wait
```

To upload a backup that has already been created, use the **ClusterControl GUI** tab above.

**Recommended Next Steps**

- *Backup Verification*
- *Upload Backup Cloud*

### 2.66.6 Steps to Re-upload a Backup (for backups already in the cloud)

As mentioned in the previous step-by-step section, there are slight changes when a backup is already uploaded and present in the cloud. This also shows that backups can be *downloaded* as an option. Backups that are already in the cloud can still be re-uploaded. Take for an example, if the backup that is uploaded already in the cloud but was accidentally deleted, the backup copy stored in the ClusterControl premise means that it is still be re-uploaded to overwrite its existing directory or file.

As showed earlier in the *Support Matrix*, most of the clusters supported upload to the cloud. However, one cluster that does not affected by this is **Microsoft SQL Server for Linux** clusters. Although it supports uploading to the cloud but download options is not yet currently supported. See *Download Backup From Cloud* for more details.

Make sure to follow the steps below to get through the uploading or re-uploading of backup to the cloud.

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster then click.
2. Go to the **Backups** tab and make sure you click or set the view to **All Backups** (default).
3. Choose a backup you would like to restore and click the ellipsis button (...).
4. Select and click **Cloud options**, then select and click **Upload** option button.
5. In the prompt, the following elements are displayed:
    - **Backup ID**: This displays the backup ID assigned for the selected backup and also when this was created.
    - **Cloud Storage**: In this section, it will cover the elements required for all cloud storage information
        - **Credentials**: Select your named cloud credentials profile to use for authentication. This is the credentials you have already setup during setting up your integrations in *Cloud storage credentials* section.
        - **Bucket**: Select your existing bucket or type to create new.

```
                - **Location path**: Directory under bucket where backup␣
↪will be stored. Backup will be stored in bucket root directory if␣
↪location path is empty
                - **Retention period**: By default, it will set to 180␣
↪days. Specify a number to set the number of days you want your backup to␣
↪retain.
        - **Delete after upload**: This is **Off** by default. When␣
↪enabled, this means that delete local backup copy after it has been␣
↪uploaded to the cloud.

6. Click **Upload** button to execute and upload the backup to the cloud.
```

#### ClusterControl CLI

The `s9s` CLI has no dedicated command to re-upload a backup that is already in the cloud. Use the **ClusterControl GUI** tab above.

**Recommended Next Steps**

- *Backup Verification*
- *Upload Backup Cloud*

## 2.67 Download Backup from Cloud

ClusterControl offers downloading backup from the cloud. Only existing backups that are allowed to be downloaded from the cloud. This means that downloading of backups are tightly coupled or source-bounded. It is locked only to that uploaded backuptaken using ClusterControl.

### 2.67.1 Support Matrix

Most database clusters supported by ClusterControl include the backup verification feature. For the complete list, checkout below:

| Database | Vendor | Topology | Backup Methods |
|---|---|---|---|
| MySQL | Percona, Oracle | Standalone, replication | `mysqldump`, `xtrabackup` |
| MariaDB | MariaDB | Standalone, replication | `mysqldump`, `mariabackup` |
| Galera Cluster | MariaDB, Percona | Galera certification-based replication | `mysqldump`, `xtrabackup`/`mariabackup` |
| PostgreSQL | PostgreSQL, EnterpriseDB | Standalone, streaming replication | `pgdumpall`, `pg_basebackup` |
| TimescaleDB | TimescaleDB | Standalone, streaming replication | `pgdumpall`, `pg_basebackup` |
| MongoDB | MongoDB, Percona, MongoDB Enterprise | Replica Set | `rdb, aof` |
| Redis | Redis, Valkey | Sentinel, cluster | `rdb, aof` |

### 2.67.2 Prerequisites

- Backups uploaded from ClusterControl can only be downloaded.
- Download only to the same cluster where the backup was uploaded.
- Supported cloud providers for object storage services are only AWS S3, Google Cloud Storage (GCS), Azure Blob Storage, and other S3-compatible storage (MinIO, DigitalOcean Spaces, etc.).
- Ensure proper network connectivity and interoperability between the controller, your database clusters, and your cloud providers.
- Ensure that you have appropriate user permissions and privileges to your bucket or target object storage location.

### 2.67.3 Default Configuration

- Retention period is set to 180 days if not specified.

### 2.67.4 Steps to Download a Backup

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster then click.
2. Go to the **Backups** tab and make sure you click or set the view to **All Backups** (default).
3. Choose a backup you would like to restore and click the ellipsis button (…).
4. Select and click **Cloud options**, then select and click **Download** option button.
5. In the prompt, the following elements are displayed:
   - **Backup ID**: This displays the backup ID assigned for the selected backup and also when this was created.
   - **Download from**: This shows where the backup was uploaded and to what cloud it was stored too, which might also linked to your *cloud storage credentials* already setup.
   - **Download to**: This is a drop-down element which shows you the nodes in your cluster including the controller. This allows you to choose what node shall store the backup to be downloaded.
   - **Backup dir in host**: Specify the location of your backup directory to be stored. For example, `/opt/backps/aws-backups-download/`
   - **Retention period**: By default, it will set to 180 days. Specify a number to set the number of days you want your backup to retain.
6. Click **Download** button to execute and download the backup from the cloud.

**Recommended Next Steps**

- *Backup Verification*
- *Upload Backup Cloud*

## 2.68 Create Replica Cluster

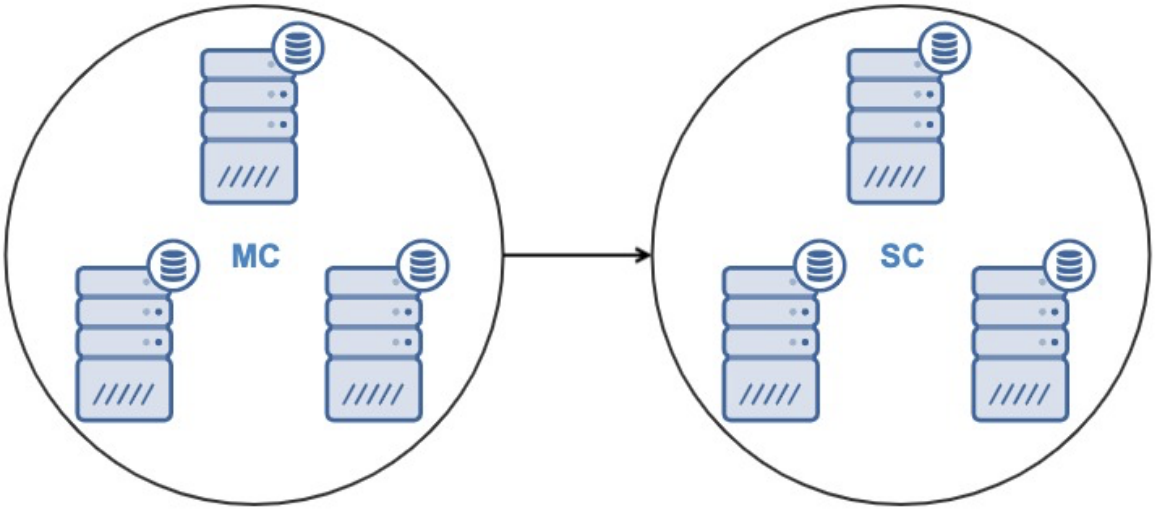
Create a replica cluster is basically cloning a cluster but it sets up a replication layer afterwards from your Source Cluster going to your target cluster (Replica Cluster). Creating a Replica Cluster does copy (or clones) the data from source including its configuration but ensuring that any required parameters such as hostname or IP address are replaced accordingly to have the replica cluster deployment running functional. After the data and configuration is setup on the Slave Cluster, cmon will handle the replication setup to link both clusters to communicate (that is bidirectional replication).

With this feature:

- One node in each cluster will replicate from another node in the other cluster.
- As mentioned earlier, replication will be bi-directional (or uni-directional) between the clusters.
- All nodes in the replica cluster will be read-only by default.

- Active-active clustering is only recommended if applications are only touching disjoint data sets on either cluster since the engine doesn't offer any conflict detection or resolution.

The following diagram shows you how does it look like for an ideal scenario of a Cluster-to-Cluster replication, where MC as Source Cluster and SC as Replica Cluster:




### 2.68.1 Support Matrix

The following database cluster types, vendors and topology are supported:

| Database | Vendor | Topology | Version Supported |
|---|---|---|---|
| Galera Cluster | Percona XtraDB Cluster (PXC) | Galera certification-based replication | 5.6.x and later |
| Galera Cluster | MariaDB Galera | Galera certification-based replication | 5.6.x and later |
| PostgreSQL | PostgreSQL | Standalone, streaming replication | 9.6 and later |
| TimescaleDB | TimescaleDB | Standalone, streaming replication | 9.6 and later |

### 2.68.2 Prerequisites

There are prerequisites that need to be fulfilled prior to the deployment of a replica cluster. See list below:

- Make sure the target database nodes are running on a supported architecture platform and operating system. See *Hardware* and *Operating System*.
- Passwordless SSH (SSH using key-based authentication) is configured from the ClusterControl node to all database nodes. See *SSH Key-based Authentication*.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*.
- The database credentials for admin user/password should be the same from Source Cluster.

### 2.68.3 Create Replica Cluster Options

When creating a replica cluster, you have two options to choose from:

- **Stream data from primary cluster**. This option allows you to setup your desired job to clone or copy the data from the primary or Source Cluster via streaming. This means that, copying of data is happening on-demand and this shall add traffic and load to your Source Cluster.
- **Stage cluster from a backup**. For large dataset or a cluster serving on a high-traffic load, creating a cluster by staging from a backup would be ideal.

### 2.68.4 Galera Cluster

A minimal setup is comprised of one Galera node (no high availability). Recommended setup should be at least 3-nodes at bare minimum for high availability.

The following vendors and versions are supported for a new deployment:

- Percona XtraDB Cluster - 8.0.
- MariaDB Cluster - 10.5, 10.6, 10.11, 11.4 and 11.8.

#### Choosing stream data from primary cluster

Streaming data from primary cluster on a Galera Cluster shall do the following:

- For Percona XtraDB Cluster, it will use Percona Xtrabackup to stream data from its source copying its data to the target node, that is the replica cluster.
- For MariaDB Galera, it will use Mariabackup to stream data from its source copying its data to the target node, that is the replica cluster.

#### Default configuration

Creating a replica cluster, the ClusterControl will simply copy the configuration from the Source Cluster, leaving most everything set based on the changes or its default values given during deployment of the cluster (see *Default Configuration* for deploying a cluster). Other than that, create replica cluster shall have these default configuration but can be modified or changed in the GUI. See the following list below:

- Bi-directional replication is enabled or **"On"**
- Disable firewall is checked
- Disable SELinux/AppArmor is checked
- Default SSH port is 22
- Default database port is 3306
- Shall use vendor repositories when pulling and instaling packages
- After Replica Cluster is deployed, auto-recovery (Node and Cluster) are enabled.

#### Deployment steps

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Clone/Replicate the cluster  → Create replica cluster**

3. Choose **Stream data from primary cluster** to open the GUI for␣
↪**Replicate from** **&lt;cluster_name&gt;** GUI..

4. First step is **Cluster details**.

    1. Specify on the **Select primary** at this step. In the **Cluster␣
↪replication primary**, choose the target primary node where replication␣
↪link shall be established.

    2. **Bi-directional replication** is set to **On**. You can turn it off␣
↪or disable it to have only unidirectional replication.

    3. **Name your cluster**. In the **Name** field, it is optional but␣
↪fill-in here the desired name of the cluster.

    4. **Tags** is a drop-down field but can accept input fields. Any␣
↪existing tags in the drop-down will show up, otherwise, input the new tag␣
↪to be added.

5. Click **Continue**.

6. Second step is **SSH configuration**.

    1. Specify the **SSH Credentials** at this step. In the **SSH user**,␣
↪input the SSH user, then **SSH user key path** for the key path (for␣
↪example /home/clustercontrol/id_rsa.pub).
       Specify the **SSH port** if the port is not using the default 22.␣
↪Specify the **SSH sudo password** if there is a sudo password for the SSH␣
↪user being used here. Specify the option for OS elevation command (`sudo`,
↪ `doas`, `pbrun`).

    2. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    3. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

7. Click **Continue**.

8. Third step is **Node configuration**.
```

(continues on next page)

(continued from previous page)

```
   1. Specify the **Node configuration** at this step. Specify the␣
↪**Server port** if it's not using default port 3306.
       Specify the **Server data directory** if it's not using default `/
↪var/lib/mysql`.

   2. Specify the database admin user/password. Make sure to have the␣
↪value of the admin user and password from the source cluster.
       Specify the database admin user in **Admin/Root user** field.␣
↪Specify the database admin password in *Admin/Root password** field.

   3. Choose the repository in the **Repository** section. Default is set␣
↪to **Use vendor repositories**. Other options are `Do not setup vendor␣
↪repositories` and `Create new repository`.
       If choosing `Do not setup vendor repositories`, this shall not␣
↪install the vendor repository leaving to use AppStream or your main␣
↪repository.
       If choosing `Create new repository`, this shall create/mirror the␣
↪vendor's repositories. Deploy with the mirrored repositories.

   4. Specify the **Version** of your desired database. When creating a␣
↪replica, it is best to determine the source version including the minor␣
↪version number of the Source Cluster.

   5. Select the **Configuration template**. ClusterControl will show the␣
↪templates that has been added or configured in `/etc/cmon/templates`. For␣
↪more details see [Configuration␣
↪template](../management/configuration-management.md#configuration-template)

   6. By default, **Enable SSL encryption** is enabled or **On**. If setup␣
↪does not require SSL (not recommended), opt-out to disable it.

   7. By default, **Create cluster as readonly** is enabled or **On**.␣
↪Disable it if Replica Cluster is writable.

9. Click **Continue**.

10. Fourth step is **Add nodes**.

   1. Specify the **Galera node** that will be added and setup as the␣
↪Replica Cluster. Please note that an odd number of nodes is recommended,␣
↪i.e., 3, 5, 7, etc.
       On this part, input the hostname or IP address of the node.␣
↪Ideally, the cmon will setup the first node added as the node to be␣
↪acting as a replica of the primary node from the Source Cluster.
       Make sure that this node has enough resources to handle the␣
↪replication link from the Source Cluster.

11. Click **Continue** to proceed the last step **Preview**.
```

(continues on next page)

(continued from previous page)

```
12. Fifth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the Replica Cluster.

13. Click **Finish** button to deploy the job.
```

### 2.68.5 PostgreSQL

A minimal setup is comprised of one PostgreSQL standalone primary node (no high availability). For most setup, recommended setup should have at least one replica or standby node. The following vendors and versions are supported for a new deployment:

- PostgreSQL from postgresql.org repository - 14, 15, 16, 17 and 18.
- PostgreSQL from EDB repository - 14, 15, 16 and 17 (requires a valid EDB repository token).

#### Choosing stream data from primary cluster

Streaming data from the Source Clusters on a PostgreSQL should be established in the primary node since it will act as the primary node for replication. ClusterControl will use pg_basebackup to copy and stream from primary node (the primary node from your Source Cluster) to the replica node (primary node in the Replica Cluster).

#### Default configuration

Creating a replica cluster, the ClusterControl will simply copy the configuration from the Source Cluster, leaving most everything set based on the changes or its default values given during deployment of the cluster (see *Default Configuration* for deploying a cluster). Other than that, create replica cluster shall have these default configuration but can be modified or changed in the GUI. See the following list below:

- Bi-directional replication is enabled or **"On"**
- Disable firewall is checked
- Disable SELinux/AppArmor is checked
- Default SSH port is 22
- Default database port is 5432
- Shall use vendor repositories when pulling and instaling packages
- After Replica Cluster is deployed, auto-recovery (Node and Cluster) are enabled.

#### Deployment steps

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Clone/Replicate the cluster  → Create replica cluster**

3. Choose **Stream data from primary cluster** to open the GUI for␣
↪**Replicate from** **&lt;cluster_name&gt;** GUI..
```

(continues on next page)

(continued from previous page)

```
4. First step is **Cluster details**.

    1. Specify on the **Select primary** at this step. In the **Cluster␣
↪replication primary**, choose the target primary node where replication␣
↪link shall be established.

    2. **Bi-directional replication** is set to **On**. You can turn it off␣
↪or disable it to have only unidirectional replication.

    3. **Name your cluster**. In the **Name** field, it is optional but␣
↪fill-in here the desired name of the cluster.

    4. **Tags** is a drop-down field but can accept input fields. Any␣
↪existing tags in the drop-down will show up, otherwise, input the new tag␣
↪to be added.

5. Click **Continue**.

6. Second step is **SSH configuration**.

    1. Specify the **SSH Credentials** at this step. In the **SSH user**,␣
↪input the SSH user, then **SSH user key path** for the key path (for␣
↪example /home/clustercontrol/id_rsa.pub).
       Specify the **SSH port** if the port is not using the default 22.␣
↪Specify the **SSH sudo password** if there is a sudo password for the SSH␣
↪user being used here. Specify the option for OS elevation command (`sudo`,
↪ `doas`, `pbrun`).

    2. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    3. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

7. Click **Continue**.

8. Third step is **Node configuration**.

    1. Specify the **Node configuration** at this step. Specify the␣
↪**Server port** if it's not using default port 3306.
```

(continues on next page)

(continued from previous page)

```
       Specify the **Server data directory** if it's not using default `/
↪var/lib/mysql`.

   2. Specify the database admin user/password. Make sure to have the␣
↪value of the admin user and password from the source cluster.
       Specify the database admin user in **Admin/Root user** field.␣
↪Specify the database admin password in *Admin/Root password** field.

   3. Choose the repository in the **Repository** section. Default is set␣
↪to **Use vendor repositories**. Other options are `Do not setup vendor␣
↪repositories` and `Create new repository`.
       If choosing `Do not setup vendor repositories`, this shall not␣
↪install the vendor repository leaving to use AppStream or your main␣
↪repository.
       If choosing `Create new repository`, this shall create/mirror the␣
↪vendor's repositories. Deploy with the mirrored repositories.

   4. By default, **Enable SSL encryption** is enabled or **On**. If setup␣
↪does not require SSL (not recommended), opt-out to disable it.

9. Click **Continue**.

10. Fourth step is **Add nodes**.

   1. Specify the **Primary node** that will be added and setup as the␣
↪Replica Cluster.

       2. If there are replicas to be setup and added for the Replica␣
↪Cluster, fill in the hostname or IP address in the **Replica nodes**␣
↪field. Make sure to hit enter or
       click the **+** button.

11. Fifth step is **Extensions**. By default, **pgvector** and **PostGIS**␣
↪extensions that ClusterControl supports are disabled.

   1. If you want **pgvector** to be enabled, switch the toggle button to␣
↪**On**

       2. If you want **PostGIS** to be enabled, switch the toggle button␣
↪to **On**

12. Click **Continue** to proceed the last step **Preview**.

13. Sixth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the Replica Cluster.

14. Click **Finish** button to deploy the job.
```

### 2.68.6 TimescaleDB

A minimal setup is comprised of one TimescaleDB standalone primary node (no high availability). Recommended setup should have at least one replica or standby node. The following vendors and versions are supported for a new deployment:

- TimescaleDB - 14, 15, 16, 17 and 18.

#### Choosing stream data from primary cluster

Streaming data from the Source Clusters on a TimescaleDB should be established in the primary node since it will act as the primary node for replication. ClusterControl will use pg_basebackup to copy and stream from primary node (the primary node from your Source Cluster) to the replica node (primary node in the Replica Cluster).

#### Default configuration

Creating a replica cluster, the ClusterControl will simply copy the configuration from the Source Cluster, leaving most everything set based on the changes or its default values given during deployment of the cluster (see *Default Configuration* for deploying a cluster). Other than that, create replica cluster shall have these default configuration but can be modified or changed in the GUI. See the following list below:

- Bi-directional replication is enabled or **"On"**
- Disable firewall is checked
- Disable SELinux/AppArmor is checked
- Default SSH port is 22
- Default database port is 5432
- Shall use vendor repositories when pulling and instaling packages
- After Replica Cluster is deployed, auto-recovery (Node and Cluster) are enabled.

#### Deployment steps

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Clone/Replicate the cluster  → Create replica cluster**

3. Choose **Stream data from primary cluster** to open the GUI for␣
↪**Replicate from** **&lt;cluster_name&gt;** GUI..

4. First step is **Cluster details**.

    1. Specify on the **Select primary** at this step. In the **Cluster␣
↪replication primary**, choose the target primary node where replication␣
↪link shall be established.

    2. **Bi-directional replication** is set to **On**. You can turn it off␣
↪or disable it to have only unidirectional replication.
```

(continues on next page)

(continued from previous page)

```
    3. **Name your cluster**. In the **Name** field, it is optional but␣
↪fill-in here the desired name of the cluster.

    4. **Tags** is a drop-down field but can accept input fields. Any␣
↪existing tags in the drop-down will show up, otherwise, input the new tag␣
↪to be added.

5. Click **Continue**.

6. Second step is **SSH configuration**.

    1. Specify the **SSH Credentials** at this step. In the **SSH user**,␣
↪input the SSH user, then **SSH user key path** for the key path (for␣
↪example /home/clustercontrol/id_rsa.pub).
       Specify the **SSH port** if the port is not using the default 22.␣
↪Specify the **SSH sudo password** if there is a sudo password for the SSH␣
↪user being used here. Specify the option for OS elevation command (`sudo`,
↪ `doas`, `pbrun`).

    2. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    3. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

7. Click **Continue**.

8. Third step is **Node configuration**.

    1. Specify the **Node configuration** at this step. Specify the␣
↪**Server port** if it's not using default port 3306.
       Specify the **Server data directory** if it's not using default `/
↪var/lib/mysql`.

    2. Specify the database admin user/password. Make sure to have the␣
↪value of the admin user and password from the source cluster.
       Specify the database admin user in **Admin/Root user** field.␣
↪Specify the database admin password in *Admin/Root password** field.

    3. Choose the repository in the **Repository** section. Default is set␣
↪to **Use vendor repositories**. Other options are `Do not setup vendor␣
```

(continues on next page)

(continued from previous page)

```
↪repositories` and `Create new repository`.
       If choosing `Do not setup vendor repositories`, this shall not␣
↪install the vendor repository leaving to use AppStream or your main␣
↪repository.
       If choosing `Create new repository`, this shall create/mirror the␣
↪vendor's repositories. Deploy with the mirrored repositories.

    4. By default, **Enable SSL encryption** is enabled or **On**. If setup␣
↪does not require SSL (not recommended), opt-out to disable it.

9. Click **Continue**.

10. Fourth step is **Add nodes**.

    1. Specify the **Primary node** that will be added and setup as the␣
↪Replica Cluster.

       2. If there are replicas to be setup and added for the Replica␣
↪Cluster, fill in the hostname or IP address in the **Replica nodes**␣
↪field. Make sure to hit enter or
       click the **+** button.

11. Fifth step is **Extensions**. By default, **pgvector** and **PostGIS**␣
↪extensions that ClusterControl supports are disabled.

    1. If you want **pgvector** to be enabled, switch the toggle button to␣
↪**On**

       2. If you want **PostGIS** to be enabled, switch the toggle button␣
↪to **On**

12. Click **Continue** to proceed the last step **Preview**.

13. Sixth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the Replica Cluster.

14. Click **Finish** button to deploy the job.
```

## 2.69 Add Database Node

Adding a database node in ClusterControl is similar to adding a database node or adding a primary node. To simplify the terminology in this page, the name **Add Node** is simply adding a primary database node or denoting *database node* is simply a primary database node.

A primary node denotes that it is a database node ready for accepting read and writes. In contrast with *Add Replica Node*, the Add Database Node refers only to a primary database node. If you are looking to add a replica instead, see *Add Replica Node*.

Adding a database node is a common goal in ClusterControl for improving the resilience, read scalability,

and availability of your database systems; especially for having a disaster recovery located on a separate region, collocation, or in different geography. ClusterControl simplifies this process significantly by eliminating the hassle and manual work.

**Info**

Adding a database node stepwise, and that you can only add one node in the course of action. You can run an iterative process by automating this with ClusterControl CLI.

### 2.69.1 Support Matrix

The following database cluster types, vendors and topology supports adding a database node:

| Database | Vendor | Topology |
|---|---|---|
| Galera Cluster | MariaDB, Percona | Galera certification-based replication |
| MongoDB | MongoDB, Percona, MongoDB Enterprise | Replica set, sharded cluster |
| Redis | Redis, Valkey | Sentinel, cluster |
| Microsoft SQL Server for Linux | Microsoft | Standalone, Always On availability group |
| Elasticsearch | Elastic | Single-node cluster, high availability cluster |
| ClickHouse | ClickHouse | Standalone, replicated cluster, sharded cluster |

### 2.69.2 Prerequisites

When ading a node, it is very important to determine and prepare your host to have the requirements such as hardware, network, and user that is necessary to be accessed by ClusterControl and other database nodes.

There are prerequisites that need to be fulfilled prior to the deployment for all database clusters:

- Make sure the target database nodes are running on a supported architecture platform and operating system. See *Hardware* and *Operating System*.
- Passwordless SSH (SSH using key-based authentication) is configured from the ClusterControl node to all database nodes. See *SSH Key-based Authentication*.
- Make sure that the OS user you setup for passwordless login is the same user your ClusterControl and its cluster used during deployment.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*.

### 2.69.3 MySQL Galera

The following vendors and versions are supported for adding database node:

- Percona XtraDB Cluster - 8.0.
- MariaDB Galera - 10.4, 10.5, 10.6 (LTS), 10.11 (LTS) and 11.4 (LTS).

Adding a database node requires that your existing cluster must have at least one node running. When the job is in progress, ClusterControl will rely on the Galera's mechanism to do a full SST from source to the

target node (database node to be setup and added). This means that ClusterControl relies on binary backup to stream data to copy from source to the target database. Binary backup does not lock your primary node once a new node is added. However, If you are concerned of the traffic and/or the data to be copied from your Primary node is big, we recommended to add it during low peak period or use the *backup created* taken using ClusterControl. This shall be discussed on how to rebuild from backup in the following section below.

Below is how ClusterControl relies on Galera's mechanism to apply a full SST:

- For Percona XtraDB Cluster, it will use Percona XtraBackup to stream data from its existing primary to the target node.
- For MariaDB Server, it will use Mariabackup to stream data from its its existing primary to the target node.

This is set through `wsrep_sst_method` upon deployment.

### Default configuration

By default, ClusterControl deploys MySQL/MariaDB database with the following configurations:

- During the job process, ClusterControl disables cluster and node recovery. It will re-instate its former values accordingly.
- ClusterControl sets `wsrep_sst_method` to `xtrabackup` for Perocna XtraDB Cluster or `mariabackup` for MariaDB Galera.
- MySQL GTID with `log_slave_updates` enabled (Oracle MySQL and Percona only).
- MariaDB GTID with `log_slave_updates` enabled (MariaDB only).
- a database node shall act as a primary having `read_only=OFF` and `super_read_only=OFF` (if supported).
- `performance_schema` is enabled.
- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.

**Note**

Note If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add new  → Primary node**

3. Choose **Create a primary node**

4. First step is **Node configuration**.
```

(continues on next page)

(continued from previous page)

```
    1. **Data directory** is set to `/var/lib/mysql`. If you have prepared␣
↪a separate path or on a different volume, replace the value with your␣
↪custom path.

    2. **Galera Segment** is set to `0` by default. If you are adding a␣
↪node into different segment, set the value here i.e. > 0. See␣
↪[here](https://severalnines.com/blog/using-mysql-galera-cluster-replication-create-geo-
↪for more about [segments in␣
↪Galera](https://severalnines.com/blog/using-mysql-galera-cluster-replication-create-geo

    3. **Configuration template** is set to `my.cnf.[galera-version]galera`␣
↪where _[galera-version]_ entails the vendor and version of Galera␣
↪ClusterControl intends to add this node. If you have your own custom␣
↪template, ClusterControl will show the templates that has been added or␣
↪configured in `/etc/cmon/templates`. For more details see [Configuration␣
↪template](../management/configuration-management.md#configuration-template)

    4. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package version instead.

    5. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

5. Click **Continue**.

6. Second step is **Advanced Settings**.

    1. **Rebuild from a backup** is disabled. If you desire to rebuild your␣
↪data instead of streaming from the Primary node, toggle it to **On**.␣
↪This shall reveal a drop-down list with all full-backup created using␣
↪Clustercontrol.

    2. **Include in LoadBalancer set (if exists)** is disabled. Switch it␣
↪to **On* if you want ClusterControl to include this new node in the␣
↪existing node load balancing set.

7. Click **Continue**.

8. Third step is **Add node**.

    1. In the **Node** field, this is where you have to specify and input␣
```

(continues on next page)

(continued from previous page)

```
↪the IP address or hostname of your target node. If you intend to add␣
↪another one, this will just overwrite what was previously inputted and␣
↪added since it only allows one node at a time.

9. Click **Continue** to proceed the last step **Preview**.

10. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

11. Click **Finish** button to deploy the job.
```

#### ClusterControl CLI

Add a new Galera node, streaming a full SST from an existing node in cluster ID 1:

```
s9s cluster \
    --add-node \
    --cluster-id=1 \
    --nodes=10.10.10.14 \
    --wait
```

To rebuild the new node from an existing full backup instead of streaming from the primary (the **Rebuild from a backup** option in the GUI wizard), add `--backup-id`:

```
s9s cluster \
    --add-node \
    --cluster-id=1 \
    --nodes=10.10.10.14 \
    --backup-id=10 \
    --wait
```

**Recommended Next Steps**

- *Add Replica Node*
- *Create Replica Cluster*

### 2.69.4 MongoDB Replica Set

The following vendors and versions are supported for adding database node:

- MongoDB Enterprise - 7.0 and 8.0
- MongoDB Community - 7.0 and 8.0
- Percona Server for MongoDB - 7.0 and 8.0

Adding a database node requires that your existing cluster must have at least one node running as primary. If no primary found, ClusterControl will eventually fail the job process since it will fail to identify an elibible primary in that replica set.

When the job is in progress, ClusterControl will execute the `rs.add()` function to add the new node to its target replica set. If you are concerned of the traffic and/or the data to be copied from your Primary node is big, we recommended to add it during low peak period.

Once the adding a node job is done, the new node shall have **Role** of **Secondary** once viewed in the **Nodes** tab.

### Default configuration

By default, ClusterControl deploys MongoDB Replica Set members with the following configurations:

- Configure `setParameter.enableLocalhostAuthBypass: true` inside MongoDB configuration file.
- ClusterControl will create and grant necessary roles for additional MongoDB users - `admin.cmon_backup` for backup and restore purposes and `admin.cmonexporter` for query monitoring.
- Configure `sharding.clusterRole: shardsvr` inside MongoDB configuration file.

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template* for details.

**Warning**

Attention

It is possible to deploy only a DB server only that has one primary database node already in the replica set and no other nodes or arbiter. The caveat of this approach is no automatic failover. If the primary node goes down then manual failover is required to make the remaining secondary node run as primary. Automatic failover works fine with 3 nodes and more.

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add new  → Primary node**

3. Choose **Create a primary node**

4. First step is **Node configuration**.

    1. **Node type** has values `Db server` and `Arbiter`. Choose `Db␣
↪server` to add a primary database node.

    2. **Replica set** is a drop-down list. When click this shall reveal␣
↪replica set created in the cluster. Make sure you know the replica set␣
↪you are intending this node to be assigned and choose it as you desire.

    3. Specify the **Port** of your target replica node. Default is set to␣
↪27017.

    4. **Install software** is set to **On**. You can disable this if you␣
```

(continues on next page)

(continued from previous page)

```
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    5. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

5. Click **Continue**.

6. Third step is **Add node**.

    1. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target node. If you intend to add␣
↪another one, this will just overwrite what was previously inputted and␣
↪added since it only allows one node at a time.

7. Click **Continue** to proceed the last step **Preview**.

8. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

9. Click **Finish** button to deploy the job.
```

#### ClusterControl CLI

Add a new secondary node to an existing MongoDB replica set on cluster ID 3:

```
s9s cluster \
    --add-node \
    --cluster-id=3 \
    --nodes=192.168.99.20 \
    --wait
```

### 2.69.5 MongoDB Sharded Cluster

The following vendors and versions are supported for adding database node:

- MongoDB - 6.0, 7.0 and 8.0
- MongoDB Enterprise - 7.0 and 8.0
- Percona - 6.0, 7.0 and 8.0

Adding a database node requires that your existing MongoDB Sharded Cluster must have at least one node running as primary. If no primary found, ClusterControl will eventually fail the job process since it will

fail to identify an eligible primary in that replica set. Whether this replica set is a Config Server or Shard Server, it requires a primary node to run healthy so that the new node can join the cluster.

When the job is in progress, ClusterControl will execute the `rs.add()` function to add the new node to its target replica set. If you are concerned of the traffic and/or the data to be copied from your Primary node is big, we recommended to add it during low peak period. If you are going to add a router or `mongos`, this can be quick since router will act as your load balancer and does not store data. ClusterControl can configure quickly and does not have any impact at all to your database nodes or existing MongoDB cluster.

Once the adding a node job is done, the following roles shall be assigned:

- **Config Server**. This role is assigned if you added a Config server node
- **Mongos**. This role is assigned if you added a router/mongos node. This is your load balancer.
- **Primary/Secondary**. If you have added the node as part of your sharded server, then primarily, the new node shall have **Secondary** since your existing cluster shall have an assigned primary role assigned.

### Default configuration

By default, ClusterControl deploys MongoDB Sharded Cluster with the following configurations:

- Configure `setParameter.enableLocalhostAuthBypass: true` inside MongoDB configuration file.
- ClusterControl will create and grant necessary roles for additional MongoDB users - `admin.cmon_backup` for backup and restore purposes and `admin.cmonexporter` for query monitoring.
- Configure `sharding.clusterRole: shardsvr` inside MongoDB configuration file if node added is a DB server and part of the shard.
- Configure `sharding.clusterRole: configsvr` inside MongoDB configuration file if node added is a Configuration server.
- sets the `sharding.configDB` to the replica set config servers when adding a router/mongos.
- sets `security.keyFile: /etc/mongo-cluster.key` when adding a router/mongos.

**Warning**

Attention

It is possible to deploy nodes either router, config, or database node with less than 3 nodes. However, for high availability and no single-poit of failure (SPOF), we recommend to make sure to have at least 2 or 3 nodes per shard, at least 2 nods for config server, and 2 nods for your router/mongos.

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (...)

```
2. Choose **Add new  → Node**

3. Choose **Create a database node**
```

(continues on next page)

(continued from previous page)

```
4. First step is **Node configuration**.

    1. **Create router/mongos**. By default, this is `Off`.

    2. **Node type** has values `Db server`, `Config server`, and `Arbiter`.
↪ Choose `Db server` to add a primary database node with shardsvr role.␣
↪Choose `Config server` if you add it as a `configsvr` role, or arbiter as␣
↪the name suggest.

    2. **Replica set** is a drop-down list. When click this shall reveal␣
↪replica set created in the cluster. Make sure you know the replica set␣
↪you are intending this node to be assigned and choose it as you desire.

    3. Specify the **Port** of your target replica node. Default is set to␣
↪27017. Make sure that you set the port correctly to which port you are␣
↪aiming or to make it uniform as what your existing node roles are using,␣
↪but not required instead it's recommended.

    4. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    5. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

5. Click **Continue**.

6. Third step is **Add node**.

    1. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target node. If you intend to add␣
↪another one, this will just overwrite what was previously inputted and␣
↪added since it only allows one node at a time.

7. Click **Continue** to proceed the last step **Preview**.

8. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

9. Click **Finish** button to deploy the job.
```

#### ClusterControl CLI

Add a data node to shard replica set `replset2` on cluster ID 12:

```
s9s cluster \
    --add-node \
    --cluster-id=12 \
    --nodes="mongodb://192.168.1.20?rs=replset2" \
    --wait
```

Add a config server node instead, using the `mongocfg://` scheme:

```
s9s cluster \
    --add-node \
    --cluster-id=12 \
    --nodes="mongocfg://192.168.1.21" \
    --wait
```

Add a router/mongos node, using the `mongos://` scheme:

```
s9s cluster \
    --add-node \
    --cluster-id=12 \
    --nodes="mongos://192.168.1.22" \
    --wait
```

### 2.69.6 Redis Sentinel

The following vendors and versions are supported for adding database node:

- Redis Sentinel - 6 and 7

Essentially, as mentioned earlier, adding a database node in this page is simply adding a primary database node. However, in this case, action goes for adding a node but the end result, the newly database node added shall act as a replica, instead of a primary. In this manner, adding a database node requires a primary node in the course of action in order to complete the job successfully.

If the existing primary is down, job can still be done successfully. However, the primary node needs to be up sometime so that the replica can send a `PSYNC` to the primary node when its up and it can start syncing its data from the primary going to be replicated down to the replica node (the node that had just been added).

Apart from adding your database node as secondary, ClusterControl will also deploy Sentinel node which shall run on port 26379.

#### Default Configuration

By default, ClusterControl deploys Redis cluster instances with the following configurations:

- Default port points to `6379` for the Redis server and port `26379` for Sentinel service.
- ClusterControl will configure the Redis instances with `appendonly` enabled.
- ClusterControl will secure the instance with AUTH enabled and configure the `requirepass` and `masterauth` options.

- ClusterControl will enable TLS encryption. To access the cluster, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Redis being killed by OOM.
- Deploys sentinel node co-located within the same node (Redis instance) being added.

!!! Tips

```
Redis Sentinel requires 3 nodes for automatic primary promotion. When you␣
↪have only one-single Redis Sentinel node, make sure you add two or make␣
↪sure to have an odd total number of nodes per cluster.
```

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add node**

3. First step is **Node configuration**.

    1. Specify the **Port** of your target database (replica) node. Default␣
↪is set to `6379`.

    2. Specify the **Redis sentinel port** for your sentinel service.␣
↪Default is set to `26379`.

    3. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    4. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

4. Click **Continue**.

5. Third step is **Add node**.

    1. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target node. If you intend to add␣
↪another one, this will just overwrite what was previously inputted and␣
↪added since it only allows one node at a time.
```

(continues on next page)

(continued from previous page)

```
6. Click **Continue** to proceed the last step **Preview**.

7. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

8. Click **Finish** button to deploy the job.
```

#### ClusterControl CLI

Add a new replica node, with Sentinel co-located on the same host, to cluster ID 5:

```
s9s cluster \
    --add-node \
    --cluster-id=5 \
    --nodes="redis://192.168.40.193;redis-sentinel://192.168.40.193" \
    --wait
```

### 2.69.7 Redis/Valkey Cluster

The following vendors and versions are supported for adding a database node:

- Redis Cluster - 6 and 7
- Valkey Cluster - 7 and 8 (only RHEL 8/9 are supported)

Adding a database node (primary database node) requires a running and healthy primary node in the course of action in order to complete the job successfully. When the job is in progress, ClusterControl will setup the database node then executes CLUSTER MEET and rebalances the slots in the cluster to ensure a more even distribution across all nodes.

#### Default Configuration

By default, ClusterControl deploys Redis cluster instances with the following configurations:

- ClusterControl will configure the Redis/Valkey instances with `appendonly` enabled.
- ClusterControl will secure the instance with AUTH enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the cluster, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Redis being killed by OOM.
- ClusterControl uses cluster bus port i.e. main-port + 1000 unless manually specified during *cluster deployment*.

#### Deployment steps

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add new → Primary Node**

3. First step is **Node configuration**.

    1. Specify the **Port** of your target database node. Default is set to␣
↪`6379`.

    2. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    3. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

4. Click **Continue**.

5. Third step is **Add node**.

    1. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target node. If you intend to add␣
↪another one, this will just overwrite what was previously inputted and␣
↪added since it only allows one node at a time.

6. Click **Continue** to proceed the last step **Preview**.

7. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

8. Click **Finish** button to deploy the job.
```

### ClusterControl CLI

Add a new primary node for a new shard, using the `redis-primary://` scheme, to cluster ID 6:

```
s9s cluster \
    --add-node \
    --cluster-id=6 \
    --nodes="redis-primary://192.168.0.131" \
    --wait
```

### 2.69.8 SQL Server

The following versions for Microsoft SQL Server are supported for adding a database node:

- Microsoft SQL Server for Linux - 2019, 2022 and 2025

Adding a database node will deploy a secondary database node. In this action, ClusterControl will setup the node as part of the Always On Availability Groups (AGs). This requires to have a running and healthy primary node in the course of action in order to complete the job successfully.

**Warning**

Attention

Only hostname or FQDN is supported. Therefore, proper host naming and mapping must be performed beforehand. You may use `/etc/hosts` or DNS mapping to achieve this. When adding a database node in ClusterControl, entering an IP address will produce an error.

#### License

Deploying SQL Server with ClusterControl does not change how you must license SQL Server. Each node where SQL Server is running requires appropriate licensing, just as it would if you deployed SQL Server manually. The presence of ClusterControl as an orchestration or management layer has no bearing on Microsoft's licensing rules.

When deploying SQL Server in a clustered environment, licensing rules still apply to each server running SQL Server. However, the specifics differ depending on whether the node is actively running SQL Server or is a passive failover node:

- Active Nodes: Each active node that runs SQL Server services needs to be fully licensed.
- Passive Failover Nodes: Under current SQL Server licensing rules, you can typically have one passive failover node per licensed instance without needing an additional license for that passive node. The passive node is considered a "standby" server that can take over if the active server fails. It must truly remain passive, running no active workloads except those required for maintaining the standby state.

Obtain your SQL Server license directly from Microsoft, a certified Microsoft Partner, or a cloud marketplace (if applicable).

#### Default configuration

By default, ClusterControl deploys SQL Server with the following configurations:

- All database nodes must use fully qualified domain name (FQDN). IP address is not supported.
- Enforces the SQL Server user's password policy as shown here.
- At the moment, ClusterControl only deploys AlwaysOn with asynchronous-commit mode, where it does not wait for any secondary replica to write incoming transaction log records to disk.
- Sets `hadrenabled = 1` in `/var/opt/mssql/mssql.conf` config.
- Enables sqlagent in `/var/opt/mssql/mssql.conf` config.

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add node**

3. First step is **Node configuration**.

    1. **Port** of your target database node. It is set to `1433` and␣
↪currently, this is disabled.

    2. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    3. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

4. Click **Continue**.

5. Third step is **Add node**.

    1. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target node. If you intend to add␣
↪another one, this will just overwrite what was previously inputted and␣
↪added since it only allows one node at a time.

6. Click **Continue** to proceed the last step **Preview**.

7. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

8. Click **Finish** button to deploy the job.
```

### ClusterControl CLI

Add a new secondary node to an existing Always On availability group on cluster ID 7. Only hostname or FQDN is accepted, not an IP address:

```
s9s cluster \
    --add-node \
    --cluster-id=7 \
```

(continues on next page)

(continued from previous page)

```
    --nodes="mssql://dsa-mssql-db2:1433" \
    --wait
```

### 2.69.9 ClickHouse

The following versions are supported for adding a node:

- ClickHouse 24.8, 25.3

ClickHouse organizes data nodes into **shards** (for horizontal write/storage scaling) and **replicas** within each shard (for read scaling and high availability), coordinated by **ClickHouse Keeper** (an embedded replacement for ZooKeeper). From Cluster Management, ClusterControl offers two wizards for adding a node to an existing ClickHouse cluster:

- **Data node**, which can add either a replica to an existing shard (for read scaling and high availability) or a new shard (to scale out storage and write throughput horizontally).
- **Keeper node**, to grow the Keeper ensemble for quorum (recommended in odd numbers, e.g. 3 or 5, for HA).

**Note**

Note The **Create a data node** wizard (see below) and the equivalent `s9s cluster --add-node` command add the new node as a replica in the existing shard.

#### Default configuration

ClusterControl deploys ClickHouse nodes with the following configuration:

- Only SSL/TLS-secured ports are enabled; the plaintext native protocol and HTTP ports are disabled. The secure native protocol port is `9440`.
- Interserver communication (used for replication between nodes) uses HTTPS on port `9010`.
- A standalone (single-node) cluster's embedded ClickHouse Keeper runs inside the same `clickhouse-server` process, listening for secure client connections on port `9281` and using port `9234` for its internal Raft consensus protocol.
- A dedicated Keeper node added later (see below) instead runs the standalone `clickhouse-keeper` package/service, with its own config under `/etc/clickhouse-keeper/`, separate from a data node's `/etc/clickhouse-server/`.
- Each node's `<macros>` (`shard`, `replica`, `cluster`) are configured automatically by ClusterControl so that `ON CLUSTER` DDL statements and `Replicated*` table engines resolve correctly.

The **Create a data node** and **Create a keeper node** wizards share the same three-step layout; only the default port and modal title differ.

**Note**

Note For high availability, deploy Keeper nodes in odd numbers (3 or 5). An even number of Keeper nodes does not improve fault tolerance.

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target ClickHouse cluster and click **Actions…**.
2. Choose **Add new**, then either:
   - **Data node**, to add a ClickHouse data node. Opens the **Create a data node** wizard.
   - **Keeper node**, to add a node to the Keeper ensemble. Opens the **Create a keeper node** wizard.
3. First step is **Node configuration**:
   - **Port** (required): the port the new node listens on. Defaults to `9440` for a data node, `9281` for a Keeper node, matching ClusterControl's default secure ports for each.
   - **Data directory** (optional): custom path for the node's data directory.
   - **Install software**: **On** by default. Disable if the package is already installed.
   - **Security configuration**: **Disable firewall** and **Disable SELinux/AppArmor**, both checked by default.
4. Click **Continue**.
5. Next step is **Add node**.
   1. In the **Node** field, type the IP address or hostname of your target node and click **+** to add it. Once added, it appears as a card showing the address; a green checkmark confirms it was validated.
   2. Optionally, expand the card and fill in **Data IP**, to use a separate IP address for the node's data traffic.
6. Click **Continue** to proceed to **Preview**.
7. **Preview** shows a read-only summary of the values you chose: **Node configuration** (Port, Install software), **Security configuration** (Disable firewall, Disable SELinux/AppArmor), and **Node** (the address and port of the node being added).
8. Click **Finish** to deploy the job.

### ClusterControl CLI

Unlike other cluster types on this page, adding a ClickHouse data node requires the `clickhouse://` scheme prefix on the target address; without it, the job fails immediately with `No ClickHouse data nodes specified in the job`.

Add a data node (as a new replica) to ClickHouse cluster ID 4:

```
s9s cluster \
    --add-node \
    --cluster-id=4 \
    --nodes="clickhouse://10.10.15.12:9440" \
    --wait
```

Removing a ClickHouse node does **not** need the `clickhouse://` prefix; plain `host:port` works:

```
s9s cluster \
    --remove-node \
    --cluster-id=4 \
    --nodes="10.10.15.12:9440" \
    --uninstall \
    --wait
```

**Warning**

Attention

The CLI has no equivalent for adding a **Keeper** node: `--nodes="keeper://..."` is rejected outright with `The protocol 'keeper' is not supported.`. Use the RPC API tab below (which is also what the GUI wizard uses) instead.

### ClusterControl RPC API

The `s9s` CLI commands above are a thin wrapper over the `createJobInstance` operation on `/v2/jobs`. Captured directly from the CLI with `--print-request`, add a ClickHouse data node with:

```
curl -k -XPOST \
-d '{
   "operation": "createJobInstance",
   "cluster_id": 4,
   "job": {
     "class_name": "CmonJobInstance",
     "title": "Add Node to Cluster",
     "job_spec": {
       "command": "addnode",
       "job_data": {
         "node": {
           "class_name": "CmonClickHouseHost",
           "hostname": "10.10.15.12",
           "port": 9440
         },
         "install_software": true,
         "disable_firewall": true,
         "disable_selinux": true
       }
     }
   },
   "authenticate": {
     "username": "myrpcadmin",
     "password": "mySuperStr0ngP@ssw0rd999"
   }
}' \
https://localhost:9501/v2/jobs
```

!!! info

```
`"class_name": "CmonClickHouseHost"` on the `node` object is what tells the␣
↪controller this is a ClickHouse data node; this is the same distinction␣
↪the CLI's `clickhouse://` prefix maps to internally.
```

Add a **Keeper** node instead by setting `"nodetype": "clickhouse_keeper"` on the node object:

```
curl -k -XPOST \
-d '{
   "operation": "createJobInstance",
   "cluster_id": 4,
   "job": {
     "class_name": "CmonJobInstance",
     "title": "Create a keeper node",
     "job_spec": {
       "command": "ADDNODE",
       "job_data": {
         "install_software": true,
         "disable_selinux": true,
         "disable_firewall": true,
         "nodes": [
           {
             "class_name": "CmonClickHouseHost",
             "hostname": "10.10.15.12",
             "hostname_internal": "",
             "port": 9281,
             "nodetype": "clickhouse_keeper"
           }
         ]
       }
     }
   },
   "authenticate": {
     "username": "myrpcadmin",
     "password": "mySuperStr0ngP@ssw0rd999"
   }
}' \
https://localhost:9501/v2/jobs
```

Once added, the new node shows `nodetype: "clickhouse_keeper"` and `role: "follower"`. See *Nodes Management → ClickHouse-specific actions*.

Remove the node (same `job_spec.command`, now `removenode`, with a `nodes` array instead of a single `node` object). This works the same way whether the node being removed is a data node or a Keeper node:

```
curl -k -XPOST \
-d '{
   "operation": "createJobInstance",
   "cluster_id": 4,
   "job": {
     "class_name": "CmonJobInstance",
     "title": "Remove '\''10.10.15.12'\'' from the Cluster",
```

(continues on next page)

(continued from previous page)

```
    "job_spec": {
      "command": "removenode",
      "job_data": {
        "nodes": [
          { "hostname": "10.10.15.12", "port": 9440 }
        ],
        "enable_uninstall": true,
        "unregister_only": false
      }
    }
  },
  "authenticate": {
    "username": "myrpcadmin",
    "password": "mySuperStr0ngP@ssw0rd999"
  }
}' \
https://localhost:9501/v2/jobs
```

## 2.70 Add Replica Node

In ClusterControl, scaling an existing cluster to add more nodes for high availability can be done by adding a replica. Adding a replica in ClusterControl is a common task for improving the resilience, read scalability, and availability of your database systems.

ClusterControl simplifies this process significantly eliminating hassle and manual work. This page will guide you add replica node based on the database cluster that ClusterControl supports.

### 2.70.1 Support Matrix

The following database cluster types, vendors and topology are supported:

| Database | Vendor | Topology |
|---|---|---|
| MySQL | Percona, Percona Server for MySQL Pro, Oracle | Standalone, replication |
| MariaDB | MariaDB | Standalone, replication |
| Galera Cluster | MariaDB, Percona | Galera certification-based replication |
| PostgreSQL | PostgreSQL, EnterpriseDB | Standalone, streaming replication |
| TimescaleDB | TimescaleDB | Standalone, streaming replication |
| Redis | Redis, Valkey | Sentinel, cluster |

### 2.70.2 Prerequisites

Same as deploying a new cluster, nodes needs to be provisioned based on the requirements that ClusterControl requires in order to have a deployment for adding node successful. See list below:

- Make sure the target database nodes are running on a supported architecture platform and operating system. See *Hardware* and *Operating System*.

- Passwordless SSH (SSH using key-based authentication) is configured from the ClusterControl node to all database nodes. See *SSH Key-based Authentication*.
- Make sure that the OS user you setup for passwordless login is the same user your ClusterControl and its cluster used during deployment.
- Verify that sudo is working properly if you are using a non-root user. See *Operating System User*.

#### Percona Server for MySQL Pro - 8.0.

Make sure you have acquired the Percona Client ID and Percona Access Token.

#### PostgreSQL Entperise DB (EDB)

Make sure you have acquired a valid EDB repository token.

### 2.70.3 MySQL Replication

The following vendors and versions are supported for adding replica node:

- Oracle MySQL - 8.0.
- Percona Server for MySQL - 8.0.
- Percona Server for MySQL Pro - 8.0.
- MariaDB Server - 10.4, 10.5, 10.6 (LTS), 10.11 (LTS), 11.4 (LTS) and 11.8.

Adding a replication requires a running and healthy primary node in the course of action in order to complete the job successfully. When the job is in progress, ClusterControl will copy the data from the primary but this shall not lock the primary node since ClusterControl uses binary backup as explained below:

- For Oracle MySQL and Percona Server (either Community edition or Pro), it will use Percona Xtrabackup to stream data from its primary to the target replica node.
- For MariaDB Server, it will use Mariabackup to stream data from its primary to the target replica node.

If you are concerned of the traffic and/or the data to be copied from your Primary node is large, it is recommended to use it during low peak period or rebuild your replica node using *backup created* and taken using ClusterControl. This shall be discussed on how to rebuild from backup in the following section below.

#### Default configuration

By default, ClusterControl deploys MySQL/MariaDB replication with the following configurations:

- ClusterControl uses `netcat` or `nc` to stream data using a binary backup (`xtrabackup or mariabackup`). It uses port 9999, or if used, it will search between 9990-9998 range and will use the next port available.
- MySQL GTID with `log_slave_updates` enabled (Oracle MySQL and Percona only).
- MariaDB GTID with `log_slave_updates` enabled (MariaDB only).
- Replica node shall be configured with `read_only=ON` and `super_read_only=ON` (if supported) for protection against accidental write.
- Semi-sync replication is enabled
- `performance_schema` is enabled

- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- Since ClusterControl will copy its data from the Primary node which means users will be copied as well, ClusterControl will add one user with its IP address or hostname to grant necessary privileges for replication:

```
| Database user      | Purpose                                          |
| ------------------ | ------------------------------------------------ |
| `rpl_user`         | MySQL/MariaDB replication.                       |
```

**Note**

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template*.

**Warning**

Attention

ClusterControl sets `read_only=ON` on all slaves but a privileged user (SUPER) can still write to a slave (except for MySQL versions that support `super_read_only`).

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add new  → Replica node**

3. Choose **Create a replica node**

4. First step is **Node configuration**.

    1. Specify the **Port** of your target replica node. Default is set to␣
↪3306.

    2. **Data directory** is set to `/var/lib/mysql`. If you have prepared␣
↪a separate path or on a different volume, replace the value with your␣
↪custom path.

    3. **Configuration template** is set to `my.cnf.repl84`. If you have␣
↪your own custom template, ClusterControl will show the templates that has␣
↪been added or configured in `/etc/cmon/templates`. For more details see␣
↪[Configuration␣
↪template](../management/configuration-management.md/#configuration-template)

    4. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
```

(continues on next page)

(continued from previous page)

```
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    5. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise uncheck if any of the two needs not be disabled. If␣
↪you have uncheck **Disable SELinux/AppArmor**, make sure that you have␣
↪configured properly to allow creating files and opening sockets for the␣
↪database to run properly.

5. Click **Continue**.

6. Second step is **Advanced Settings**.

    1. **Rebuild from a backup** is disabled. If you desire to rebuild your␣
↪data instead of streaming from the Primary node, toggle it to **On**.␣
↪This shall reveal a drop-down list with all full-backup created using␣
↪Clustercontrol.

    2. **Delay the replica node** is disabled. If you desire to delay the␣
↪replica especially for data recovery purposes, switch this to **On**.

    3. **Include in LoadBalancer set (if exists)** is disabled. Switch it␣
↪to **On* if you want ClusterControl to include this new node in the␣
↪existing node load balancing set.

    4. **Semi-synchronous replication** is enabled by default. Switch to␣
↪**Off** if you only need the traditional asynchronous replication.

7. Click **Continue**.

8. Third step is **Add node**.

    1. Choose and select your **Primary node**.

    2. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target replica node. If you intend to␣
↪add another one, this will just overwrite what was previously inputted␣
↪and added since it only allows one node at a time.

9. Click **Continue** to proceed the last step **Preview**.

10. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

11. Click **Finish** button to deploy the job.
```

#### ClusterControl CLI

Add a new replica node to an existing MySQL/MariaDB replication cluster with cluster ID 1:

```
s9s cluster \
    --add-node \
    --cluster-id=1 \
    --nodes=10.10.10.15 \
    --wait
```

To rebuild the replica from an existing full backup instead of streaming from the primary (the **Rebuild from a backup** option in the GUI wizard), add `--backup-id`:

```
s9s cluster \
    --add-node \
    --cluster-id=1 \
    --nodes=10.10.10.15 \
    --backup-id=10 \
    --wait
```

**Recommended Next Steps**

- *Add Database Node*
- *Create Load Balancer*
- *Create Replica Cluster*

### 2.70.4 Galera Cluster

The following vendors and versions are supported for adding replica node:

- Percona XtraDB Cluster - 8.0.
- MariaDB Galera - 10.4, 10.5, 10.6 (LTS), 10.11 (LTS), 11.4 (LTS) and 11.8.

Adding a replication requires a running and healthy primary node in the course of action in order to complete the job successfully. A galera cluster can have multiple primary nodes so you can choose and assign which is the desired primary node to be used as a primary of the target replica node. When the job is in progress, ClusterControl will copy the data from the primary but this shall not lock the primary node since ClusterControl uses binary backup as explained below:

- For Percona XtraDB Cluster, it will use Percona Xtrabackup to stream data from its primary to the target replica node.
- For MariaDB Server, it will use Mariabackup to stream data from its primary to the target replica node.

If you are concerned of the traffic and/or the data to be copied from your Primary node is large, it is recommended to use it during low peak period or rebuild your replica node using *backup created* and taken using ClusterControl. This shall be discussed on how to rebuild from backup in the following section below.

### Default configuration

By default, ClusterControl deploys MySQL/MariaDB replication with the following configurations:

- ClusterControl uses `netcat` or `nc` to stream data using a binary backup (`xtrabackup` or `mariabackup`). It uses port 9999, or if used, it will search between 9990-9998 range and will use the next port available.
- MySQL GTID with `log_slave_updates` enabled (Oracle MySQL and Percona only).
- MariaDB GTID with `log_slave_updates` enabled (MariaDB only).
- Replica node shall be configured with `read_only=ON` and `super_read_only=ON` (if supported) for protection against accidental write.
- Semi-sync replication is enabled
- `performance_schema` is enabled
- The generated account credentials are stored inside `secrets-backup.cnf` under the MySQL configuration directory.
- Since ClusterControl will copy its data from the Primary node which means users will be copied as well, ClusterControl will add one user with its IP address or hostname to grant necessary privileges for replication:

```
| Database user      | Purpose                                          |
| ------------------ | ------------------------------------------------ |
| `rpl_user`         | MySQL/MariaDB replication.                       |
```

If you would like to customize the above configurations, modify the template base file to suit your needs before proceeding to the deployment. See *Configuration Template*.

**Warning**

Attention

ClusterControl sets `read_only=ON` on all slaves but a privileged user (SUPER) can still write to a slave (except for MySQL versions that support `super_read_only`).

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add new  → Replica node**

3. Choose **Create a replica node**

4. First step is **Node configuration**.

    1. Specify the **Port** of your target replica node. Default is set to␣
↪3306.

    2. **Data directory** is set to `/var/lib/mysql`. If you have prepared␣
```

(continues on next page)

(continued from previous page)

```
↪a separate path or on a different volume, replace the value with your␣
↪custom path.

    3. **Configuration template** is set to `my.cnf.repl84`. If you have␣
↪your own custom template, ClusterControl will show the templates that has␣
↪been added or configured in `/etc/cmon/templates`. For more details see␣
↪[Configuration␣
↪template](../management/configuration-management.md/#configuration-template)

    4. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    5. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

5. Click **Continue**.

6. Second step is **Advanced Settings**.

    1. **Rebuild from a backup** is disabled. If you desire to rebuild your␣
↪data instead of streaming from the Primary node, toggle it to **On**.␣
↪This shall reveal a drop-down list with all full-backup created using␣
↪Clustercontrol.

    2. **Delay the replica node** is disabled. If you desire to delay the␣
↪replica especially for data recovery purposes, switch this to **On**.

    3. **Include in LoadBalancer set (if exists)** is disabled. Switch it␣
↪to **On* if you want ClusterControl to include this new node in the␣
↪existing node load balancing set.

    4. **Semi-synchronous replication** is enabled by default. Switch to␣
↪**Off** if you only need the traditional asynchronous replication.

7. Click **Continue**.

8. Third step is **Add node**.

    1. Choose and select your **Primary node**.

    2. In the **Node** field, this is where you have to specify and input␣
```

(continues on next page)

(continued from previous page)

```
↪the IP address or hostname of your target replica node. If you intend to␣
↪add another one, this will just overwrite what was previously inputted␣
↪and added since it only allows one node at a time.

9. Click **Continue** to proceed the last step **Preview**.

10. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.
```

#### ClusterControl CLI

Add a new replica node to an existing Galera Cluster with cluster ID 2:

```
s9s cluster \
    --add-node \
    --cluster-id=2 \
    --nodes=10.10.10.16 \
    --wait
```

**Recommended Next Steps**

- *Add Database Node*
- *Create Load Balancer*
- *Create Replica Cluster*

### 2.70.5 PostgreSQL

The replica will be staged with data from the primary. The primary's configuration will be altered to allow the replica to join the primary. The replica server must be reachable by SSH (key-based auth.) from the controller.

The following vendors and versions are supported for a new deployment:

- PostgreSQL from postgresql.org repository - 14, 15, 16, 17 and 18.
- PostgreSQL from EDB repository - 14, 15, 16 and 17.

Adding a replication requires a running and healthy primary node in the course of action in order to complete the job successfully. When the job is in progress, ClusterControl will copy the data through streaming from the primary node by using `pg_basebackup`. `pg_basebackup` is a binary backup which does not lock the primary node. If data is huge and streaming the data to the target replica node can take time, the process can create a significant amount of I/O load and impact performance. If you are concerned of such impact and overhead from your primary, it is recommended to use it during low peak period when adding a replica.

### Default configuration

By default, ClusterControl deploys PostgreSQL instances with the following configurations:

- Configure and load the `pg_stat_statements` module.
- The WAL level is set to `replica`.
- All replica will be set with `hot_standby`.
- Configure the `cluster_name` value in postgresql.conf given if **Instance name** is provided n the GUI under **Advanced settngs**.
- ClusterControl will configure the PostgreSQL instance with SSL encryption for clients and primary-replica connections.
- The **Disable firewall** and **Disable SELinux/AppArmor** is hidden in the background but functionality is set to enabled.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | PostgreSQL streaming replication. |

### Deployment steps

### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add new  → Replica node**

3. Choose **Create a replica node**

4. First step is **Node configuration**.

    1. Specify the **Port** of your target replica node. Default is set to␣
↪3306.

    2. **Data directory** is set to `/var/lib/postgresql/<version>/main` in␣
↪Ubuntu/Debian, whereas  `/var/lib/pgsql/<version>/data` in RHEL-variant␣
↪Linux OS. If you have prepared a separate path or on a different volume,␣
↪set **Use package default for datadir** to `Off` and specify the custom␣
↪full path.

    3. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

5. Click **Continue**.
```

(continues on next page)

(continued from previous page)

```
6. Second step is **Advanced Settings**.

    1. **Include in LoadBalancer set (if exists)** is disabled. Switch it␣
↪to **On* if you want ClusterControl to include this new node in the␣
↪existing node load balancing set.

    2. **Synchronous replication** is `Off` by default. Switch to `On` to␣
↪enable synchronous replication.

    3. Specify the **Instance name** which shall be the value of␣
↪**cluster_name** in the `postgresql.conf`. Otherwise, you can leave it␣
↪empty.

7. Click **Continue**.

8. Third step is **Add node**.

    1. Choose and select your **Primary node**.

    2. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target replica node. If you intend to␣
↪add another one, this will just overwrite what was previously inputted␣
↪and added since it only allows one node at a time.

9. Click **Continue** to proceed the last step **Preview**.

10. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

11. Click **Finish** button to deploy the job.
```

### ClusterControl CLI

Add a new replica node to an existing PostgreSQL cluster with cluster ID 3:

```
s9s cluster \
    --add-node \
    --cluster-id=3 \
    --nodes=10.10.10.17 \
    --wait
```

**Recommended Next Steps**

- *Add Database Node*
- *Create Load Balancer*
- *Create Replica Cluster*

### 2.70.6 TimescaleDB

The replica will be staged with data from the primary. The primary's configuration will be altered to allow the replica to join the primary. The replica server must be reachable by SSH (key-based auth.) from the controller.

The following vendors and versions are supported for a new deployment:

- TimescaleDB - 14, 15, 16, 17 and 18.

Adding a replication requires a running and healthy primary node in the course of action in order to complete the job successfully.

When the job is in progress, ClusterControl will copy the data through streaming from the primary node by using `pg_basebackup`. `pg_basebackup` is a binary backup which does not lock the primary node. If data is huge and streaming the data to the target replica node can take time, the process can create a significant amount of I/O load and impact performance. If you are concerned of such impact and overhead from your primary, it is recommended to use it during low peak period when adding a replica.

#### Default configuration

By default, ClusterControl deploys PostgreSQL instances with the following configurations:

- Configure and load the `pg_stat_statements` module.
- The WAL level is set to `replica`.
- All replica will be set with `hot_standby`.
- Configure the `cluster_name` value in postgresql.conf given if **Instance name** is provided n the GUI under **Advanced settngs**.
- ClusterControl will configure the PostgreSQL instance with SSL encryption for clients and primary-replica connections.
- The **Disable firewall** and **Disable SELinux/AppArmor** is hidden in the background but functionality is set to enabled.
- ClusterControl will create and grant necessary privileges for the following database users:

| Database user | Purpose |
|---|---|
| `cmon` | Management and automation. |
| `cmonexporter` | Prometheus exporter for database monitoring. |
| `cmonagent` | ClusterControl query monitoring agent. |
| `cmon_replication` | PostgreSQL streaming replication. |

#### Deployment steps

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (…)

```
2. Choose **Add new  → Replica node**

3. Choose **Create a replica node**
```

(continues on next page)

(continued from previous page)

```
4. First step is **Node configuration**.

    1. Specify the **Port** of your target replica node. Default is set to␣
↪3306.

    2. **Data directory** is set to `/var/lib/postgresql/<version>/main` in␣
↪Ubuntu/Debian, whereas  `/var/lib/pgsql/<version>/data` in RHEL-variant␣
↪Linux OS. If you have prepared a separate path or on a different volume,␣
↪set **Use package default for datadir** to `Off` and specify the custom␣
↪full path.

    3. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

5. Click **Continue**.

6. Second step is **Advanced Settings**.

    1. **Include in LoadBalancer set (if exists)** is disabled. Switch it␣
↪to **On* if you want ClusterControl to include this new node in the␣
↪existing node load balancing set.

    2. **Synchronous replication** is `Off` by default. Switch to `On` to␣
↪enable synchronous replication.

    3. Specify the **Instance name** which shall be the value of␣
↪**cluster_name** in the `postgresql.conf`. Otherwise, you can leave it␣
↪empty.

7. Click **Continue**.

8. Third step is **Add node**.

    1. Choose and select your **Primary node**.

    2. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target replica node. If you intend to␣
↪add another one, this will just overwrite what was previously inputted␣
↪and added since it only allows one node at a time.

9. Click **Continue** to proceed the last step **Preview**.

10. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

11. Click **Finish** button to deploy the job.
```

#### ClusterControl CLI

Add a new replica node to an existing TimescaleDB cluster with cluster ID 4:

```
s9s cluster \
    --add-node \
    --cluster-id=4 \
    --nodes=10.10.10.18 \
    --wait
```

**Recommended Next Steps**

- *Add Database Node*
- *Create Load Balancer*
- *Create Replica Cluster*

### 2.70.7 Redis/Valkey Cluster

The following vendors and versions are supported for adding replica node:

- Redis Cluster - 6 and 7
- Valkey Cluster - 7 and 8 (only RHEL 8/9 are supported)

Adding a replication requires a running and healthy primary node in the course of action in order to complete the job successfully. When the job is in progress, ClusterControl will setup the replica running full sync or incremental resynchronization running and executing `CLUSTER MEET` and `CLUSTER REPLICATE` commands. These commands update cluster metadata and replication roles without causing the master to become unresponsive.

#### Default Configuration

By default, ClusterControl deploys Redis cluster instances with the following configurations:

- ClusterControl will configure the Redis/Valkey instances with `appendonly` enabled.
- ClusterControl will secure the instance with AUTH enabled and configure the `requirepass` and `masterauth` options.
- ClusterControl will enable TLS encryption. To access the cluster, one must use the `--tls` flag to connect.
- The configuration `maxMemory` (70% of node's RAM, rounded to the nearest power of 2) and `maxMemoryPolicy=allkeys-lru` will be set, to reduce the risk of Redis being killed by OOM.

#### Deployment steps

#### ClusterControl GUI

1. Go to the **Clusters** dashboard. Choose your target source cluster and click the ellipsis button (...)

```
2. Choose **Add new  → Replica node**

3. First step is **Node configuration**.
```

(continues on next page)

(continued from previous page)

```
    1. Specify the **Port** of your target replica node. Default is set to␣
↪6379.

    2. **Install software** is set to **On**. You can disable this if you␣
↪do not want to install the software if the package is already installed.␣
↪If set to **On**, ClusterControl will uninstall any existing package␣
↪found and install the right package and version of the database instead.

    3. **Security configuration** to either disable firewall or␣
↪SELinux/AppArmor. Leave check if you need to **Disable firewall**␣
↪(default). Leave check if you need to **Disable SELinux/AppArmor**␣
↪(default). Otherwise
       uncheck if any of the two needs not be disabled. If you have␣
↪uncheck **Disable SELinux/AppArmor**, make sure that you have configured␣
↪properly to allow creating files and opening sockets for the database to␣
↪run properly.

4. Click **Continue**.

5. Third step is **Add node**.

    1. Choose and select your **Primary node**.

    2. In the **Node** field, this is where you have to specify and input␣
↪the IP address or hostname of your target replica node. If you intend to␣
↪add another one, this will just overwrite what was previously inputted␣
↪and added since it only allows one node at a time.

9. Click **Continue** to proceed the last step **Preview**.

10. Fourth and the last step is **Preview**, this is the last step where it␣
↪shows all the values and choosen options for the ClusterControl to use␣
↪for setting up the node.

11. Click **Finish** button to deploy the job.
```

### ClusterControl CLI

Add a new replica node for an existing shard, using the `redis-replica://` scheme, to cluster ID 6:

```
s9s cluster \
    --add-node \
    --cluster-id=6 \
    --nodes="redis-replica://192.168.0.132" \
    --wait
```

**Recommended Next Steps**

- *Add Database Node*
- *Create Load Balancer*
- *Create Replica Cluster*

### 2.70.8 Post Deployment

Once a replica is added in the cluster, ClusterControl will immediately install Prometheus exporters on the database node and add it into the Prometheus monitoring list.

## 2.71 Remove Database Node (or Remove Node)

Removing a database node or removing a node helps you scale down your cluster. Scaling down a database cluster is useful when optimizing costs, reducing complexity, and maintaining an efficient infrastructure without over-provisioning resources. ClusterControl makes this straightforward providing you options to either just unregister the node in the ClusterControl, or stop the service, or stop and remove its configuration and the package installed. This is important especially if you are also going to reuse the node as a new resource for your other productivity or business use.

Removing a node does not convey its a destructive in nature such as wiping out the data or totally scrap the server itself ready for decomissioning. Removing a node is a term used here to manage the node by unregistering from ClusterControl management and services. It also offers the means of managing the configuration or service to just stop and normally terminates itself, does a graceful exit within the cluster to avoid abnormal consequences with regards to its cluster health.

**Info**

In this page, the term **Node** here refers to any node that is registered in ClusterControl such as database node, a load balancer, or even a backup verification server.

This page will guide you how to remove the node with ClusterControl.

### 2.71.1 Support Matrix

All database clusters that are supported by ClusterControl allows you to remove a node without affecting the other service. For a complete list of our supported database clusters, see below:

| Database | Vendor | Topology |
|---|---|---|
| MySQL | Percona, Oracle | Standalone, replication |
| MariaDB | MariaDB | Standalone, replication |
| Galera Cluster | MariaDB, Percona | Galera certification-based replication |
| PostgreSQL | PostgreSQL, EnterpriseDB | Standalone, streaming replication |
| TimescaleDB | TimescaleDB | Standalone, streaming replication |
| MongoDB | MongoDB, Percona, MongoDB Enterprise | Replica set, sharded cluster |
| Redis | Redis, Valkey | Sentinel, cluster |
| Microsoft SQL Server for Linux | Microsoft | Standalone, Always On availability group |
| Elasticsearch | Elastic | Single-node cluster, high availability cluster |

### 2.71.2 Prerequisites

Only those registered nodes that are imported or deployed using ClusterControl can be managed to remove a node.

### 2.71.3 Deployment steps

The recipe for removing a database node is consistent and standardized to all database clusters we support.

#### ClusterControl GUI

1. In the **Home** dashboard page, Go to the **Clusters** view and choose your target database cluster.
2. Go to the **Nodes** tab.
3. Choose your desired node to be removed and click the ellipsis button (…) under the **Actions** field or column
4. Choose **Remove node** and this shall open a new dialog view
    1. Under the **Remove node** dialog view, you have three options to choose from what actions should you desire:
        1. **Keep service running**. If you choose this action, the following actions will be performed:
            1. Node will be unregistered from ClusterControl but the service will be kept running

2. Node will be unregistered from load balancer (if applicable)

2. **Stop service and keep files untouched**. If you choose this action, the following actions will be performed:
    1. Node will be unregistered from ClusterControl and service will be stopped
3. **Stop and uninstall service**. If you choose his action, the following actions will be performed:
    1. Node will be unregistered, stopped and all software will be uninstalled

5. Click **Remove** to perform and execute the action

### ClusterControl CLI

- Remove a HAProxy node (that is running on port `9600`).

```
s9s cluster \
    --cluster-id 99 \
    --remove-node \
    --nodes="192.168.40.60:9600"  \
    --wait --log \
```

    - where `--cluster-id=99` targets the cluster having the cluster ID of 99.
    - where `--remove-node` defines the action to take
    - where `--nodes="192.168.40.60:9600"` will be the node to be removed.
    - where the `--wait` and `--log` as the name suggest to print the job logs while waiting.

- To delete multiple nodes, you can simply achieve this by running a "for" loop. For example:

```
```

```
        for i in 192.168.40.60:9600 192.168.40.61:9600 192.168.40.60:112␣
↪192.168.40.61:112;
        do
            s9s cluster --remove-node --cluster-id 99 --nodes="${i}"  --
↪print-request --print-json --wait --log;
                sleep 0.9;
                printf "0.%s" `seq 5`;
        done;
    ```

    - `--print-request` parameter means to print the HTTP request for this␣
↪action and `--print-json` will print the logs in JSON format
```

**Recommended Next Steps**

- *Add Replica Node*
- *Create Load Balancer*
- *Create Replica Cluster*
```

ClusterControl Operations Center (CC Ops-C) is a platform management designed to simplify and enhance the management of database clusters, providing a single interface for multiple ClusterControl instances (controllers) for handling complex database operations and ensuring the reliability, performance, and security of database systems. You can use the ClusterControl Operation center to handle large-volume environments of 1000+ of nodes.

Horizontal scaling of ClusterControl is accomplished by registering or adding additional ClusterControl installations to the designated primary ClusterControl installation. CMON controllers are manually registered or added to this primary installation. It also provides multi-tenancy isolation where each tenant is assigned their own CMON Controller.

## 2.72 Single-Controller vs Multi-Controller

| Mode | Description |
|---|---|
| Single-Controller | Traditional setup using a single CMON controller to manage clusters. |
| Multi-Controller (Ops-C) | Ops-C manages multiple CMON controllers via the **ClusterControl Manager** (`cmon-proxy`), providing consolidated control and visibility across different environments. |

In a multi-controller architecture, each deployment can have its own isolated CMON controller, registered to a primary Ops-C instance.

**Note**

Note Starting from ClusterControl 2.5.0, the toggle for this feature in the GUI is labeled **Ops-Center** (previously **Multi-controller**).

## 2.73 Activating ClusterControl Ops-C

Since ClusterControl version 2.3.2, ClusterControl Ops-C is installed by default so you only need to enable it from the GUI.

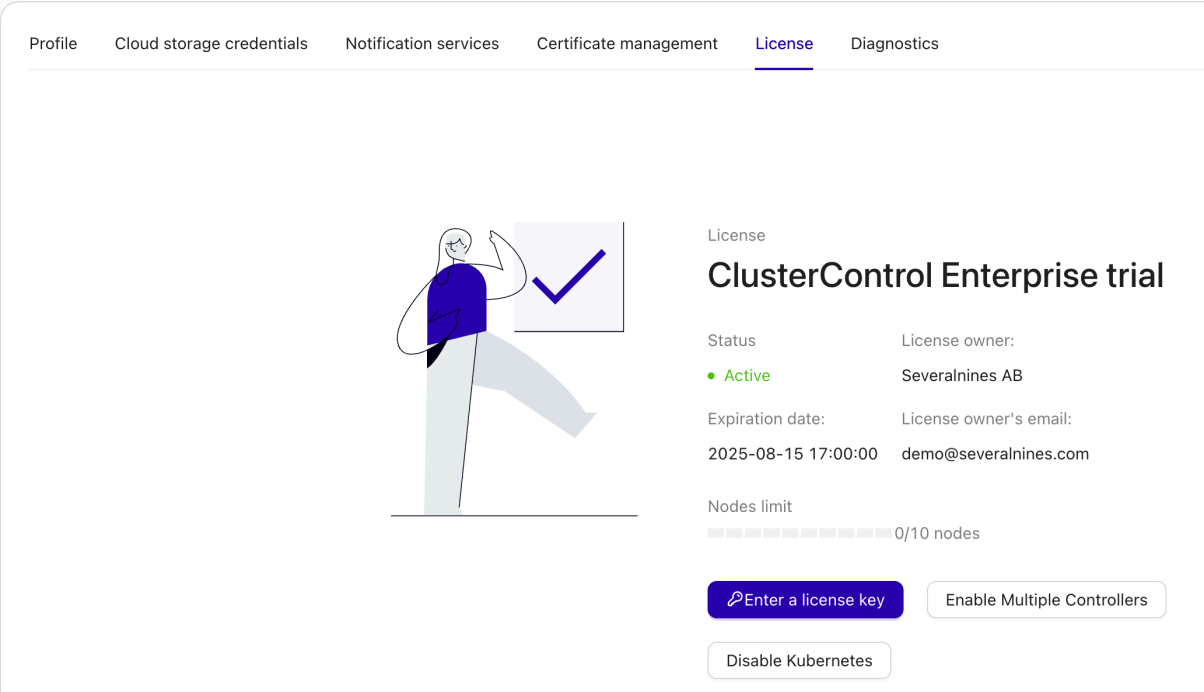
### ClusterControl 2.5.0 and later

1. Go to ***ClusterControl GUI → Settings → Addons*** and toggle on **Ops-Center**. This opens the **Enable Ops-Center** wizard.
2. **Step 1, About:** review the feature summary, optionally following **Learn more about how Ops-Center works**, then click **Continue**.
3. **Step 2, Set root user:** enter a **Root username** and **Root user password** (email is optional), confirm the password, then click **Enable**.
4. The page reloads in multi-controller mode.

See *User Guide → Overview → Addons → Ops-Center* for details.

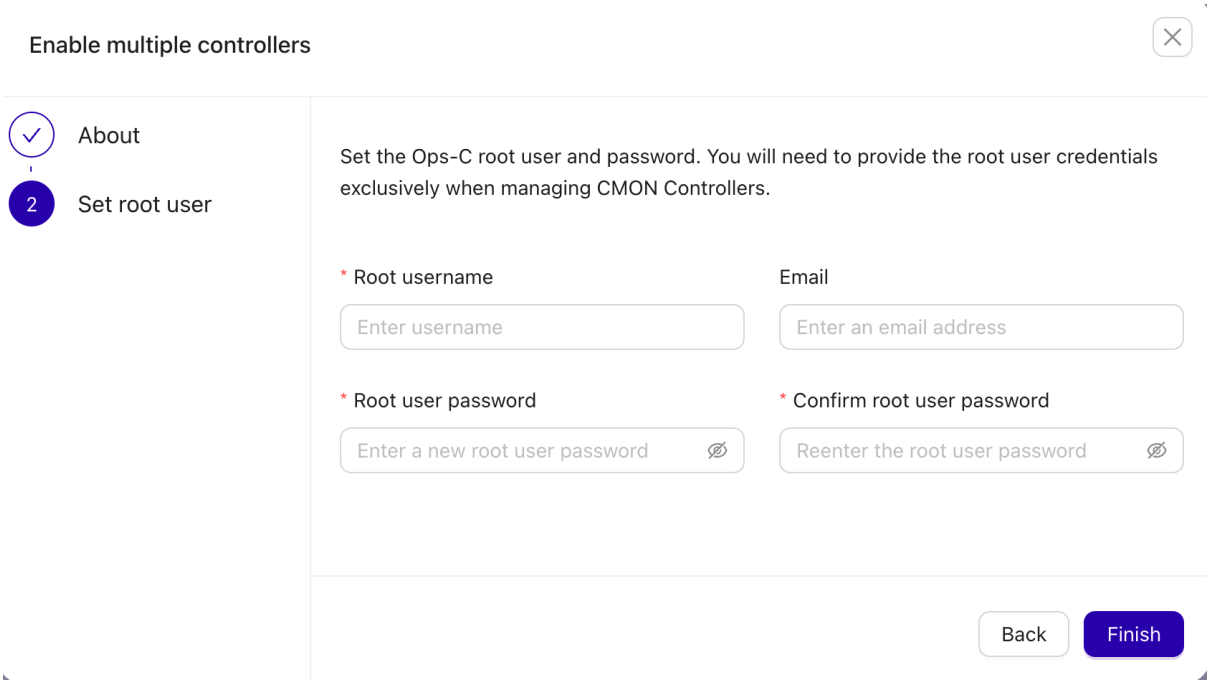
### Before ClusterControl 2.5.0

Go to *ClusterControl GUI → Settings → License* and click **Enable Multiple Controllers**.

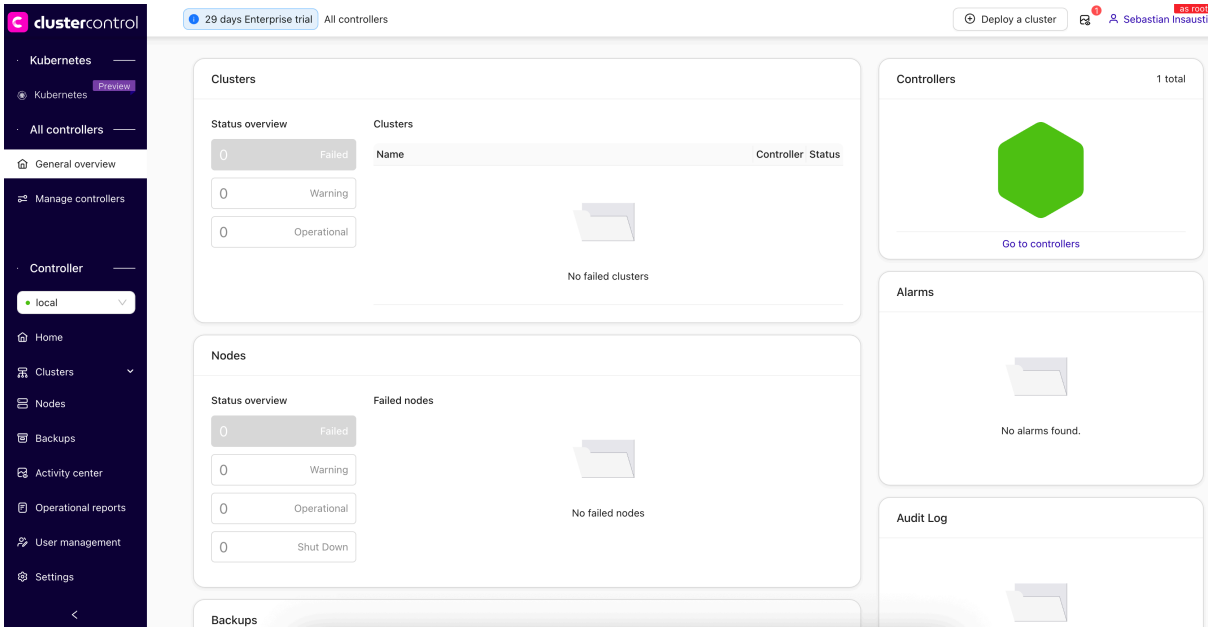



**Info**

When you enable ClusterControl Ops-C, it will ask you to create a **root** user and **password**. This root user is used only for managing CMON Controllers and serves as an extra layer of security.




After you enable it, you will see the corresponding UI.




## 2.74 Deactivating ClusterControl Ops-C

Disabling Ops-Center switches the console back to single-controller mode. Since more than one controller may be registered while Ops-Center is active, ClusterControl needs you to pick which one of the registered controllers stays as the single active controller before the switch completes.

### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Addons***.
2. Toggle off **Ops-Center**. This opens the **Disable Ops-Center** dialog, warning that the general overview across all controllers will no longer be available and that the Ops-Center root user will be deleted.
3. Under **Controller to keep**, choose the controller that should remain as the single active controller (required).
4. If more than one controller is registered, optionally check **Remove the other controllers from the configuration** to remove them now; leave it unchecked to keep them on record for re-enabling Ops-Center later without re-adding them.
5. Click **Disable**. The page reloads in single-controller mode, pointed at the controller you picked. The other registered controllers are not uninstalled or otherwise affected on their own hosts.

See *User Guide → Overview → Addons → Disable Ops-Center* for the exact dialog wording.

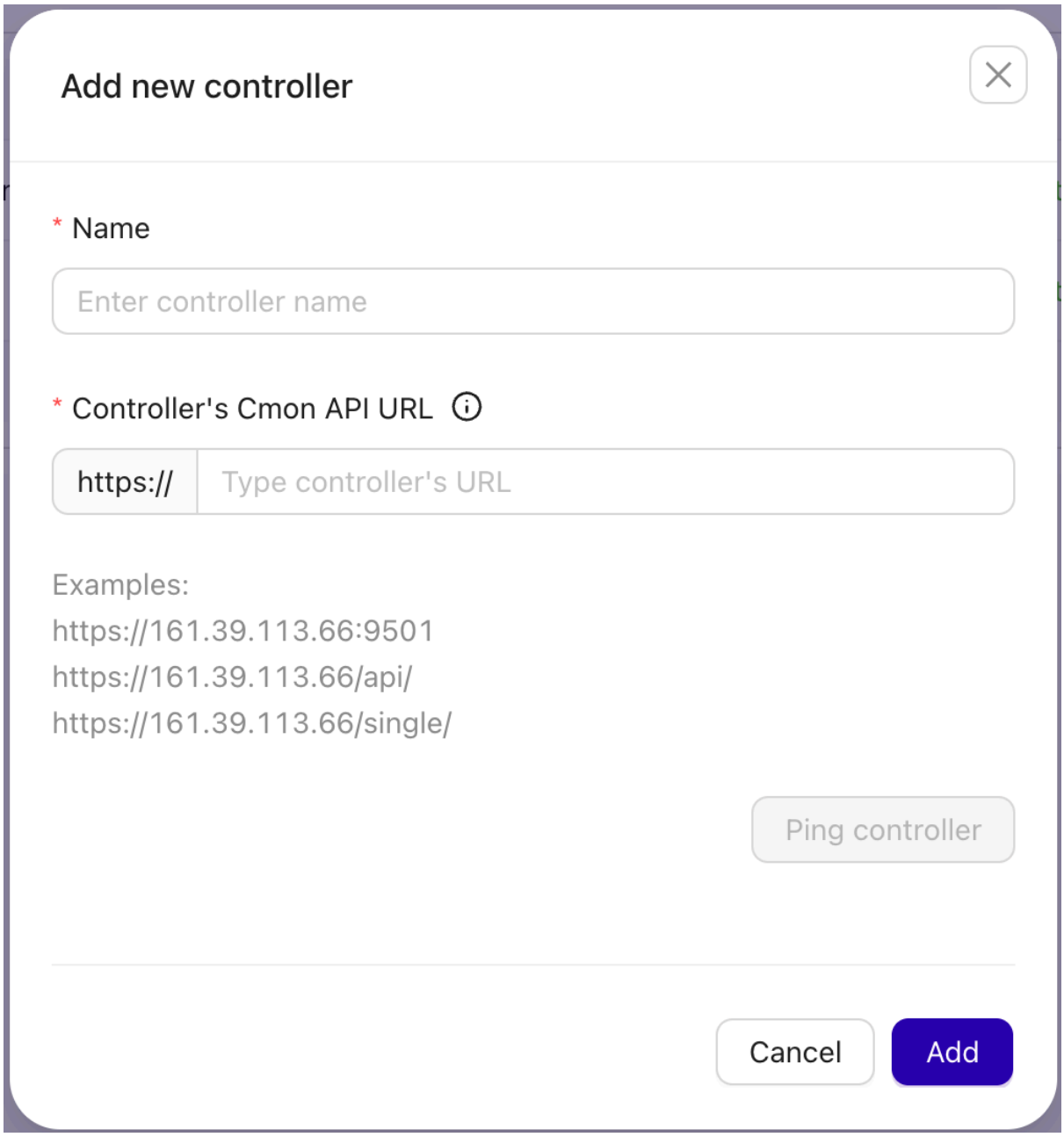
## 2.75 Controller Management

Ops-C enables full lifecycle management of CMON controllers through the web UI.

### 2.75.1 Add Controller

Adding a new ClusterControl Controller into the ClusterControl Ops-C is straightforward. Below are the instructions:

1. Navigate to Manage Controllers -> Add new controller. It will ask you to authenticate with the **root** user **password** created when you enabled ClusterControl Ops-C.
2. Provide the controller name and Controller's Cmon API URL. For example, `https://161.39.113.66/api/`
3. Test the connection by using the `Ping controller` button, and press on `Add`.
4. The new controller will appear in the controller list.



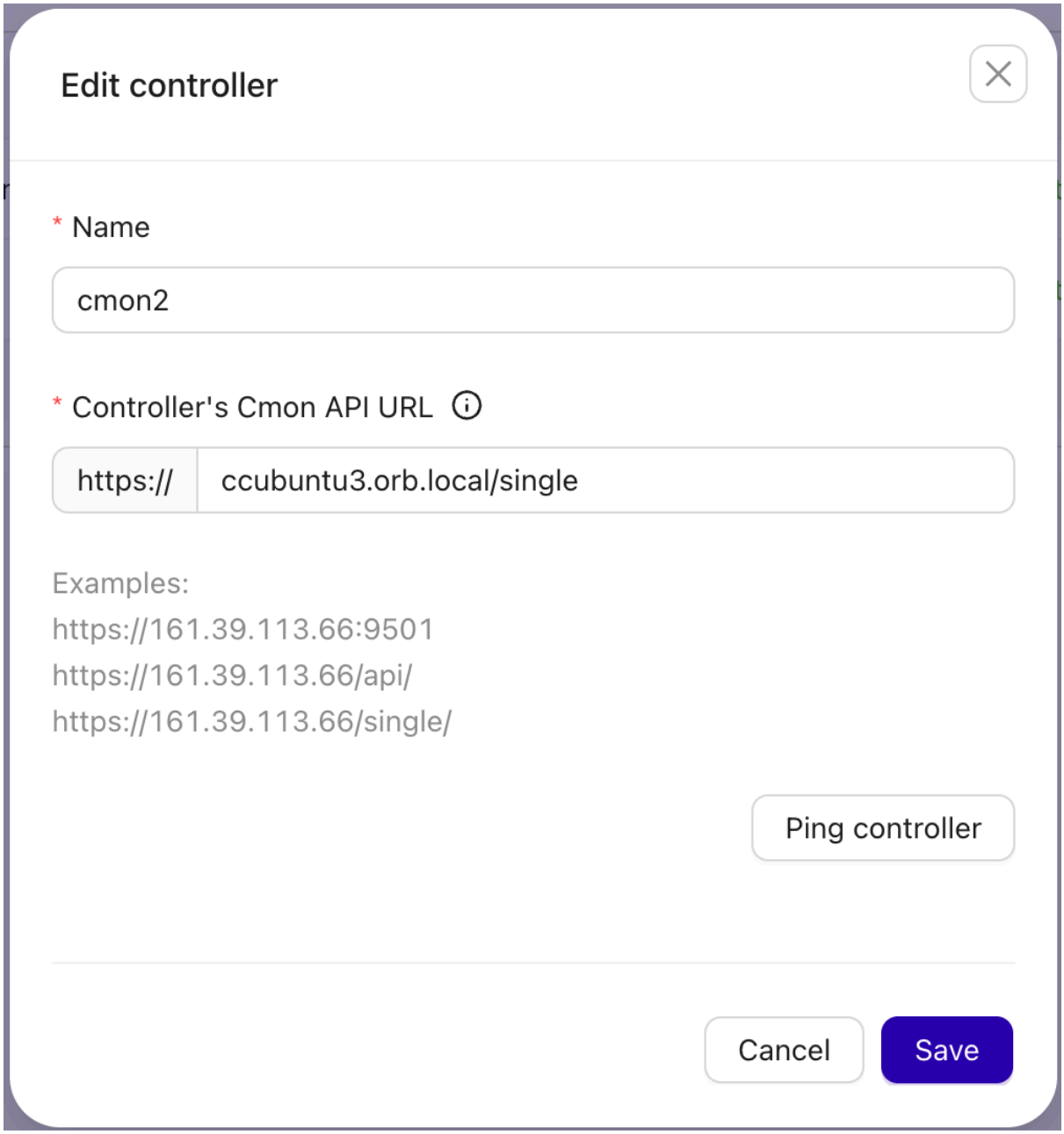
**Info**

To display aggregated data from multiple controllers, Ops-C app will attempt to log in to all registered controllers using your current credentials upon login. For consistent access, ensure this user exists on all controllers with the same credentials and permissions.

### 2.75.2 Edit Controller

To update the configuration of an existing controller:

1. Go to Manage Controllers.
2. Click the `Edit` button in the `Actions` section.
3. Modify properties such as name or API URL.
4. Save changes.




### 2.75.3 Remove Controller

To remove a controller from Ops-C:

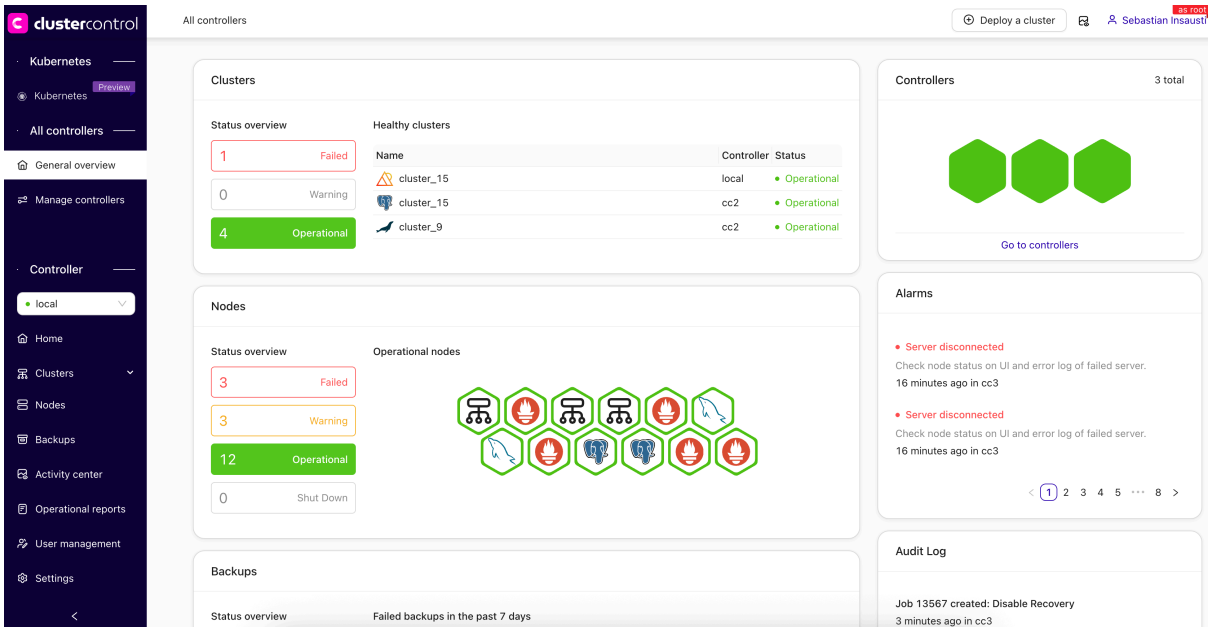
1. Navigate to Manage Controllers.
2. Click the `Remove` button in the `Actions` section.
3. Confirm removal.

The controller will be unlinked but not uninstalled from the remote host.

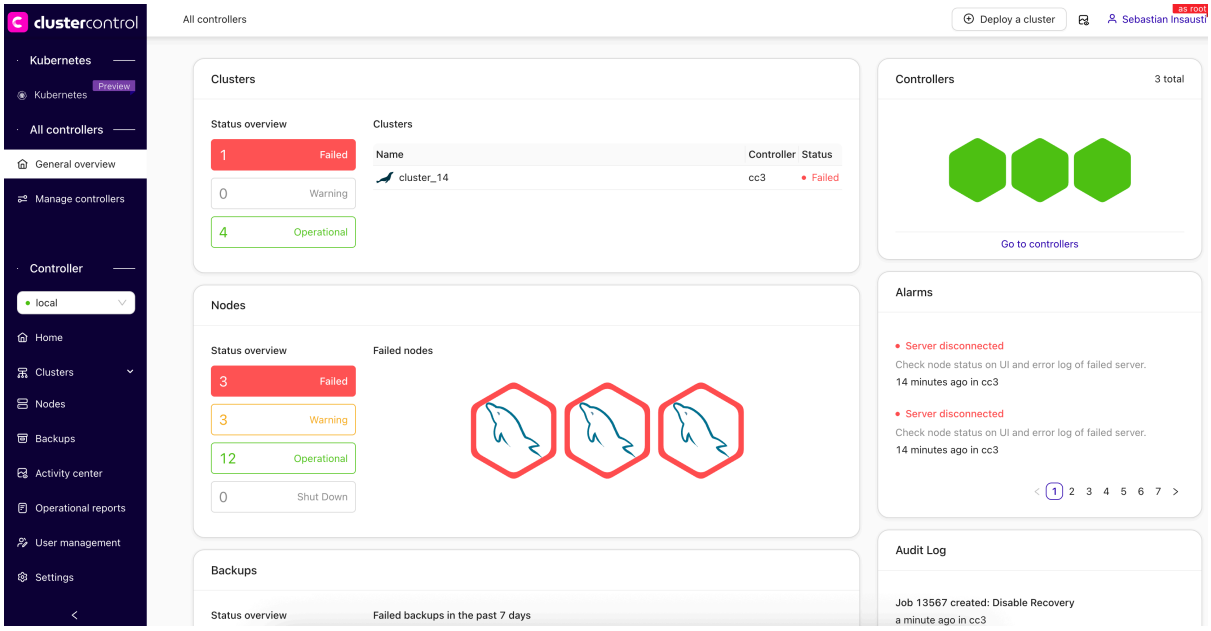
## 2.76 Monitoring Dashboard

The Controllers' Overview dashboard provides a global view of:

- Controller health and job status
- Cluster alarms and backups
- Node-level availability
- Active jobs per controller

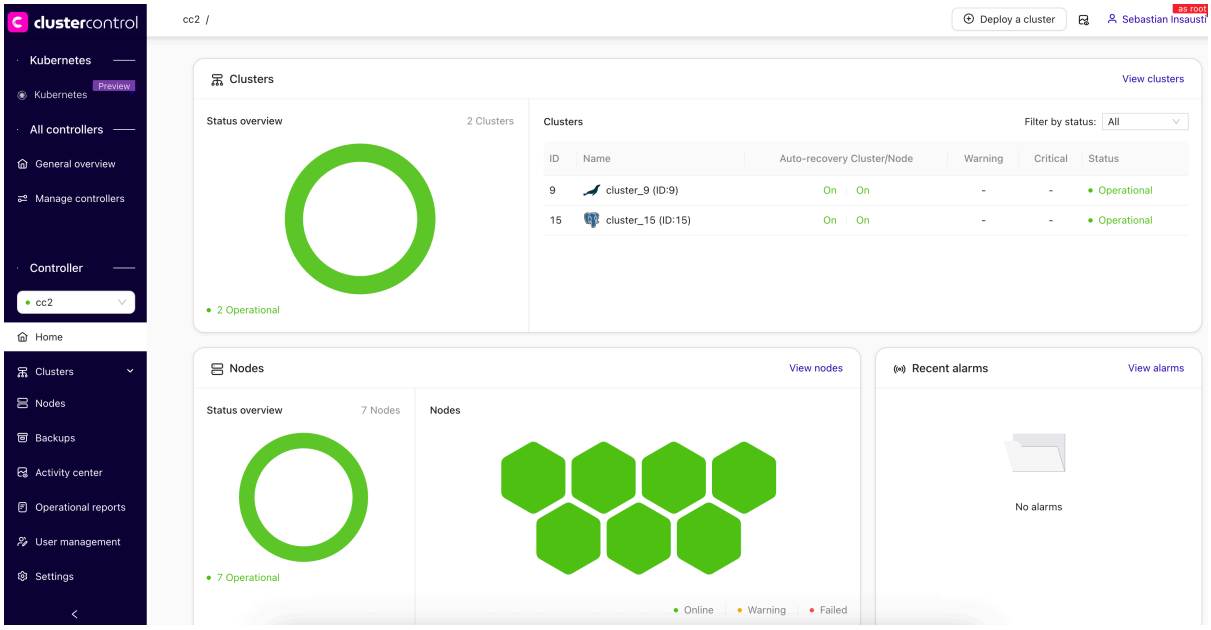



You cna filter by status Failed, Warning, Operational for both Clusters and Nodes.

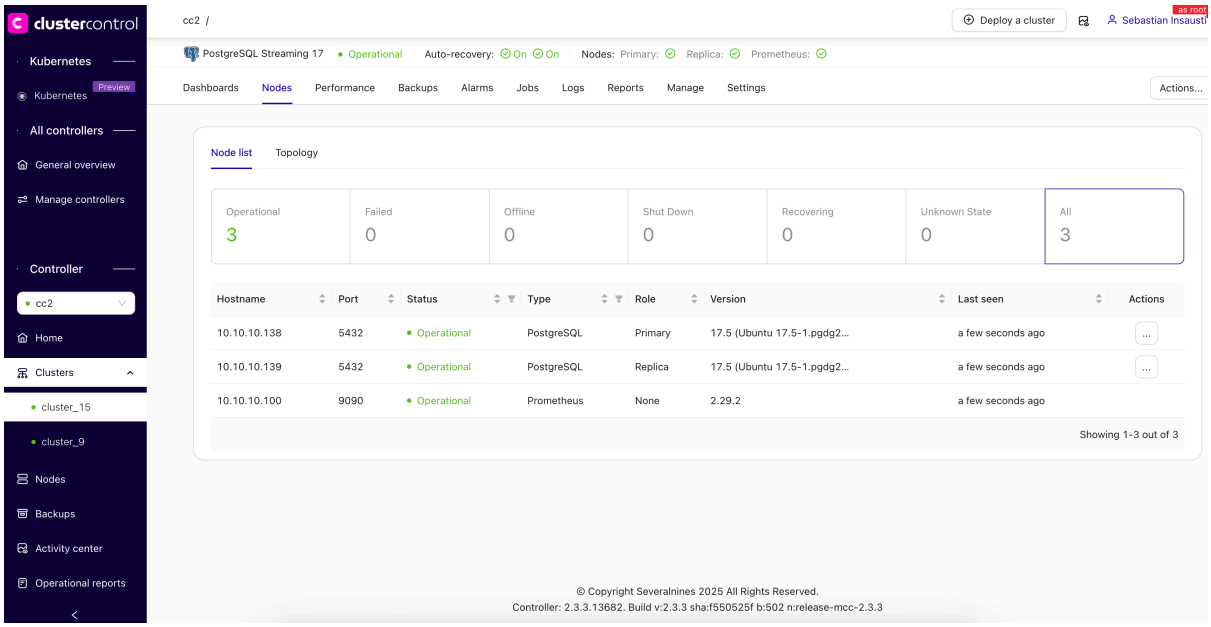



Users can switch between controllers using the Active CMON Controller selector. The application context is scoped to the selected controller for all operations outside the overview.

Discover the status of Clusters and Nodes in specific controllers as shown below:




Go through the node list for specific clusters in the selected controller:




## 2.77 ClusterControl Manager

A new process, the **ClusterControl Manager** (`ccmgr`), is responsible for managing, collecting data, and monitoring the CMON controllers in this configuration:

- The Controllers' Overview page features a dashboard that presents aggregated information on registered or managed CMON controllers, providing insights into the status of controllers, ongoing jobs, alarms, and backups, as well as the status of the database clusters and nodes.
- The web user interface allows users to select the "Active CMON Controller", which determines the information displayed throughout the application (excluding the Controllers' Overview) and directs user actions for clusters or nodes to the selected CMON Controller.

**Warning**

Attention

Starting from clustercontrol-mcc 2.2.4 (July 2024), `clustercontrol-mcc` web application doesn't require an Apache server any longer. There is no dependency for the Apache server and the web application is served by the `ccmgr` process (systemd unit is called `cmon-proxy`) instead. So, with an existing ClusterControl environment/installation, the Apache server and the CCv2 UI can be completely removed prior to installing ClusterControl Ops-C.

Functions of **ClusterControl Manager** (`ccmgr`) include:

- Routing API and UI traffic
- Aggregating data from multiple controllers
- Handling multi-tenant authentication
- Exposing Ops-C UI on a configurable port (default: 19052)

Ensure the service is active:

```
systemctl status cmon-proxy
```

To restart after changes:

```
systemctl restart cmon-proxy
```

ClusterControl allows DevOps and SRE teams to declaratively deploy, scale, tune, and backup open-source databases with Kubernetes database operators and ClusterControl's integration with Argo CD and GitHub-based GitOps workflows.

There are two primary methods for managing the database cluster on Kubernetes with ClusterControl:

1. Direct database cluster management using supported Kubernetes operators via the ClusterControl GUI.
2. The GitOps approach. See *GitOps* for details.

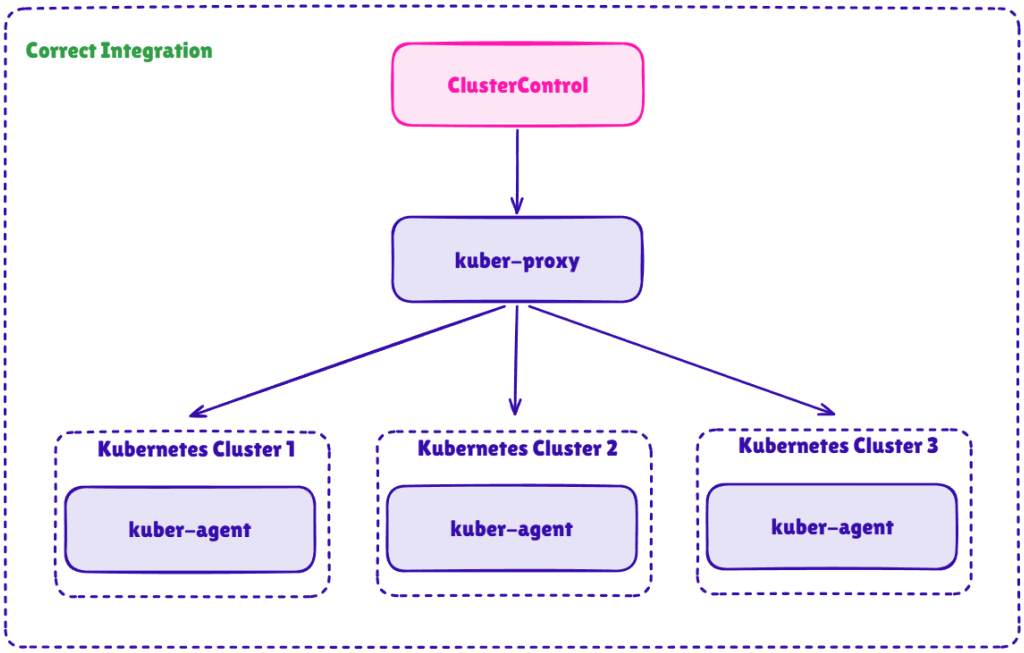
**Note**

Note This feature can be turned on or off from ***ClusterControl GUI → Settings → Addons*** (ClusterControl 2.5.0 and later) or ***ClusterControl GUI → Settings → License*** on earlier versions. See *User Guide → Overview → Addons*.

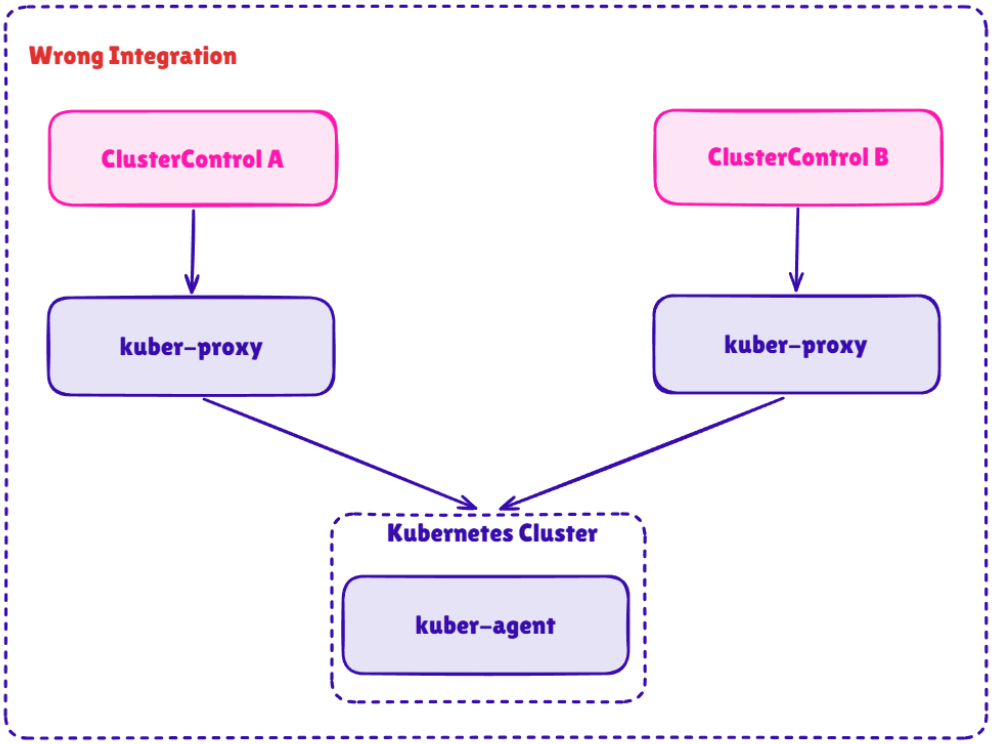
## 2.78 ClusterControl Kubernetes proxy

Integration with the Kubernetes operator is managed through a new process called kuber-proxy. This proxy is installed alongside ClusterControl Ops-C (the new GUI replacing ClusterControl GUI v2, starting from version 2.3.3) using the package name clustercontrol-kuber-proxy. The service is managed by systemd under the unit file kuber-proxy.

Each ClusterControl instance has one dedicated kuber-proxy, which acts as a gateway that accepts connections from multiple Kubernetes clusters. Each Kubernetes cluster runs its own kuber-agent, which connects back to the kuber-proxy on the ClusterControl side. The following diagram shows the correct integration:




However, the opposite is not supported, a single kuber-agent cannot connect to more than one kuber-proxy. If you have two separate ClusterControl instances, each with its own kuber-proxy, a Kubernetes cluster can only register to one of them.



**Warning**

Attention

```
The ClusterControl GUI communicates directly with the Kubernetes API server␣
↪(via the Kubernetes proxy) to manage database clusters deployed on␣
↪Kubernetes. These clusters exist exclusively within the Kubernetes␣
↪environment and are not managed or tracked by the ClusterControl␣
↪Controller (CMON). As a result, ClusterControl components that depend on␣
↪CMON, such as the ClusterControl CLI, RPC API, and Terraform Provider,␣
↪cannot be used to manage these Kubernetes-native database clusters.
```

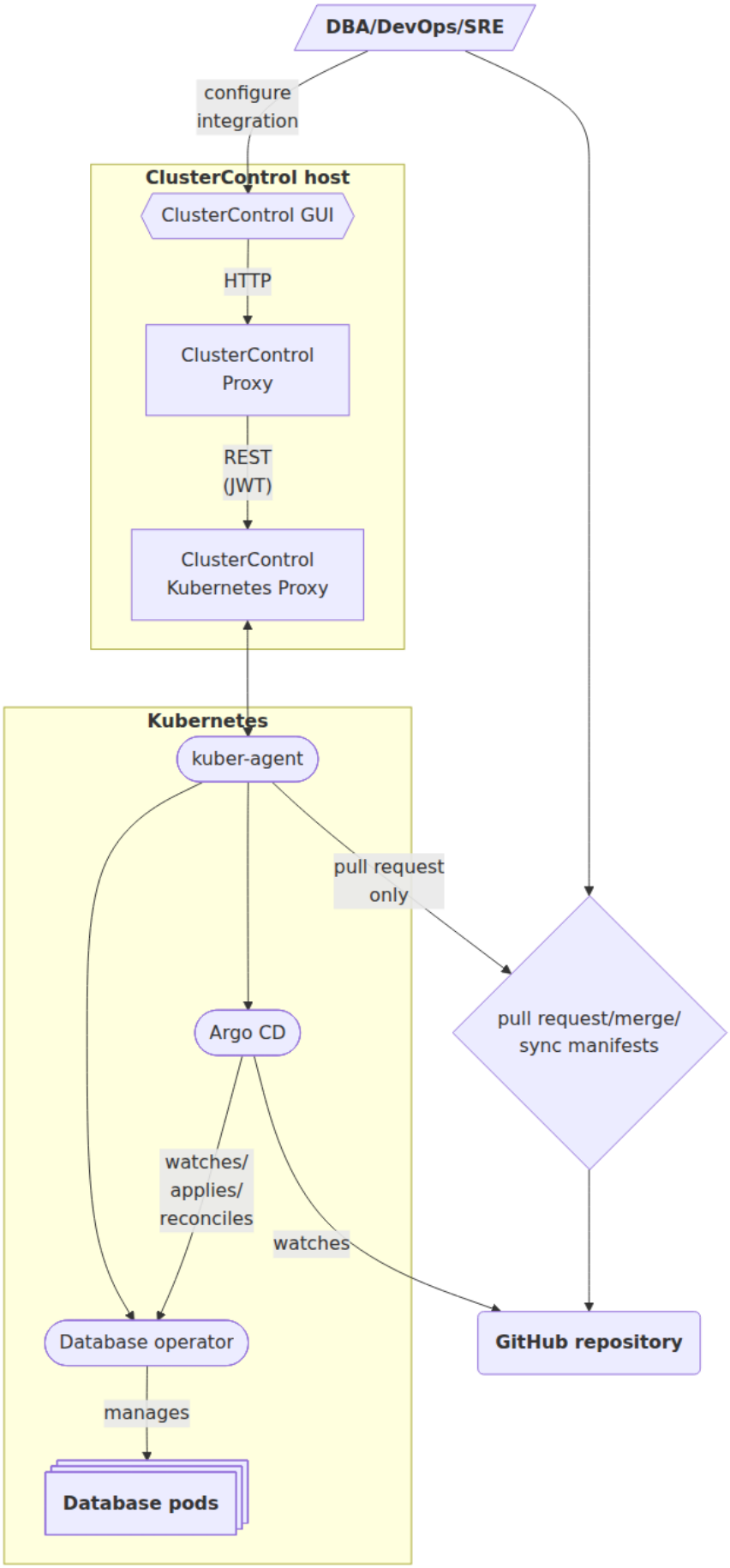
## 2.79 GitOps

ClusterControl serves as the operational layer that sits above Kubernetes and Argo CD. It facilitates traceable database operations by transforming declarative GitOps workflows. This approach establishes a modern operational model where Git is the single source of truth for declaratively managing applications and infrastructure. Configurations are version-controlled in Git and continuously reconciled with the live infrastructure.

The core principles of GitOps include:

- Declarative infrastructure using YAML.
- Versioned source of truth (Git).
- Automated reconciliation (e.g., via Argo CD).
- Continuous delivery achieved through Pull Requests.
- Security and governance inherited from the Git workflow.

The following flowchart illustrates the workflow and how the components communicate if GitOps integration is enabled:


DBA/DevOps/SRE
configure integration
ClusterControl host
ClusterControl GUI
HTTP
ClusterControl Proxy
REST (JWT)
ClusterControl Kubernetes Proxy
Kubernetes
kuber-agent
pull request only
pull request/merge/ sync manifests
Argo CD
watches/ applies/ reconciles
watches
Database operator
GitHub repository
manages
Database pods

GitOps-managed operations cover more than initial cluster provisioning:

- **Database cluster deployments** are proposed as a Git pull request instead of being applied directly to the cluster; review and merge the manifests in your repository before they take effect.
- **Backup schedules** follow the same pull-request flow, so backup policy changes are reviewed and versioned like any other change.
- **Add-on changes** made in the deployment wizard (see *Addons*) are also proposed through Git rather than applied directly.
- **Secrets** referenced by the generated manifests are handled safely within the Git workflow, and cluster configuration and certificates are stored as Kubernetes Secrets rather than local config files.
- **On-demand sync**: use the **Sync from Git** action in the ClusterControl GUI to pull the latest state from the repository on demand, instead of waiting for the next automatic reconciliation.

### 2.79.1 Argo CD integration

Argo CD is a GitOps-based, declarative continuous delivery tool specifically for Kubernetes. It automates the deployment of desired application states into specified target environments. Deployment updates can be tracked via branches or tags, or pinned to a specific Git commit version of the manifests. Argo CD mandates the use of Git repositories as the source of truth for defining the desired application state.

ClusterControl offers integrated support for Argo CD by providing:

- Automatic Installation: Simple deployment and configuration of Argo CD within the target Kubernetes environment.
- External Integration: The ability to connect to pre-existing Argo CD instances, ensuring continuity of established CI/CD and governance policies.

### 2.79.2 GitHub integration

ClusterControl integrates with GitHub using Fine-Grained Personal Access Tokens (PATs) for secure, least-privilege authentication.

These tokens restrict access to specific repositories and operations, minimizing exposure. To create a fine-grained PAT:

1. Go to ***GitHub → Settings → Developer Settings → Personal Access Tokens → Fine-grained tokens***.
2. Click **Generate new token**.
3. Under **Resource owner**, select your user or organization.
4. Under **Repository access**, choose **Only select repositories** and pick your GitOps repository.
5. Under Permissions, enable:
    - Contents: Read and write
    - Metadata: Read
    - Pull requests: Read and Write
6. Set an expiration (recommended: 90 days or less).
7. Click **Generate token** and copy it.
8. Paste the token in ***ClusterControl GUI → Kubernetes → GitOps Integration → Repository Settings*** to authenticate securely.

**Tip**

Store tokens as Kubernetes Secrets or integrate with HashiCorp Vault. Rotate regularly and assign repository-specific permissions only.

## 2.80 Deploy your first Kubernetes cluster

To set up your first database in Kubernetes using ClusterControl, follow the following steps.

### 2.80.1 1. Install ClusterControl Ops-C

ClusterControl Ops-C is now installed by default (it is the default ClusterControl GUI, replacing ClusterControl GUI v2) so you just need to follow the instructions in the **Installation section** to install it. See *Online Installation*.

Check that the `kuber-proxy` process is reachable on port 50051 (default) either using a public or private address depending on your setup (public managed Kubernetes cluster versus on-premise, for example).

### 2.80.2 2. Install the required tools

The kubectl tool is required to use this feature. Install it on the ClusterControl host and make sure `kubectl` is installed in the `PATH`.

### 2.80.3 3. Create a Kubernetes cluster

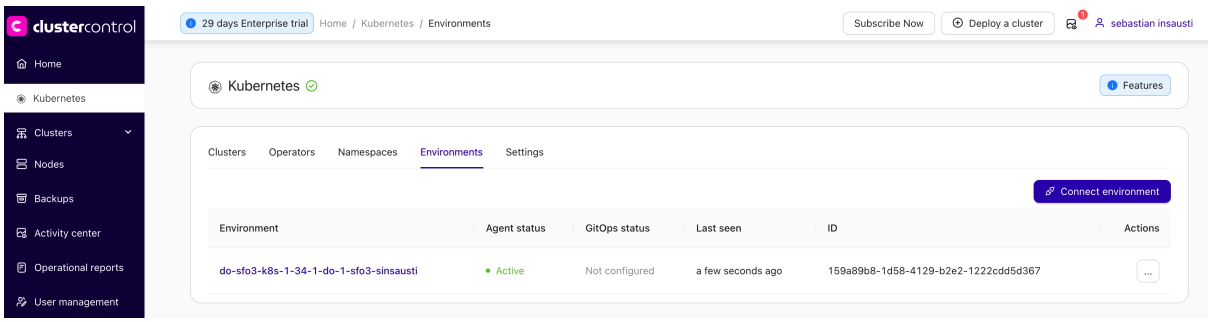
You must create a Kubernetes Cluster if you do not already have one.

The cluster can be a self-hosted on-premise installation, a public managed service, or a development environment like Docker Desktop or OrbStack. The only requirement is that the Kubernetes cluster can successfully connect to the kuber-proxy process on port 50051 of the ClusterControl host.

### 2.80.4 4. Connect to a Kubernetes cluster

After creating the Kubernetes cluster (or if it was already created), you have to connect to the environment by installing the agent-operator. The Kubernetes agent-operator establishes the connection to your Kubernetes resources. It acts as a bridge between ClusterControl and the Kubernetes environment, allowing you to monitor and perform operations directly through ClusterControl.

The Kubernetes integration is accessible from a new dedicated page, **Kubernetes**, where you can manage all things related to Kubernetes like your database clusters, backups, database operators and connections to different Kubernetes clusters.




You can add any Kubernetes cluster whether it is an on-premise self-hosted, a public managed service, Docker Desktop or OrbStack development environment as long as the agent-operator is able to connect

to the kuber-proxy process using the provided GRPC address to the ClusterControl host where the proxy process is running. The default port that kuber-proxy is listening on is 50051.

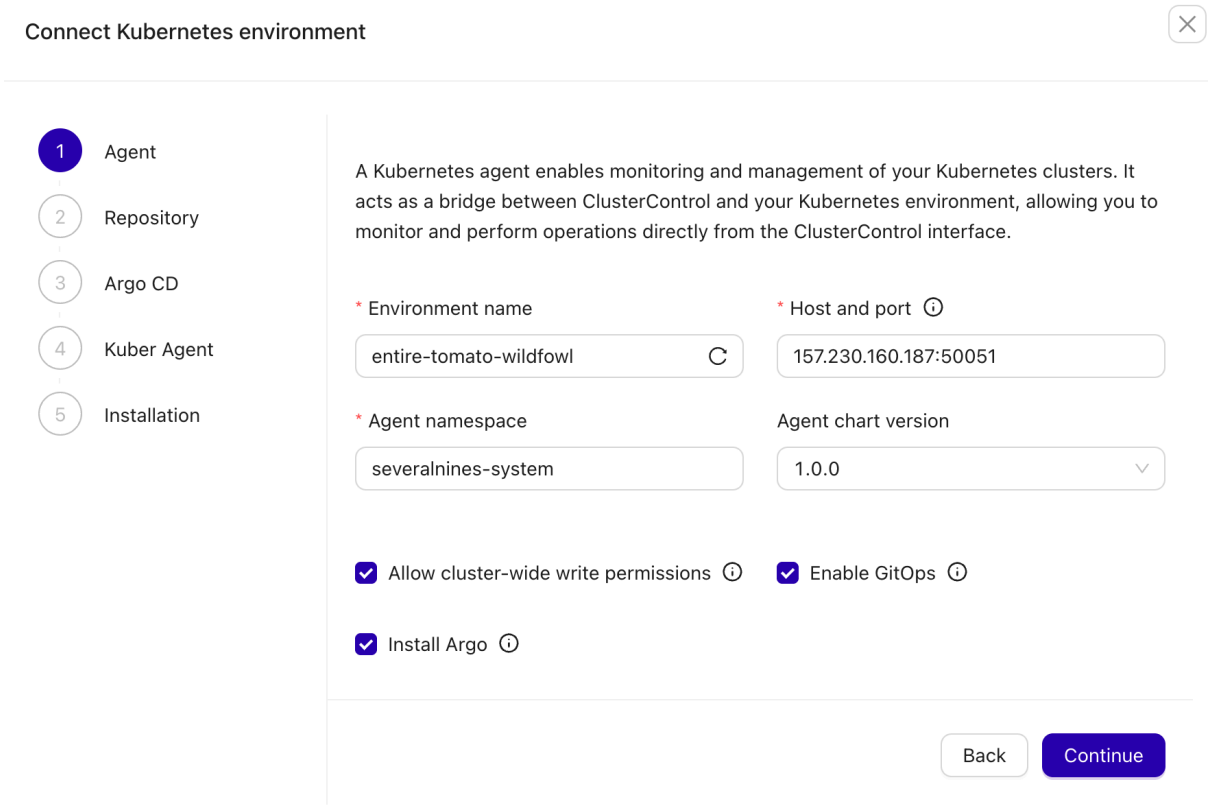
You can check which Kubernetes cluster that you are currently established with:

```
kubectl config current-context
```

After you have verified that this is the Kubernetes cluster that you want to use, then connect this Kubernetes cluster with ClusterControl by clicking on **Connect environment**.

### 2.80.5 5. Configure the agent

Begin by specifying the environment details.



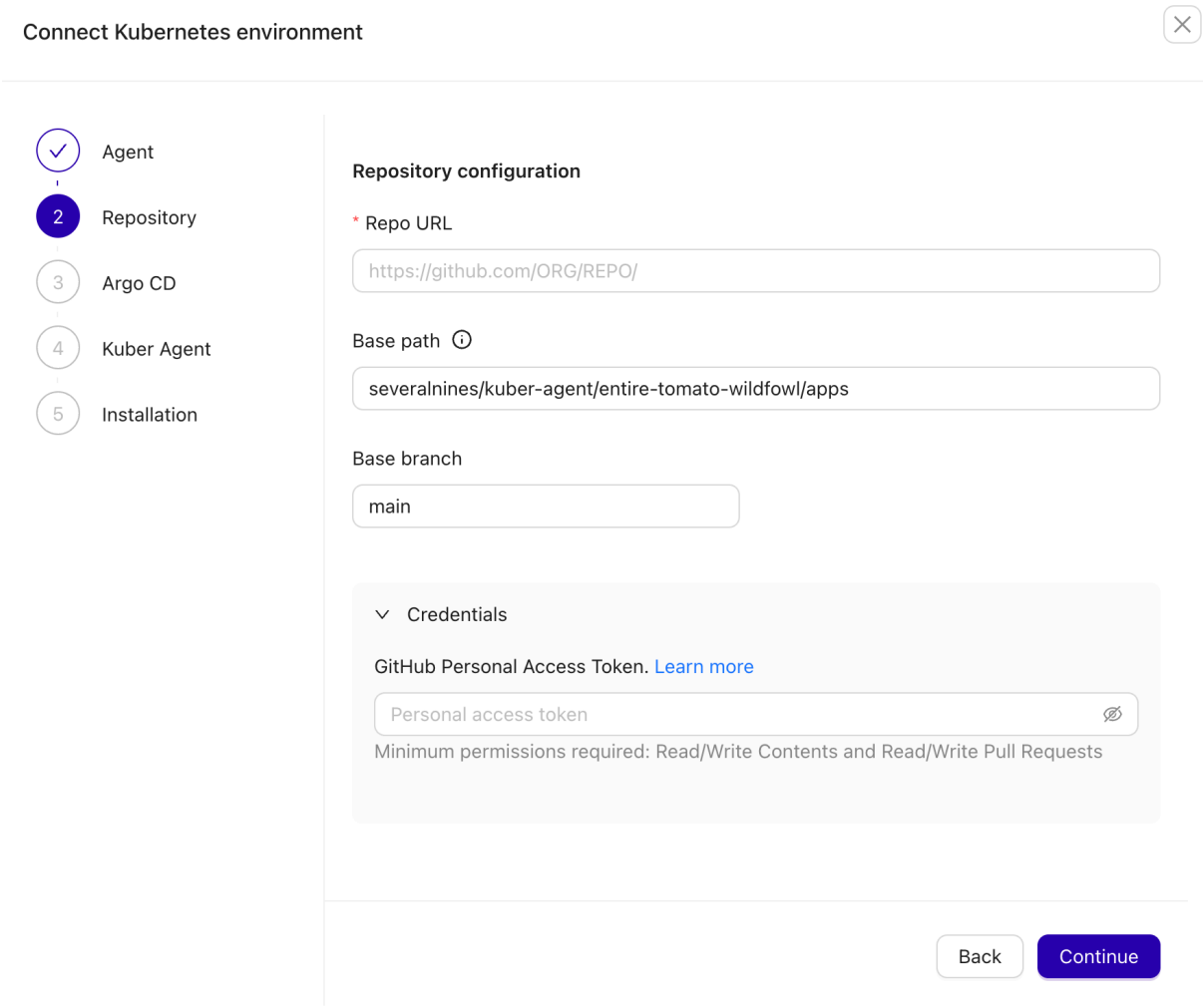

In this section, you have to complete the following information:

- **Environment name**: Enter an environment name.
- **Host and port**: ClusterControl Kubernetes Proxy host and GRPC port. The Kubernetes cluster must be able to reach this endpoint.
- **Agent namespace**: Specify the agent's namespace (e.g., severalnines-system).
- **Agent chart version**: Select the desired agent's Helm chart version (e.g., 1.0.0).
- **Allow cluster-wide write permissions**: Grants this agent permission to create, update, patch, and delete Kubernetes resources defined by the operator. Leave unchecked to confine write permissions to the namespaces you list below. Those namespaces can be used to deploy database operators and clusters.
- **Enable GitOps**: Use git repository as the single source of truth for desired cluster state.
- **Install Argo**: Install Argo CD to manage agent and database operators lifecycle.

### 2.80.6 6. Set up the repository

*Skip this step if **Enable GitOps** is unchecked in step 5. This step is for GitOps integration.*

Provide the GitHub repository URL that will store the YAML manifests. Specify the **Base path** and **Base branch**, and authenticate using your GitHub Personal Access Token (PAT) to allow ClusterControl to push configuration changes automatically.




See *GitHub integration* on step-by-step instructions on how to create a GitHub Personal Access Token (PAT).

### 2.80.7 7. Addons

==*New in 2.5.0*==

Enable addons for optional platform extensions installed alongside the agent.

- **SOPS (Secrets OPerationS)**: Backs up Kubernetes Secrets to Git, encrypted with age, so they survive cluster rebuilds without ever appearing in plaintext in the repo.
  - **Install SOPS (sops-secrets-operator)**: Installs the sops-secrets-operator into the cluster via an AppBundle. Uncheck if you already run your own operator and just want to use the Severalnines encryption pipeline against it.
  - **Create age key**: Runs a one-shot Job that generates an age keypair and stores it as a Secret in the SOPS namespace. Uncheck if you want to apply your own Secret/sops-age-key and

Secret/sops-age-pubkey before the operator starts (e.g. to restore a backed-up key).

- **Monitoring**: Install Prometheus components in this cluster or deploy a PrometheusAgent that remote-writes cluster metrics.

### 2.80.8 8. Monitoring

*==New in 2.5.0==*

*Skip this step if **Monitoring** is unchecked in step 7.*

- **External Prometheus**: Use PrometheusAgent to remote-write metrics to an existing Prometheus. This is useful if you already have a Prometheus instance running where later you can specify the remote write target details. You can also use an existing Prometheus deployed by ClusterControl but you must configure it to be remote-write receiver. When disabled, the wizard can install an in-cluster Prometheus server on the Kubernetes.

Under the **Monitoring components**, you can fine tune the Prometheus setup, as shown below:

- **kube-prometheus-stack version**: ClusterControl deploys Prometheus by using this Helm chart. Optionally, you may override the `kube-prometheus-stack` Helm chart version from the drop-down. Leave empty to use the backend's pinned default.
- **Prometheus Operator**: Toggle ON to install the Prometheus Operator and CRDs. Toggle OFF only when they already exist in this cluster.
- **Prometheus Server**: Toggle ON to install the Prometheus server component from `kube-prometheus-stack`. The Prometheus service is only reachable inside the cluster.
- **Grafana**: Toggle ON to install Grafana for monitoring dashboards.
  - **Include dashboards**: Pre-loads the ClusterControl and Terminal Grafana dashboards.
  - **Exposure**: Toggle ON to expose Grafana to the external traffic.
    - **Service type**: Choose the Kubernetes service type to forward network traffic to Grafana.
    - **NodePort**: Specify a port to allow external traffic to access a cluster by opening a specific port on each Node's IP address. Only if **Service type** is **NodePort**.

### 2.80.9 9. Prometheus Agent

*==New in 2.5.0==*

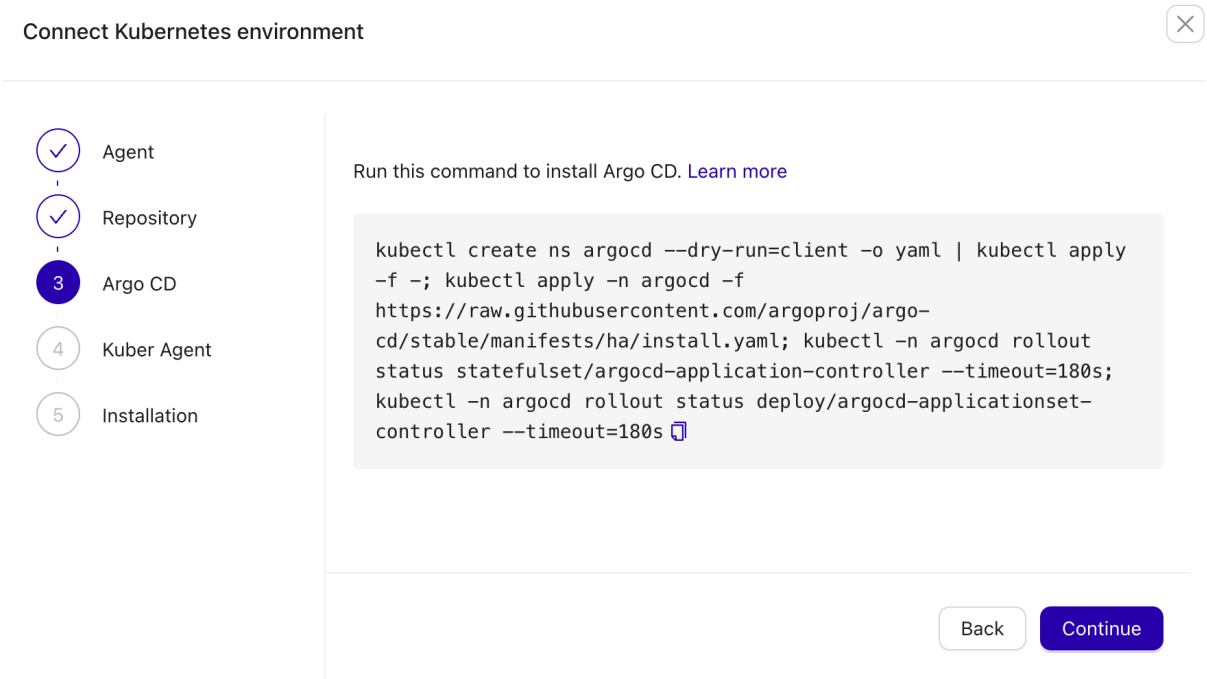
*Skip this step if **External Prometheus** is unchecked in step 8.*

- **URL**: Full URL of the receiving Prometheus, including `/api/v1/write` if your endpoint expects it. Prometheus must be started with `--web.enable-remote-write-receiver`.
- **Username**: Prometheus basic auth username to access the remote-write endpoint.
- **Password**: Password accepted by the receiving Prometheus remote-write endpoint.
- **Skip TLS verification**: Skip TLS verification for self-signed certificates. ClusterControl ships self-signed certificates by default, so this is on out of the box. Uncheck if your remote Prometheus has a CA-signed certificate.
- **Cluster label**: Stamps every series as `kubernetes_cluster` to avoid Prometheus aggregate metrics collisions from multiple workload clusters.

### 2.80.10 10. Install Argo CD

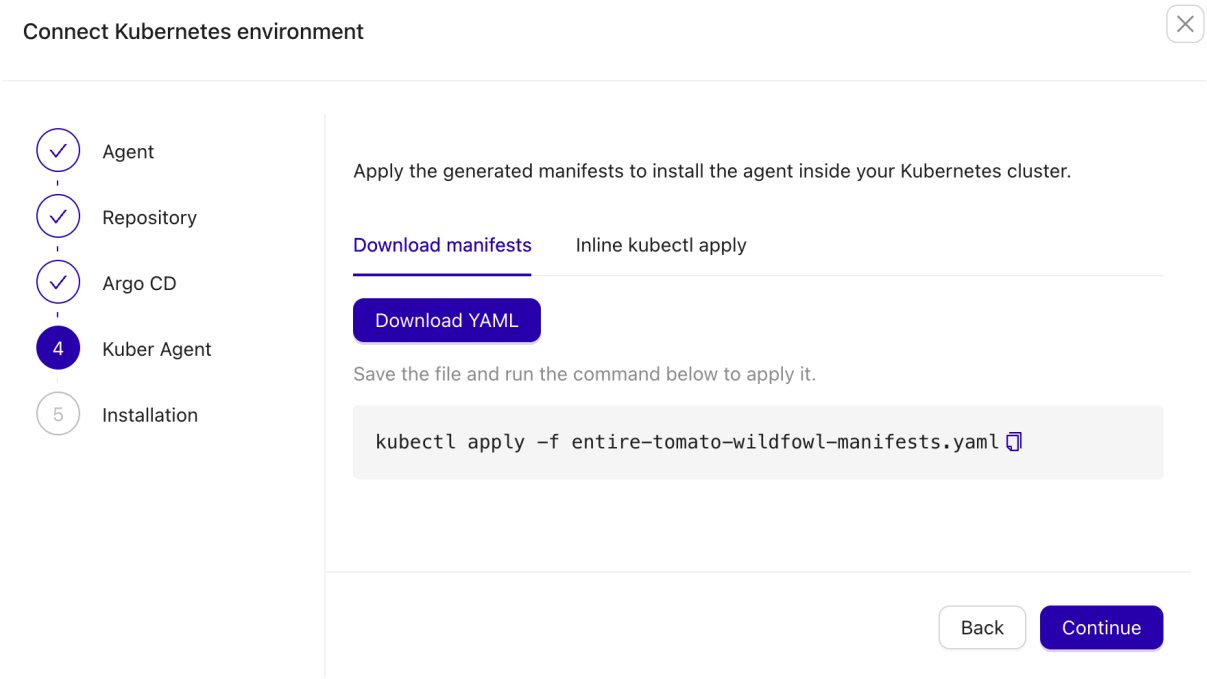
*Skip this step if **Enable GitOps** and **Install Argo** are unchecked in step 5. This step is for GitOps integration.*

The wizard generates a `kubectl` command to install Argo CD. Copy and paste this command into your Kubernetes management terminal. This installs the GitOps controller components needed for synchronization with your GitHub repository.



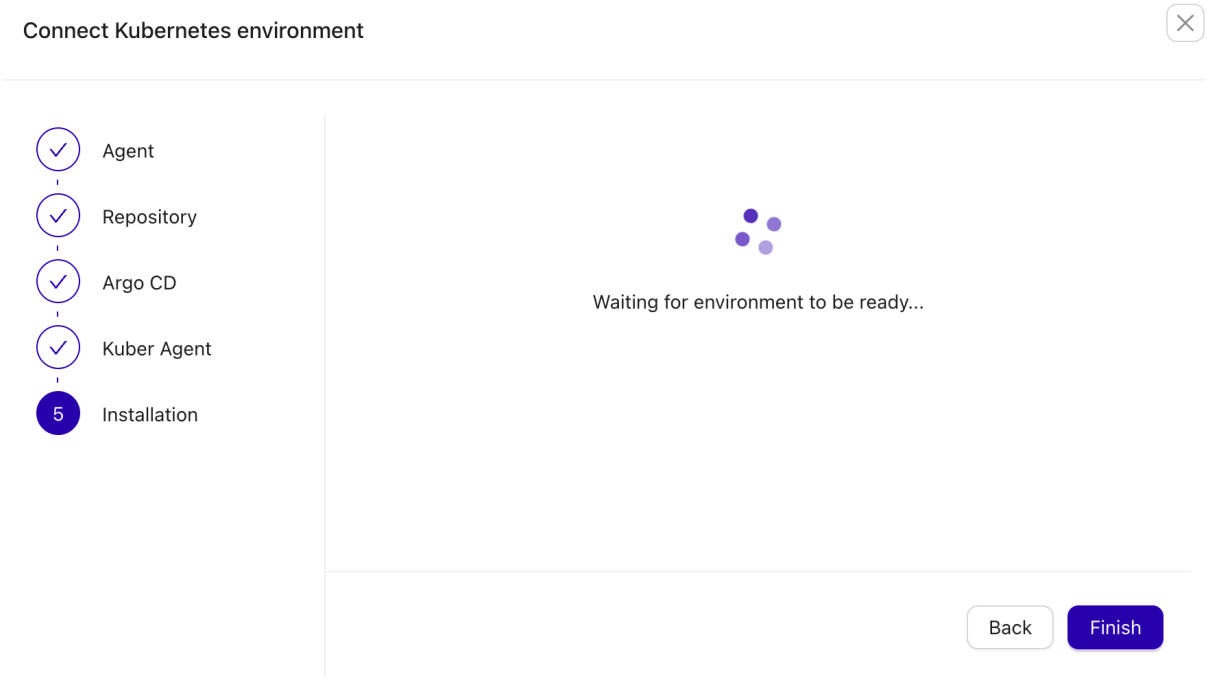

### 2.80.11 11. Install the agent

Next, you can download and execute the YAML file or you can copy and execute the command provided in the inline tab. This exports the necessary environment variables and installs the ClusterControl Kubernetes agent, which connects your cluster back to ClusterControl for management and monitoring.




### 2.80.12 12. Wait for environment detection

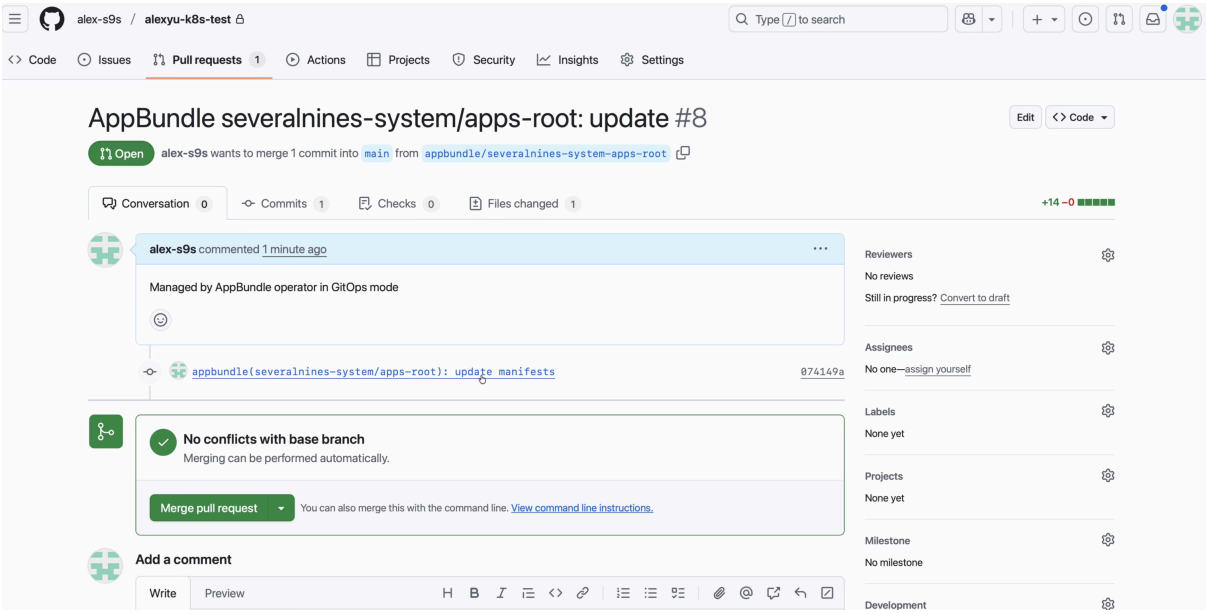
Once installation is complete, return to ClusterControl and wait for the new environment to appear in the dashboard.




### 2.80.13 10. Pull requests

*Skip this step if **Enable GitOps** and **Install Argo** are unchecked in step 5. This step is for GitOps integration.*

ClusterControl automatically creates a GitHub Pull Request for configuration deployment. Click **View PR**, review the YAML and Helm manifests, and merge it into your main branch.




### 2.80.14 11. GitOps synchronization

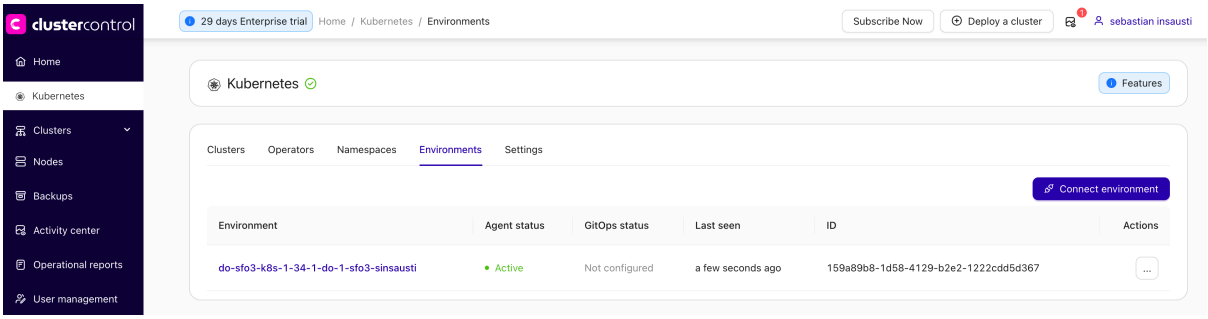
*Skip this step if **Enable GitOps** and **Install Argo** are unchecked in step 5. This step is for GitOps integration.*

Wait for Argo CD to reconcile the merged configuration with the cluster and deploy the agent components.

### 2.80.15 12. Confirm activation

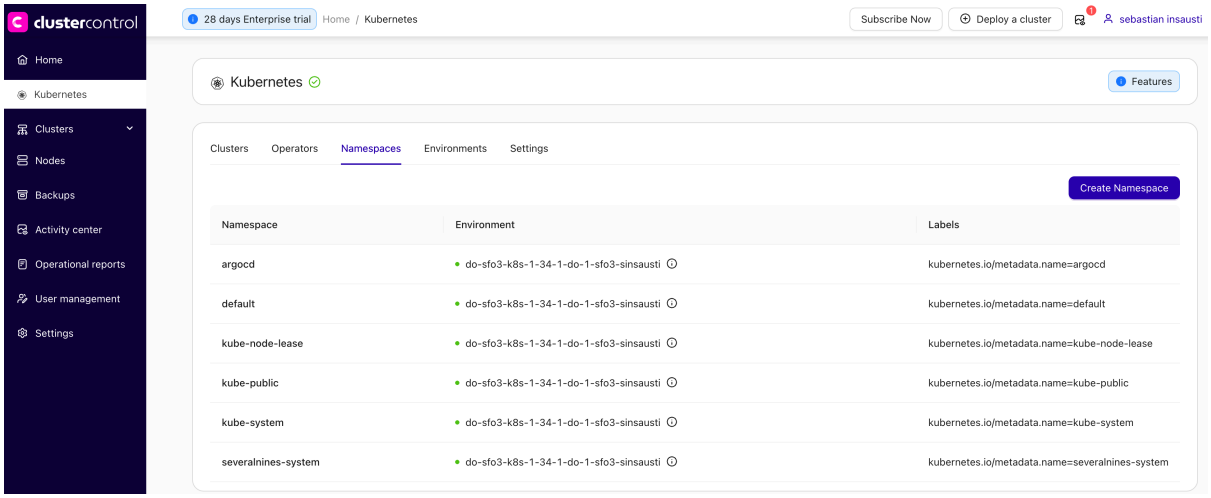
*Skip this step if **Enable GitOps** and **Install Argo** are unchecked in step 5. This step is for GitOps integration.*

Finally, verify that the Kubernetes environment's status changes to "Active". This indicates successful connection and readiness for ClusterControl operations such as monitoring, app deployment, and GitOps-based management.




## 2.81 Create a Kubernetes namespace

Go to ***ClusterControl GUI → Kubernetes → Namespaces*** and click on **Create Namespace**. Create the namespace and wait for the **View PR** to be shown.

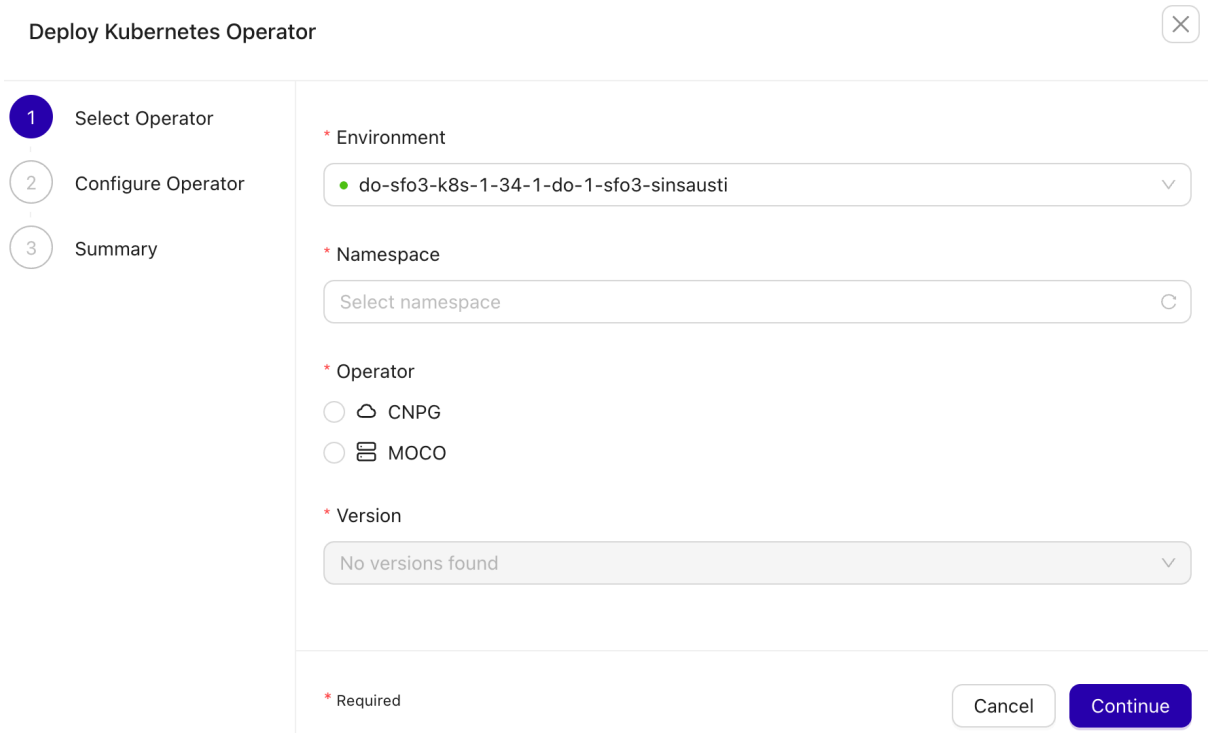



Click on **View PR** to open a page to GitHub to review and merge the Pull Request. A new namespace has been created that can be used to install a database operator.

## 2.82 Deploy a database operator

Before deploying a database cluster, a corresponding database operator for a specific database vendor or technology must be installed in the Kubernetes cluster that you want to use.

Go to ***ClusterControl GUI → Kubernetes → Operators*** and click on **Deploy operator** and select the namespace to use and the database operator to install.




### 2.82.1 Supported operators

The following database operators are currently supported:

- PostgreSQL standalone and streaming replication: Cloudnative-pg

- MySQL standalone and replication: Moco

### 2.82.2 Kubernetes and operator version compatibility

ClusterControl deploys each supported database operator against a pinned version, together with a matching `cert-manager` release where one is required. The matrix below currently covers the Moco operator; other operators are not yet tracked here. Use it to confirm that your Kubernetes cluster is supported before deploying Moco.

| Operator | Database | Pinned operator version | Pinned cert-manager version | Supported Kubernetes versions |
|---|---|---|---|---|
| Moco | MySQL | 0.35.0 | 1.20.1 | 1.33 – 1.35 |

**Warning**

MySQL on Kubernetes 1.34 and later

On Kubernetes 1.34 and later, Moco must be version 0.26.0 or newer. Moco releases below 0.25.0 ship no `ValidatingAdmissionPolicy`; the 0.25.x line is the only one that ships its `ValidatingAdmissionPolicy` and `ValidatingAdmissionPolicyBinding` objects under the `admissionregistration.k8s.io/v1beta1` API, which Kubernetes removed in 1.34, so applying those manifests fails on newer clusters. Moco 0.26.0 is the first release using the GA `admissionregistration.k8s.io/v1` API. ClusterControl pins Moco 0.35.0 to remain compatible across Kubernetes 1.33 to 1.35.

**Note**

Note For Kubernetes 1.35 specifically, Moco 0.35.0 requires Kubernetes v1.35.4 or later, or the `MaxUnavailableStatefulSet` feature gate to be disabled.

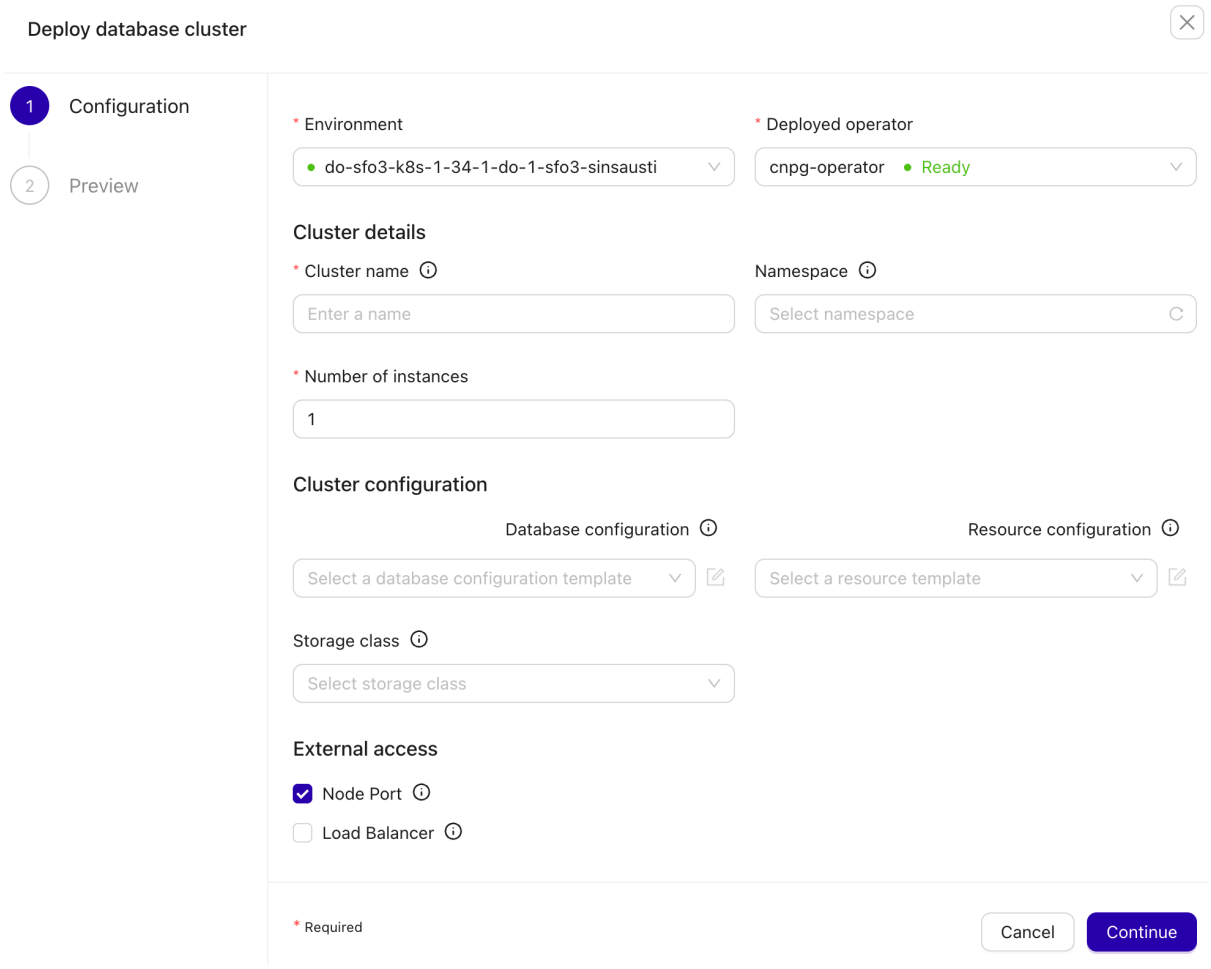
**Note**

Note All third-party Helm charts and container images used by ClusterControl's Kubernetes integration, including database operators, `cert-manager`, and related dependencies, are mirrored to the Severalnines artifact registry rather than pulled directly from upstream. This avoids depending on the availability of public registries and works in network-restricted environments.

## 2.83 Create a new database cluster

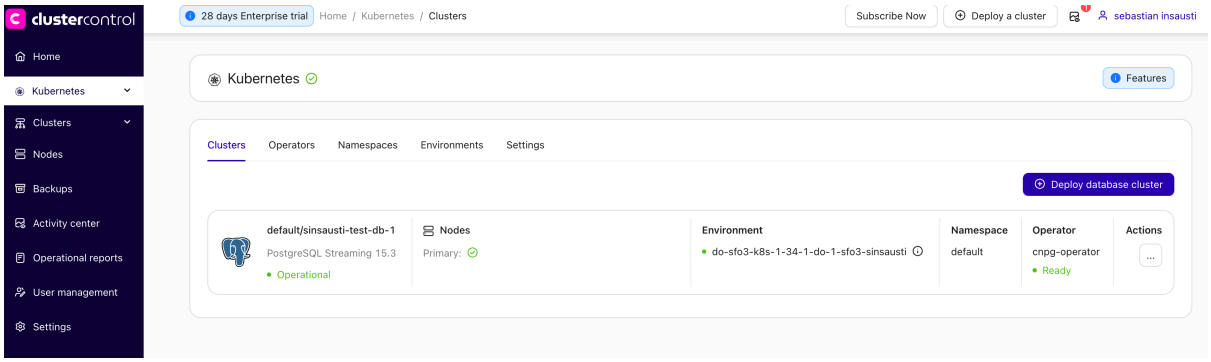
Once the desired database operators have been deployed, the next step is to deploy a new database cluster using one of those operators.

Go to ***ClusterControl GUI → Kubernetes → Clusters*** and click on **Deploy database cluster**.




In this section, complete the required information:

- Select an environment where to deploy it.
- Select which database operator to use.
- Set a cluster name and which namespace to use.
- Number of nodes to deploy.
- The database configuration template to use.
- The resource template to use for CPU, Memory and Storage.
- Define how to access the database cluster, i.e., via a port on the node or launching a load balancer (which can incur cost depending on the Kubernetes cluster provider).

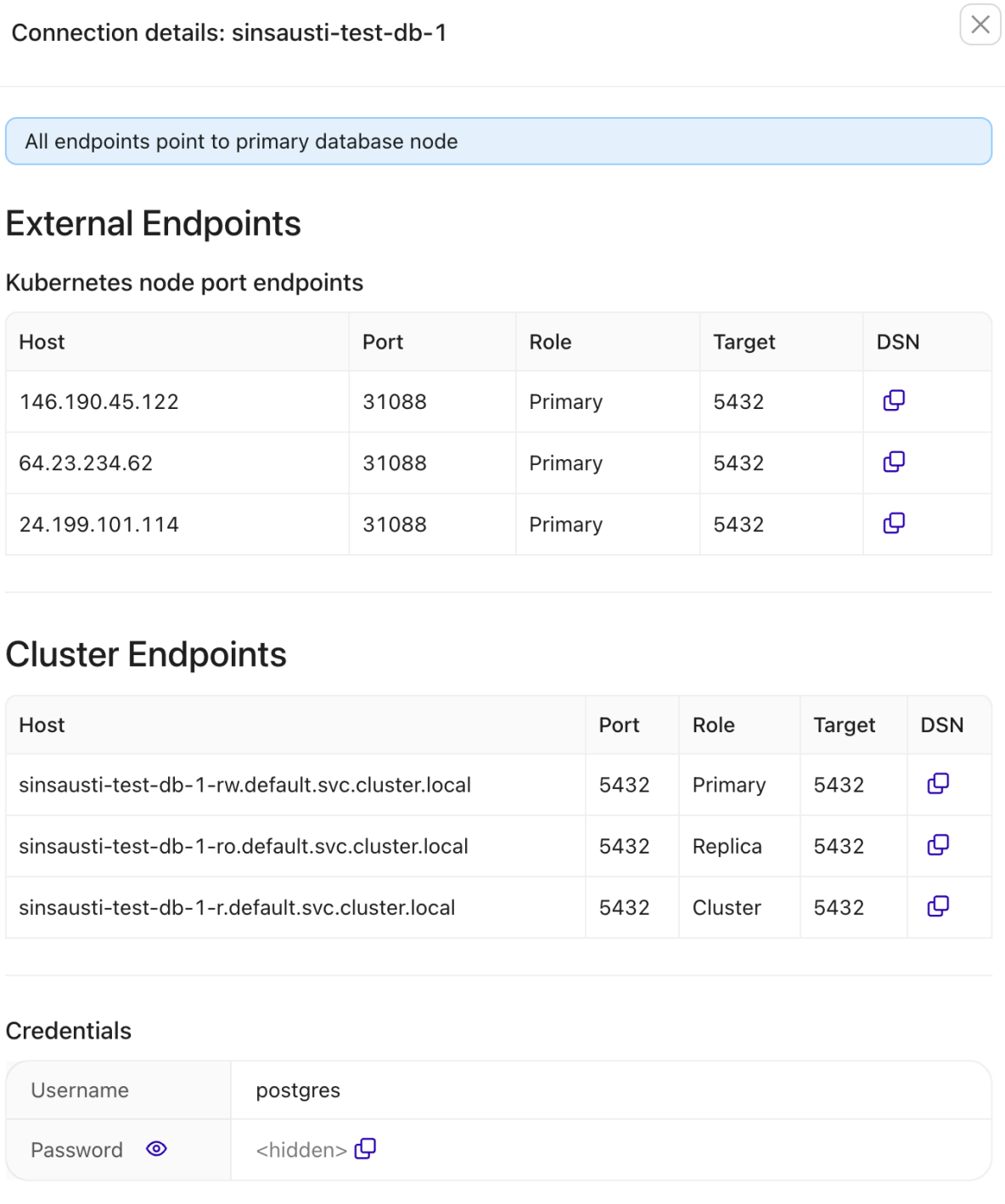


**Note**

Note Database cluster health and phase shown in the UI are driven by watching the underlying operator resources directly, so status changes during deployment, scaling, or recovery are reflected promptly.

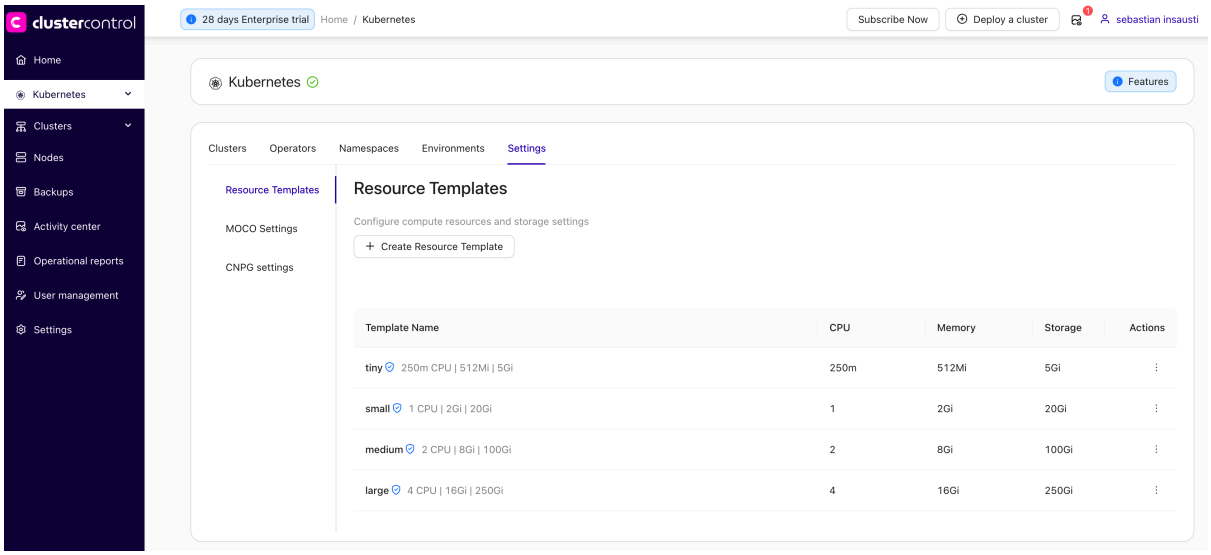
### 2.83.1 Database connection details

To enable your applications to connect to the database, you need the connection details for the database cluster. These details can be retrieved by selecting the cluster and using the **Details** menu action.



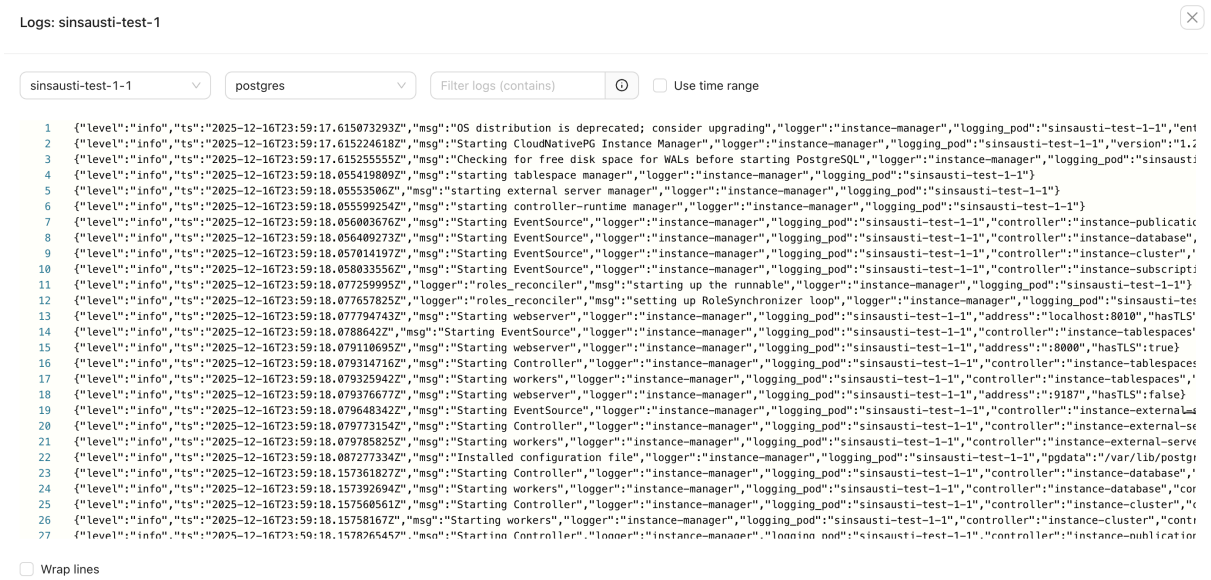
## 2.84 Resource and database configuration templates

ClusterControl supports creating templates that specifies what resources limits to set and different database configurations that can be quickly applied at deployment.




## 2.85 Retrieve and view logs

To access the logs, go to each section (**Clusters**, **Operators**, **Environments**) and click on the **Logs** option under the **Actions** menu.




## 2.86 Purge resources

- Clusters: Go to ***ClusterControl GUI → Kubernetes → Clusters → Cluster Actions → Delete***.
- Operators: Go to ***ClusterControl GUI → Kubernetes → Operators → Operator Actions → Delete***.
- Environments: Go to ***ClusterControl GUI → Kubernetes → Environments → Environment Actions → Delete***.

**Note**

Note Deleted clusters are removed from the **Clusters** list promptly; stale entries no longer linger after a delete.

## 2.87 Observability

ClusterControl's Kubernetes components (`kuber-proxy`, `kuber-agent`, and the agent-operator) emit structured logs. Each request carries a request ID that is propagated across the proxy, agent, and operator, so a single operation, such as a database cluster deployment, can be traced end to end across components.

Both `kuber-proxy` and the `kuber-agent` expose Prometheus metrics. A Prometheus agent is deployed alongside the ClusterControl Kubernetes agent on each connected workload cluster to scrape and forward these metrics. Default dashboards are provided out of the box, covering proxy and agent health and request throughput.

## 2.88 Troubleshooting issues

Use k9s to interact with your Kubernetes cluster. It makes it easier to navigate and visualize the resources deployed.

### 2.88.1 Restart agent

Sometimes you may need to restart agent when issues occurring. Known issues which are fixed by restart are:

- Agent is in a disconnected state and doesn't attempt to connect to proxy.
- Moco operator backups are successful but can't see backups in UI.

To restart, run the following command:

```
kubectl rollout restart deployment/kuber-agent-controller-manager -n␣
↪severalnines-system
```

### 2.88.2 Agent with debug log

To see debug logs when installing or upgrading agent add `--set debug.logLevel=debug` to helm install or upgrade command.

- Watch agent logs:

```
kubectl logs -l app.kubernetes.io/instance=kuber-agent -n␣
↪severalnines-system -f
```

- Search in logs:

```
kubectl logs -l app.kubernetes.io/instance=kuber-agent -n␣
↪severalnines-system --tail=-1 | grep "error"
```

### 2.88.3 Basic troubleshooting commands

- View pods in a specific namespace:

```
kubectl get pods -n <namespace>
```

- Get detailed information about a resource:

```
kubectl describe <resource> <resource-name> -n <namespace>
```

- View pod logs:

```
kubectl logs <pod-name> -n <namespace>
```

- List all pods across namespaces:

```
kubectl get pods --all-namespaces
```

## 2.89 Example issues and solutions

Let's see some examples about different issues and its possible solutions.

### 2.89.1 Failed database operators

- **Moco operator failure:** Check the operator status:

```
kubectl get databaseoperator
NAME         TYPE         STATUS   VERSION   AGE
cnpg         cnpg         Ready              38m
moco         moco         Error              37m
stackgres    stackgres    Ready              35m
```

```
kubectl get pods --all-namespaces
NAMESPACE             NAME                                                  ␣
↪READY   STATUS              RESTARTS      AGE
cert-manager          cert-manager-86d7c7b689-znbz5                         ␣
↪1/1     Running             0             13m
cert-manager          cert-manager-cainjector-77894f5f57-djr94              ␣
↪1/1     Running             0             13m
cert-manager          cert-manager-webhook-6cf469dbd6-tkvnd                 ␣
↪1/1     Running             0             13m
cnpg-system           cnpg-controller-manager-8db87d769-jcvp2               ␣
↪1/1     Running             1 (13m ago)   13m
default               cnpg-cluster-1                                        ␣
↪1/1     Running             0             11m
default               cnpg-cluster-2                                        ␣
↪1/1     Running             0             10m
default               cnpg-cluster-3                                        ␣
↪1/1     Running             0             10m
severalnines-system   kuber-agent-controller-manager-c6b7d9489-259gp        ␣
↪1/1     Running             0             19h
```

(continues on next page)

(continued from previous page)

```
stackgres             stackgres-operator-5bfff484d8-55dmj             ␣
↪0/1     ContainerCreating   0             10m
stackgres             stackgres-operator-set-crd-version-n2rm7        ␣
↪0/1     Completed           0             10m
stackgres             stackgres-restapi-6554545f7b-mmpmc              ␣
↪0/2     ContainerCreating   0             10m
```

- **Diagnosis:** Moco operator requires cert-manager and operator pods. While cert-manager is running, the operator pods have not been deployed.
- **Check agent logs for details:**

```
kubectl logs kuber-agent-controller-manager-c6b7d0000-000gp -n␣
↪severalnines-system -f
```

```
       ```
       $ kubectl logs kuber-agent-controller-manager-c6b7d0000-000gp -n␣
↪severalnines-system -f
       2025-03-19T04:27:24Z    ERROR   Failed to install operator      {
↪"controller": "databaseoperator", "controllerGroup":
↪"agent.severalnines.com", "controllerKind": "DatabaseOperator",
↪"DatabaseOperator": {"name":"moco","namespace":"default"}, "namespace":
↪"default", "name": "moco", "reconcileID":
↪"af00009c-0000-4b9a-0000-faadfbf70000", "error": "failed to install␣
↪cert-manager: failed to apply manifest: [failed to apply resource␣
↪moco-system/cert-manager-edit: clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-edit\" is forbidden: user \
↪"system:serviceaccount:severalnines-system:agent-operator-controller-manager\"␣
↪(groups=[\"system:serviceaccounts\" \
↪"system:serviceaccounts:severalnines-system\" \"system:authenticated\"])␣
↪is attempting to grant RBAC permissions not currently␣
↪held:\n{APIGroups:[\"acme.cert-manager.io\"], Resources:[\"challenges\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"acme.cert-manager.io\"],␣
↪Resources:[\"orders\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"cert-manager.io\"],␣
↪Resources:[\"certificaterequests\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"cert-manager.io\"],␣
↪Resources:[\"certificates\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"cert-manager.io\"],␣
↪Resources:[\"issuers\"], Verbs:[\"deletecollection\"]}, failed to apply␣
↪resource moco-system/cert-manager-controller-approve:cert-manager-io:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-approve:cert-manager-io\" is forbidden: user \
↪"system:serviceaccount:severalnines-system:agent-operator-controller-manager\"␣
↪(groups=[\"system:serviceaccounts\" \
↪"system:serviceaccounts:severalnines-system\" \"system:authenticated\"])␣
↪is attempting to grant RBAC permissions not currently␣
↪held:\n{APIGroups:[\"cert-manager.io\"], Resources:[\"signers\"],␣
↪ResourceNames:[\"clusterissuers.cert-manager.io/*\"],␣
↪Verbs:[\"approve\"]}\n{APIGroups:[\"cert-manager.io\"],␣
```

(continues on next page)
```

(continued from previous page)

```
↪Resources:[\"signers\"], ResourceNames:[\"issuers.cert-manager.io/*\"],␣
↪Verbs:[\"approve\"]}, failed to apply resource␣
↪moco-system/cert-manager-controller-certificatesigningrequests:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-certificatesigningrequests\" is forbidden: user \
↪"system:serviceaccount:severalnines-system:agent-operator-controller-manager\"␣
↪(groups=[\"system:serviceaccounts\" \
↪"system:serviceaccounts:severalnines-system\" \"system:authenticated\"])␣
↪is attempting to grant RBAC permissions not currently␣
↪held:\n{APIGroups:[\"certificates.k8s.io\"], Resources:[\"signers\"],␣
↪ResourceNames:[\"clusterissuers.cert-manager.io/*\"],␣
↪Verbs:[\"sign\"]}\n{APIGroups:[\"certificates.k8s.io\"],␣
↪Resources:[\"signers\"], ResourceNames:[\"issuers.cert-manager.io/*\"],␣
↪Verbs:[\"sign\"]}, failed to apply resource␣
↪moco-system/cert-manager-controller-approve:cert-manager-io:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-approve:cert-manager-io\" not found, failed to␣
↪apply resource␣
↪moco-system/cert-manager-controller-certificatesigningrequests:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-certificatesigningrequests\" not found]"}
       github.com/severalnines/clustercontrol-k8s/agent-operator/internal/controller.(*D
               /
↪app/agent-operator/internal/controller/databaseoperator_controller.go:275
       sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).Reconci
               /
↪go/pkg/mod/sigs.k8s.io/controller-runtime@v0.19.4/pkg/internal/controller/controller.go
       sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).reconci
               /
↪go/pkg/mod/sigs.k8s.io/controller-runtime@v0.19.4/pkg/internal/controller/controller.go
       sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).process
               /
↪go/pkg/mod/sigs.k8s.io/controller-runtime@v0.19.4/pkg/internal/controller/controller.go
       sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).Start.f
               /
↪go/pkg/mod/sigs.k8s.io/controller-runtime@v0.19.4/pkg/internal/controller/controller.go
       2025-03-19T04:27:24Z    ERROR   Reconciler error        {
↪"controller": "databaseoperator", "controllerGroup":
↪"agent.severalnines.com", "controllerKind": "DatabaseOperator",
↪"DatabaseOperator": {"name":"moco","namespace":"default"}, "namespace":
↪"default", "name": "moco", "reconcileID":
↪"af00009c-0000-4b9a-0000-faadfbf70000", "error": "failed to install␣
↪cert-manager: failed to apply manifest: [failed to apply resource␣
↪moco-system/cert-manager-edit: clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-edit\" is forbidden: user \
↪"system:serviceaccount:severalnines-system:agent-operator-controller-manager\"␣
↪(groups=[\"system:serviceaccounts\" \
↪"system:serviceaccounts:severalnines-system\" \"system:authenticated\"])␣
↪is attempting to grant RBAC permissions not currently␣
```

(continues on next page)

(continued from previous page)

```
↪held:\n{APIGroups:[\"acme.cert-manager.io\"], Resources:[\"challenges\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"acme.cert-manager.io\"],␣
↪Resources:[\"orders\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"cert-manager.io\"],␣
↪Resources:[\"certificaterequests\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"cert-manager.io\"],␣
↪Resources:[\"certificates\"],␣
↪Verbs:[\"deletecollection\"]}\n{APIGroups:[\"cert-manager.io\"],␣
↪Resources:[\"issuers\"], Verbs:[\"deletecollection\"]}, failed to apply␣
↪resource moco-system/cert-manager-controller-approve:cert-manager-io:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-approve:cert-manager-io\" is forbidden: user \
↪"system:serviceaccount:severalnines-system:agent-operator-controller-manager\"␣
↪(groups=[\"system:serviceaccounts\" \
↪"system:serviceaccounts:severalnines-system\" \"system:authenticated\"])␣
↪is attempting to grant RBAC permissions not currently␣
↪held:\n{APIGroups:[\"cert-manager.io\"], Resources:[\"signers\"],␣
↪ResourceNames:[\"clusterissuers.cert-manager.io/*\"],␣
↪Verbs:[\"approve\"]}\n{APIGroups:[\"cert-manager.io\"],␣
↪Resources:[\"signers\"], ResourceNames:[\"issuers.cert-manager.io/*\"],␣
↪Verbs:[\"approve\"]}, failed to apply resource␣
↪moco-system/cert-manager-controller-certificatesigningrequests:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-certificatesigningrequests\" is forbidden: user \
↪"system:serviceaccount:severalnines-system:agent-operator-controller-manager\"␣
↪(groups=[\"system:serviceaccounts\" \
↪"system:serviceaccounts:severalnines-system\" \"system:authenticated\"])␣
↪is attempting to grant RBAC permissions not currently␣
↪held:\n{APIGroups:[\"certificates.k8s.io\"], Resources:[\"signers\"],␣
↪ResourceNames:[\"clusterissuers.cert-manager.io/*\"],␣
↪Verbs:[\"sign\"]}\n{APIGroups:[\"certificates.k8s.io\"],␣
↪Resources:[\"signers\"], ResourceNames:[\"issuers.cert-manager.io/*\"],␣
↪Verbs:[\"sign\"]}, failed to apply resource␣
↪moco-system/cert-manager-controller-approve:cert-manager-io:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-approve:cert-manager-io\" not found, failed to␣
↪apply resource␣
↪moco-system/cert-manager-controller-certificatesigningrequests:␣
↪clusterroles.rbac.authorization.k8s.io \
↪"cert-manager-controller-certificatesigningrequests\" not found]"}
       sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).reconci
               /
↪go/pkg/mod/sigs.k8s.io/controller-runtime@v0.19.4/pkg/internal/controller/controller.go
       sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).process
               /
↪go/pkg/mod/sigs.k8s.io/controller-runtime@v0.19.4/pkg/internal/controller/controller.go
       sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller[...]).Start.f
               /
↪go/pkg/mod/sigs.k8s.io/controller-runtime@v0.19.4/pkg/internal/controller/controller.go
```

(continues on next page)

(continued from previous page)

```
       2025-03-19T04:27:24Z    INFO    status-predicate        DIAGNOSTIC:␣
↪Detected status-only change, filtering out reconciliation   {"name":
↪"moco", "kind": "", "oldGeneration": 1, "newGeneration": 1,
↪"resourceVersion": "24302778", "oldResourceVersion": "24302764"}
       2025/03/19 04:27:25 Received message from proxy: {
↪"group":"agent.severalnines.com","version":"v1alpha1","kind":"DatabaseOperator","namesp
↪(list_resources)
       ```
```

### 2.89.2 Failed backups

- **List database backups:**

```
kubectl get databasebackup
NAME                                                                STATUS␣
↪  AGE   STARTED   COMPLETED
cnpg-cluster-backup-cnpg-cluster-backup-backup-20250319055800   Failed␣
↪  26s
```

- **Check CNPG backup status:**

```
kubectl get backup
NAME                                              AGE   CLUSTER       ␣
↪METHOD              PHASE    ERROR
cnpg-cluster-backup-backup-20250319055800   94s   cnpg-cluster  ␣
↪barmanObjectStore   failed   can't execute backup: cmd: [/
↪controller/manager backup cnpg-cluster-backup-backup-20250319055800].
↪..
```

- **Get backup details:**

```
kubectl describe backup cnpg-cluster-backup-backup-20250319055800
```

- **Error:**

```
Status:
Error:  can't execute backup: cmd: [/controller/manager backup␣
↪cnpg-cluster-backup-backup-20250319055800]
error: command terminated with exit code 1
stdErr: {"level":"info","ts":"2025-03-19T05:58:22.131051022Z","msg":
↪"Error while requesting backup","logging_pod":"cnpg-cluster-2",
↪"backupURL":"http://localhost:8010/pg/backup","statusCode":500,
↪"body":"error while requesting backup: while starting backup: cannot␣
↪recover backup credentials: while getting secret s3-credentials:␣
↪secrets \"s3-credentials\" not found\n"}
```
```

### 2.89.3 Node maintenance and troubleshooting

- **Check node status**

```
kubectl get nodes
NAME                    STATUS   ROLES    AGE   VERSION
pool-n97uu1q8u-edf02    Ready    <none>   57d   v1.31.1
```

- **Debug node issues**

  Start a debug container on a node:

```
kubectl debug node/pool-n97uu1q8u-edf02 -it --image=ubuntu:latest -- /
↪bin/bash
```

- **Manage container runtime**

  Access the host filesystem:

```
chroot /host
```

  List container images:

```
crictl images ls
```

  List all Containers (including stopped):

```
crictl ps -a
```

- **Disk usage management**

  Check log usage:

```
du -sh /var/log/*
```

  Check container runtime storage:

```
du -sh /var/lib/containerd/*
```

- **Cleanup tasks**

  Remove stopped containers:

```
crictl ps -a -o json | jq -r '.containers[] |␣
↪select(.state=="CONTAINER_EXITED") | .id' | xargs -r crictl rm
```

  Remove dangling images:

```
crictl images -o json | jq -r '.images[] | select((.repoTags == null)␣
↪or ((.repoTags | length)==0)) | .id' | xargs -r crictl rmi
```

## 2.90 Cloud Providers

ClusterControl supports integration with object storage solutions from multiple providers:

- Amazon Web Services Simple Storage Service (AWS S3)
- Google Cloud Storage
- Azure Blog Storage (via an S3-compatible interface or directly via credentials)
- S3-compatible object storage providers (e.g., MinIO, DigitalOcean Spaces, OpenStack, etc.)

Once configured, you can use cloud object storage as an offsite storage for backups created by ClusterControl.

### 2.90.1 Components

Cloud provider integration is provided by the following ClusterControl modules:

- ClusterControl Cloud (`cmon-cloud`) - A daemonize process running on port 9518 that connects to all supported cloud providers, providing a bridge to `clud` for object storage manipulation. See *ClusterControl Cloud*.
- ClusterControl Cloud File Manager (`clud`) - A client tool used by ClusterControl Controller to manage the cloud objects. See *ClusterControl Cloud File Manager*.

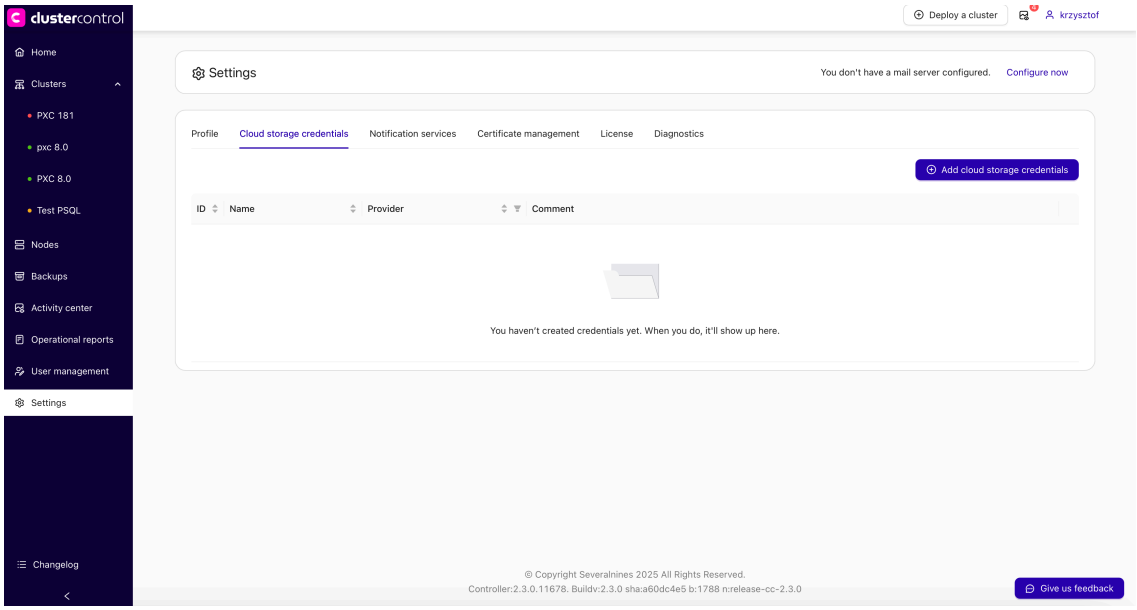
ClusterControl Controller (`cmon`) uses `clud` to create and delete bucket, upload, download and remove files.

### 2.90.2 Prerequisites

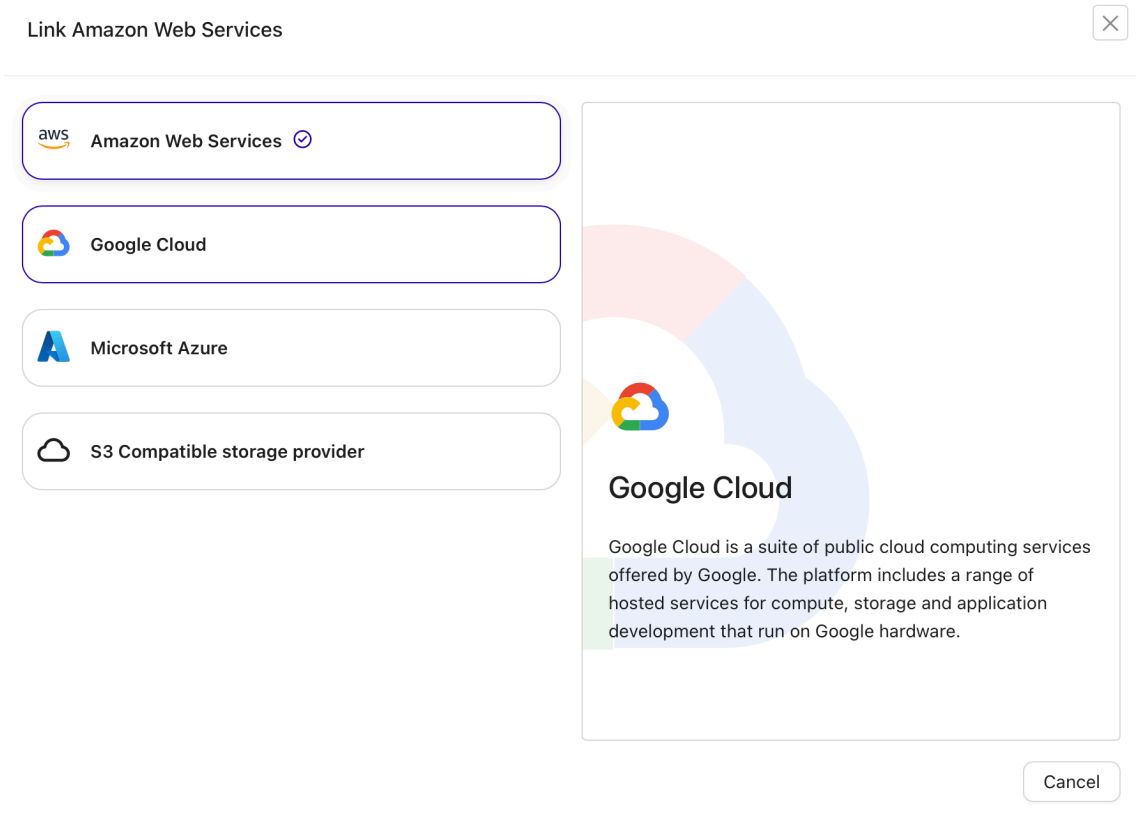
1. You will need valid credentials for the selected cloud storage provider:
   - Amazon Web Services (AWS): Access Key ID and Secret Access Key.
   - Google Cloud Storage (GCS): Service account JSON key file.
   - Microsoft Azure: Service principal credentials or S3-compatible credentials, depending on the deployment setup.
   - Other S3-Compatible Providers: Access key, secret key, region, and custom endpoint information.
2. Depending on the provider, you may need to specify the storage region or service endpoint:
   - AWS: Specify the target region (for example, eu-west-2).
   - Google Cloud Storage: Region configuration is typically handled through the service account credentials.
   - Microsoft Azure: May require a custom endpoint or S3-compatible gateway configuration.
   - Other S3-Compatible Providers: Specify the custom endpoint URL (for example, `https://nyc3.digitaloceanspaces.com`) and region if required.
3. Ensure the cloud storage user account has sufficient privileges to access, list, create, and delete the cloud buckets and objects.

### 2.90.3 Accessing the Cloud Storage Credentials section

1. In ClusterControl, navigate to the ***Settings → Cloud storage credentials*** section. There is an option to **Add cloud storage credentials**.




2. Click **Add cloud storage credentials**.




You will see a list of supported providers:

- Amazon Web Services
- Google Cloud
- Microsoft Azure

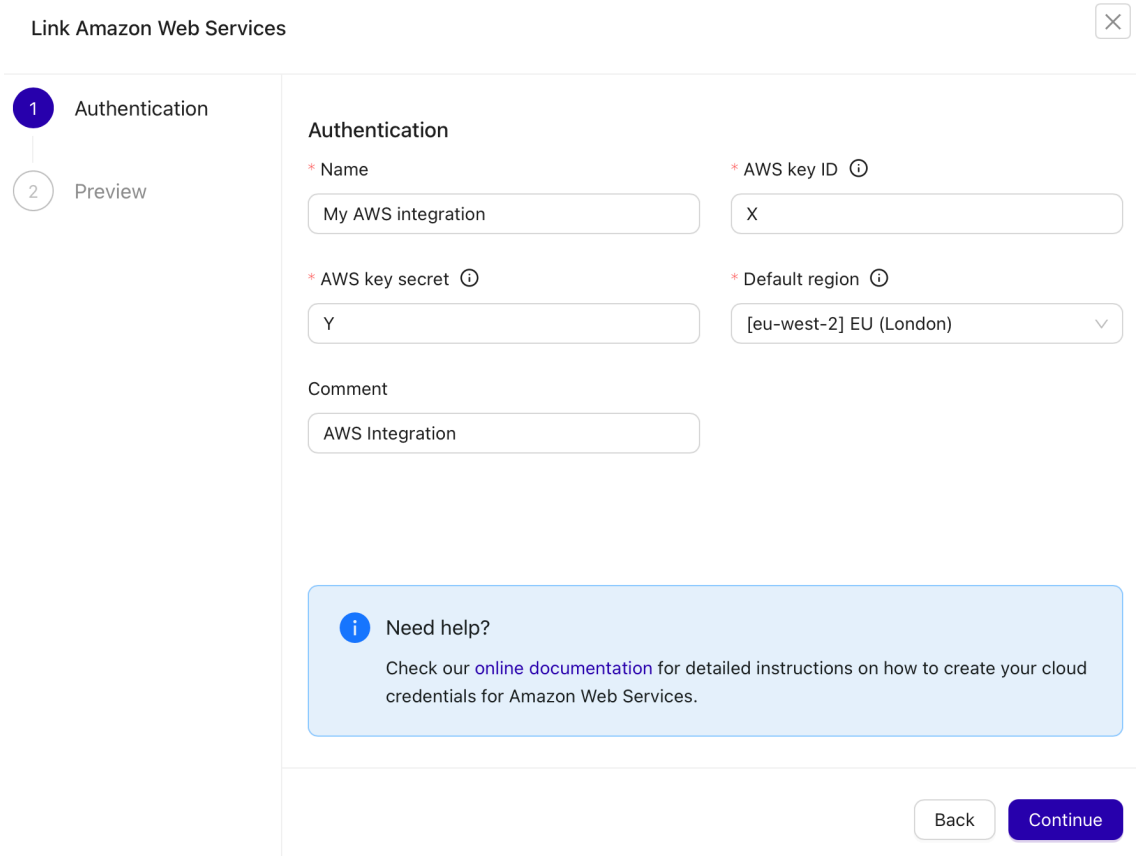
- S3 Compatible storage provider

3. Select the one you want to use.

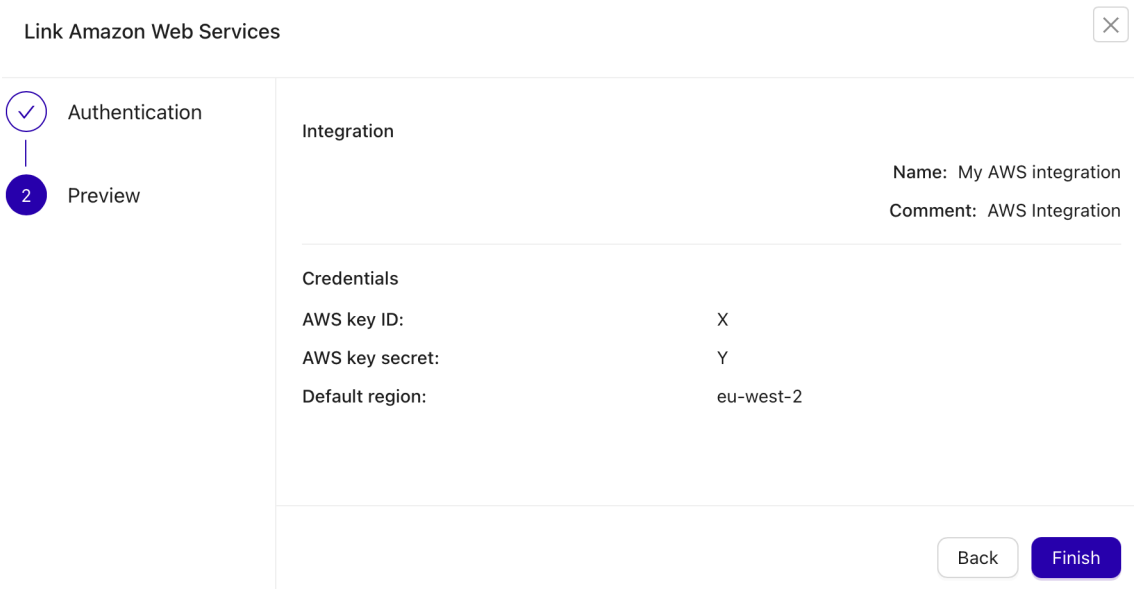
### 2.90.4 Creating a New S3 Integration

#### Setting up AWS S3

1. Select “Amazon Web Services”.

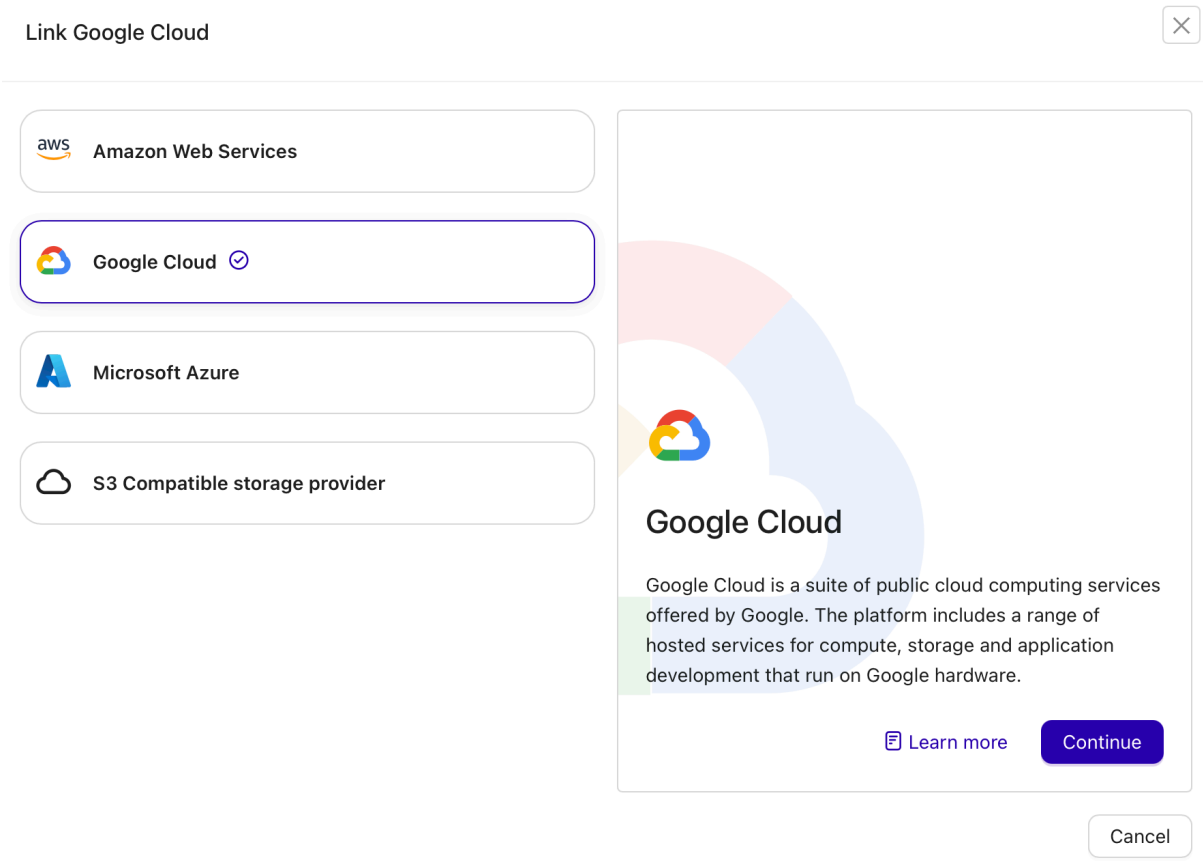



2. Enter Credentials in the Authentication step:
   - Name: A descriptive label, e.g. My AWS Integration.
   - AWS key ID: Your AWS access key ID.
   - AWS key secret: Your AWS secret access key.
   - Default region: Choose your primary region, e.g. [eu-west-2] EU (London).
   - Comment: Optional internal notes (e.g., AWS integration).
3. Click **Continue** to move to the Preview screen.

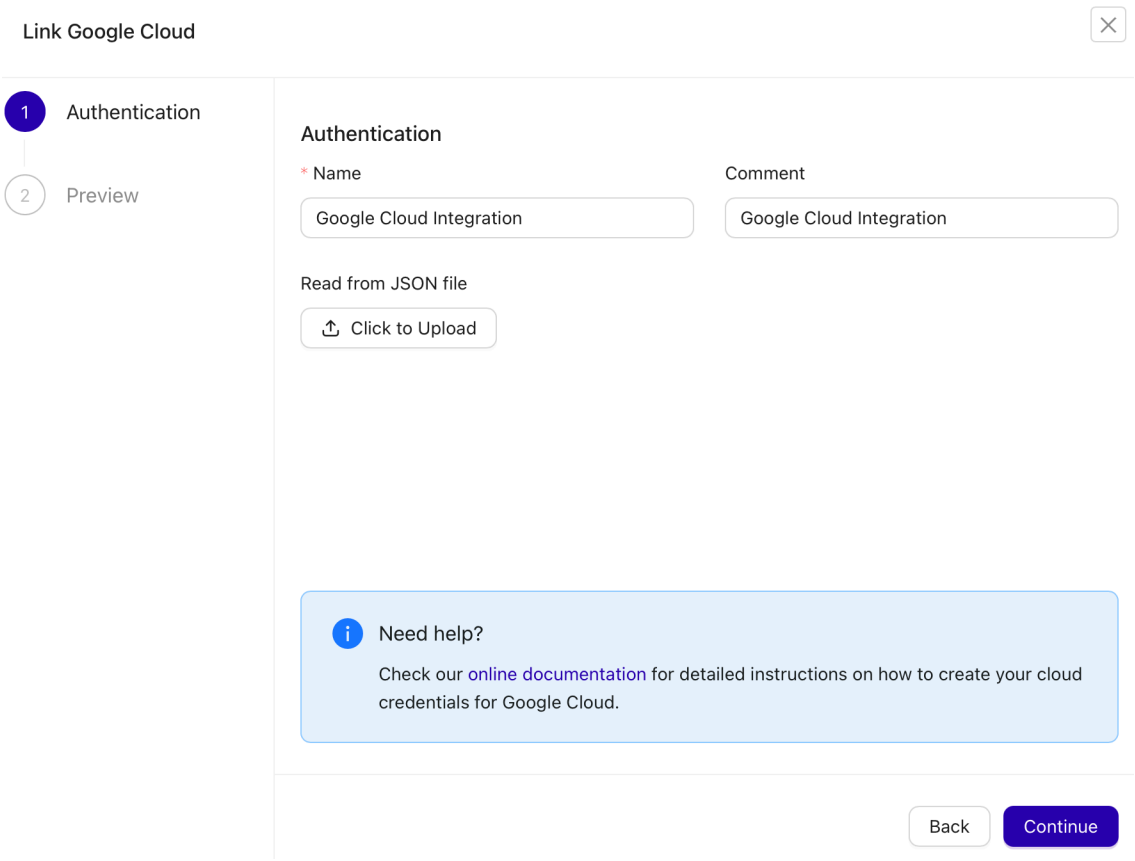



4. Review the details (key ID, region, etc.) and click **Finish** to create the integration.

## Setting up Google Cloud




1. Select Google Cloud.




2. Enter credentials in the authentication step:
   - Name: Enter a descriptive label, e.g. Google Cloud Integration.
   - Comment: Optional internal description.
   - Read from JSON file: Click the **Click to Upload** button to browse for and upload your Google Cloud service account JSON file.

   This JSON file contains the keys and project info needed to authenticate with Google Cloud Storage. For details on creating the JSON file, see Create and delete service account keys.
3. Click **Continue** to view the Preview screen.
4. Verify the integration name and any other details you entered. If correct, Click **Finish** to save the integration.

### Setting up Microsoft Azure

1. Select "Microsoft Azure" from the provider list.
2. Enter credentials in the authentication step:
   - Name: e.g. Azure Integration.
   - Comment: e.g. Azure integration for S3 storage.
   - Read from JSON file: Upload your Azure credential JSON.

   The contents and format of this file depend on your Azure setup. It may include your storage account, tenant ID, client ID, secret, etc.

**Example**

Here is an example of Azure JSON credential file content:

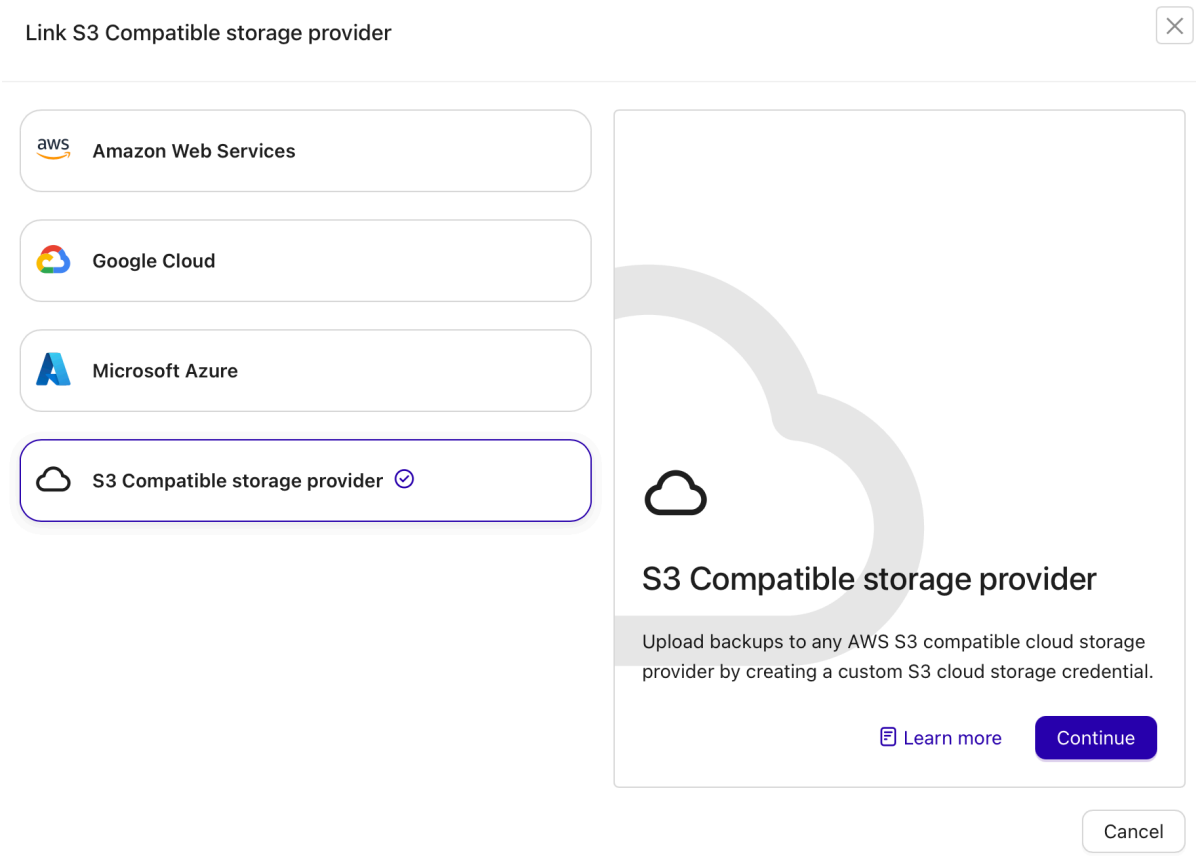
```
{
    "azureCredentials": {
        "applicationId": "d3b07384-d9a1-4b88-8c5f-2a8c4f8d1234",
        "clientSecret": "4444444444444444444444444mnopqrst",
        "subscriptionId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
        "tenantId": "f0e1d2c3-b4a5-6789-0123-456789abcdef",
        "resourceGroup": "rg-myshop-staging",
        "storageAccount": "myshopstorage01",
        "region": "westeurope",
        "environment": "AzureCloud"
    }
}
```

3. Click **Continue**, then Preview the integration details.
4. Verify the integration name and any other details you entered. If correct, Click **Finish** to save the integration.

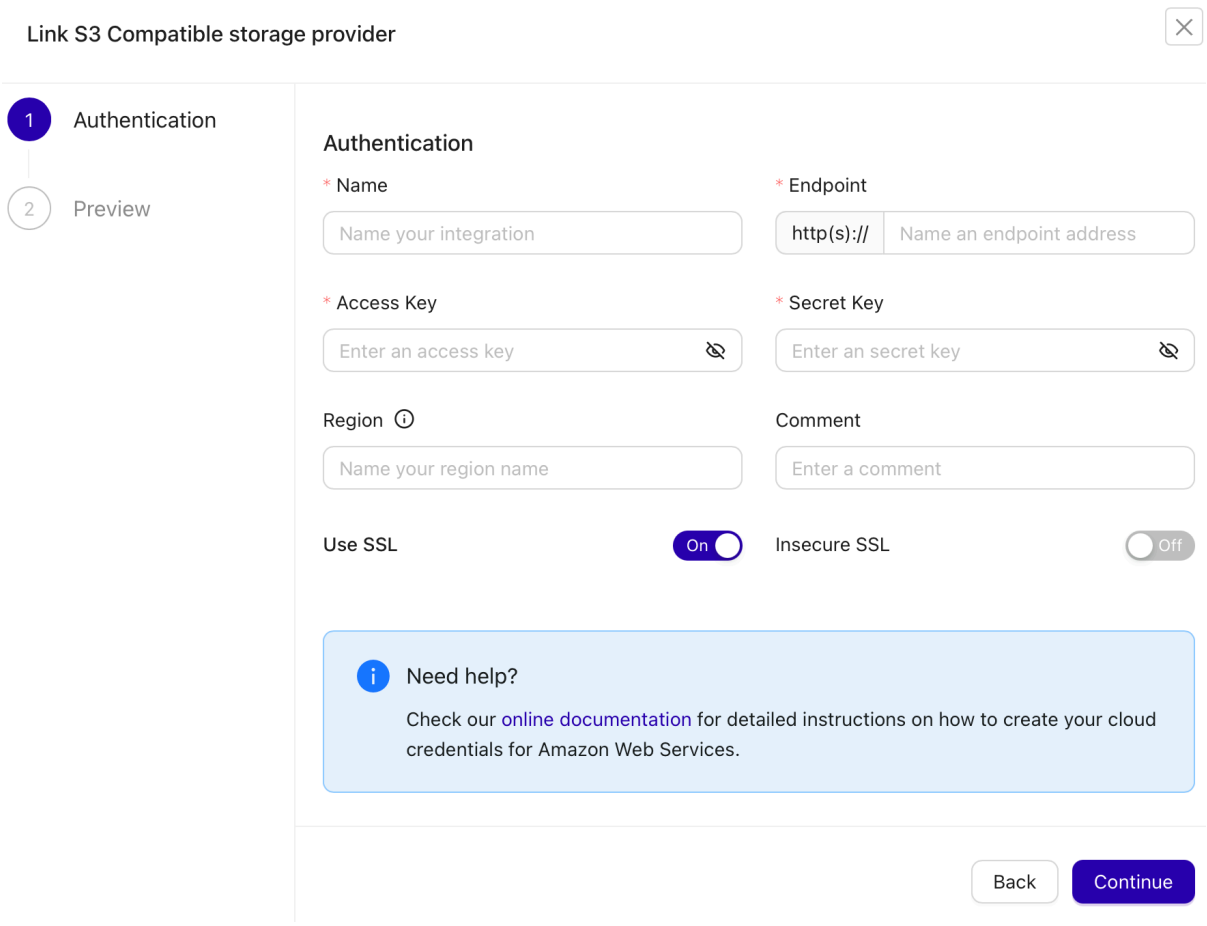
**Tip**

If you are using an S3-compatible gateway for Azure (like Blob storage with S3 proxy), you might instead follow the Other S3 steps and provide the custom endpoint, access key, and secret that the gateway supplies.

### Setting up S3-Compatible Providers




Select “S3 Compatible storage provider” if your preferred provider is not explicitly listed. This covers services like MinIO, DigitalOcean Spaces, OpenStack etc.




1. Select “S3 Compatible storage provider” from the provider list.
2. Enter required data in the Authentication step:
   - Name: e.g. My MinIO Integration.
   - Endpoint: The HTTPS URL of your service (e.g. `https://nyc3.digitaloceanspaces.com`).
   - Access Key: The access key from your S3-compatible account.
   - Secret Key: The secret key from your S3-compatible account.
   - Region: If your provider requires a specific region name (e.g. us-east-1, us-central), enter it here. Not all providers strictly require it.
   - Comment: Optional notes about this integration.
   - Use SSL: On by default. If your provider uses SSL, keep it on. Otherwise, switch to “Off” or “Insecure SSL” if you must connect insecurely (not recommended).
3. Click **Continue**, then Preview the integration details.
4. Verify the integration name and any other details you entered. If correct, Click **Finish** to save the integration.

Starting with ClusterControl 1.8.0, integration with HashiCorp Vault was introduced to securely store credentials used by ClusterControl.

With the release of ClusterControl 2.5.0, this integration has been significantly enhanced. ClusterControl can now store not only credentials, but also complete cluster configuration files and TLS certificates in a

**HashiCorp Vault** or **OpenBao** server, eliminating the need to keep this sensitive information on the local filesystem.

This provides a centralized, authoritative source of configuration data, allowing multiple ClusterControl controllers to share the same cluster configuration. This capability is especially valuable in high-availability deployments, where consistency and synchronization between controllers are critical.

Additionally, multiple independent ClusterControl installations can use the same Vault instance while maintaining isolated configuration paths, ensuring separation between environments while benefiting from centralized secret management.

## 2.91 Requirements

The following requirements must be met to use the Vault integration:

- HashiCorp Vault version 1.x or later, or OpenBao 2.x or later.
- Only the Vault **KV v2** secrets engine is supported.

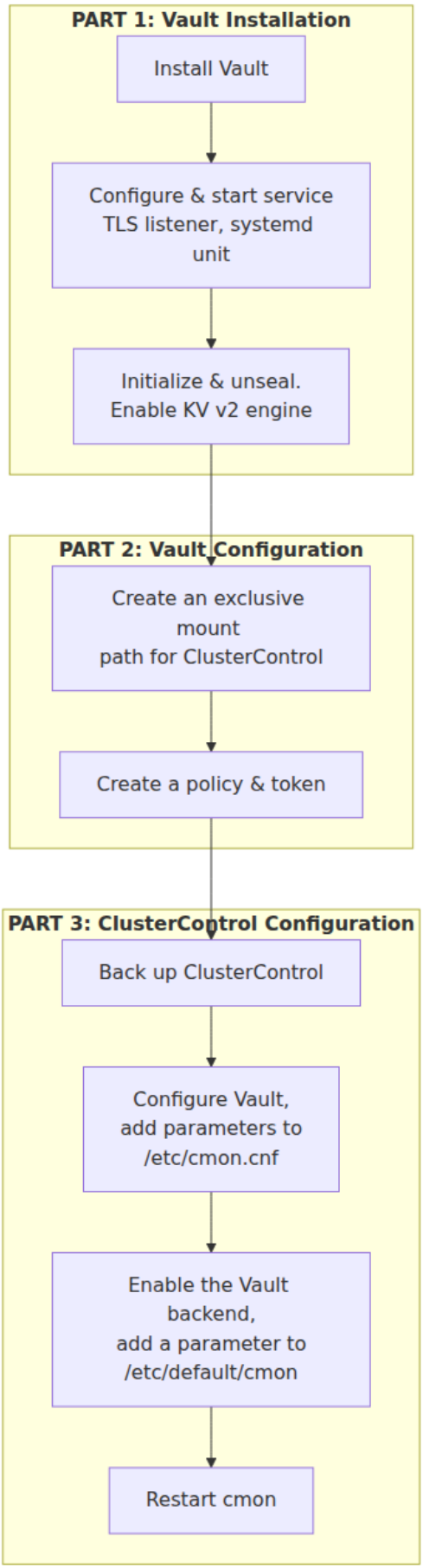
## 2.92 How it works

- On the first ClusterControl startup, all existing configuration files and certificates are automatically moved into HashiCorp Vault/OpenBao. After that, everything is read from and written to Vault instead of the filesystem.
- On every subsequent startup in vault mode, if HashiCorp Vault/OpenBao cannot be reached, ClusterControl stops immediately with a critical error rather than hanging or silently running with no clusters loaded.
- The configuration file `/etc/cmon.cnf` remains on the local filesystem, since it contains the credentials ClusterControl needs to communicate with HashiCorp Vault/OpenBao.

## 2.93 Configuration guide

The procedure to integrate ClusterControl with HashiCorp Vault/OpenBao is split into three parts: install and init the HashiCorp Vault/OpenBao server, configure HashiCorp Vault/OpenBao for ClusterControl and configure ClusterControl.

If you already have a running instance of HashiCorp Vault/OpenBao configured using HTTPS and running KV v2 engine, then skip Part 1 and go straight to Part 2.


PART 1: Vault Installation
Install Vault
Configure & start service
TLS listener, systemd
unit
Initialize & unseal.
Enable KV v2 engine
PART 2: Vault Configuration
Create an exclusive
mount
path for ClusterControl
Create a policy & token
PART 3: ClusterControl Configuration
Back up ClusterControl
Configure Vault,
add parameters to
/etc/cmon.cnf
Enable the Vault
backend,
add a parameter to
/etc/default/cmon
Restart cmon

### 2.93.1 Part 1 - Install and initialize Vault

For the purposes of this document, we'll use an Ubuntu server and HashiCorp Vault as an example.

Install Vault from HashiCorp's repository:

```
sudo apt update
wget -O- https://apt.releases.HashiCorp.com/gpg | sudo gpg --dearmor -o /
↪usr/share/keyrings/HashiCorp-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture)␣
↪signed-by=/usr/share/keyrings/HashiCorp-archive-keyring.gpg]␣
↪https://apt.releases.HashiCorp.com $(lsb_release -cs) main" | sudo tee /
↪etc/apt/sources.list.d/HashiCorp.list
sudo apt update
sudo apt install vault
```

Generate a TLS certificate for Vault (HTTPS is required), then configure the server in `/etc/vault.d/vault.hcl`:

```
    storage "file" {
      path = "/opt/vault/data"
    }

    listener "tcp" {
      address       = "0.0.0.0:8200"
      tls_cert_file = "/opt/vault/tls/tls.crt"
      tls_key_file  = "/opt/vault/tls/tls.key"
    }

    api_addr = "https://vault.test.com:8200"
    ui = true
```

Set the permissions and start the service:

```
sudo chown -R vault:vault /opt/vault/data
sudo systemctl enable vault
sudo systemctl start vault
```

Point the CLI to the server and initialize Vault. This only happens once per Vault deployment.

```
export VAULT_ADDR=https://<VAULT_IP>:8200
export VAULT_CACERT=/opt/vault/tls/tls.crt

vault operator init
```

Initialization returns five **unseal keys** and a **root token**. Any three of the five keys are required to unseal Vault each time it restarts - **store them in a safe place**. The root token is only used for initial setup and should not be used for day-to-day operations. We recommend creating a Vault user and login to the Vault with user/password from the GUI.

**Example output**

```
Unseal Key 1: iKJa/e4sYYUE6hIE42kg5ICl9laAsCWDexWa89iWFolB
Unseal Key 2: 0jeHjTwlQNtnLJFsoUgWxSLWz/OHUtsPMGwzrUr8W+17
Unseal Key 3: xIJqOKJivoaRtTtJxKyeEW/lUjxDwNT002g8d3ezTcH3
Unseal Key 4: Un7dg6bwG536LJgo8QazDtIDzl2ywa8T+mXBz8Ijy/8P
Unseal Key 5: /k0uLeFh8DUEFG4hmECZlsGkqmn7MZIHNoxoPTG/Fnpy

Initial Root Token: hvs.UW1ZiQ6R7bQLPYRBqoCdtZZf

Vault initialized with 5 key shares and a key threshold of 3. Please␣
↪securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 3 of these keys to unseal␣
↪it
before it can start servicing requests.

Vault does not store the generated root key. Without at least 3 keys to
reconstruct the root key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
existing unseal keys shares. See "vault operator rekey" for more␣
↪information.
```

Enable KV v2 secrets engine

```
vault secrets enable -path=clustercontrol -version=2 kv
```

### 2.93.2 Part 2 - Configure Vault for ClusterControl

Create a policy that allows ClusterControl access only to its own path, then issue a token from that policy:

Create a file `/etc/vault.d/cmon-policy.hcl` with the following content

```
    path "clustercontrol/*" {
    capabilities = ["create", "read", "update", "delete", "list"]
    }
```

Then use the file to create a policy

```
vault policy write cmon-policy /etc/vault.d/cmon-policy.hcl
vault token create -policy=cmon-policy
```

Save the token returned in the last step in a safe place which you'll use it to configure ClusterControl.

**Example output**

```
Key                  Value
---                  -----
token                hvs.CAESIHBGyAtdFmDQJxKRvy7dwbNsluJ8MMtdYfq-qowOqH-
↪7Gh4KHGh2cy5zRGJRbHRZdkFCSVAxVjhBSHlSZXk2eW0
```

```
token_accessor       mF0TmqqvaO1aTY87Wbmp2IIH
token_duration       768h
token_renewable      true
token_policies       ["cmon-policy" "default"]
identity_policies    []
policies             ["cmon-policy" "default"]
```

### 2.93.3 Part 3 - Configure ClusterControl

!!! danger “Important”

```
Once files are migrated to Vault, the local copies are removed. It is␣
↪mandatory to back up  your ClusterControl installation before continuing␣
↪with the integration procedure.
```

```
sudo tar -czf cmon-config-backup.tar.gz /etc/cmon* /etc/s9s.conf
sudo mysqldump -u root -p cmon > cmon-db-backup.sql
```

The following parameters are used to enable the use of Vault in ClusterControl.

| Parameter | File | Description | Default |
|---|---|---|---|
| vaultkv_t | /etc/cmon.cnf | Token used to authenticate with HashiCorp Vault/OpenBao | **Required** |
| vaultkv_a | /etc/cmon.cnf | Vault server address (HTTPS only) https://IP:8200 or https://servername:8200 | https://127.0.0.1:8200 |
| vaultkv_m | /etc/cmon.cnf | KV v2 mount path corresponding to the {mount} portion of the Vault API URL | cluster-control |
| vaultkv_p | /etc/cmon.cnf | Mandatory when multiple cmon instances will share the same mount path to manage different sets of clusters (not HA) | *(empty)* |
| vaultkv_n | /etc/cmon.cnf | Optional namespace. When configured, cmon sends it as part of the HTTPS header on each request. This option is independent of vaultkv_path_prefix. Both may be used simultaneously. During startup, cmon verifies that the namespace exists. If not, startup fails rather than silently managing zero clusters. (This feature is only available in OpenBao 2.3+ or HashiCorp Vault Enterprise) | Root namespace |
| EXTRA_OPTS= | /etc/defacmon | Enables the Vault backend. Available options are {file | k8s |

For this example, we use a single shared mount point named `clustercontrol`. Add the Vault connection details to `/etc/cmon.cnf` :

```
vaultkv_token=<CMON_POLICY_TOKEN>
vaultkv_addr=https://<VAULT_SERVER_IP>:8200
vaultkv_mount=clustercontrol
```

where `<CMON_POLICY_TOKEN>` is the token created in Part 2.

We share the mountpoint for two ClusterControl servers with independent managed clusters, so we'll set up `vaultkv_path_prefix` differently: `CC1` for ClusterControl server 1 and `CC2` for ClusterControl server 2.

Parameter for server 1:

```
vaultkv_path_prefix=CC1
```

and for server 2:

```
vaultkv_path_prefix=CC2
```

Enable the Vault backend in `/etc/default/cmon` by adding:

```
EXTRA_OPTS='--secret-storage-engine=vault'
```

**Before restarting ClusterControl**

- HashiCorp Vault/OpenBao must already be initialized and unsealed.
- The KV v2 engine must exist at the configured mount.
- A full backup should be taken.

Confirm Vault is reachable and unsealed before starting ClusterControl:

```
curl --cacert /opt/vault/tls/tls.crt https://<VAULT_IP>:8200/v1/sys/health
```

Look for `"initialized": true` and `"sealed": false` in the response.

Restart ClusterControl:

```
sudo systemctl restart cmon
```

ClusterControl automatically finds every existing cluster config and certificate on disk and migrates them to Vault. This process might take a few minutes depending on the number of existing clusters. If you are logged in to the ClusterControl GUI, you will see the clusters appear gradually. Once the process completes, ClusterControl reads and writes through HashiCorp Vault/OpenBao only.

## 2.94 Path to secrets in Vault

### 2.94.1 Secret paths

Each managed file is stored as a single KV v2 secret.

The path is structured as:

```
/<vaultkv_mount>/severalnines.cmon/<file_path>
```

or:

```
/<vaultkv_mount>/<vaultkv_path_prefix>/severalnines.cmon/<file_path>
```

If `vaultkv_path_prefix` is set, where `<file_path>` indicates the original file path.

Example:

```
/etc/cmon.d/cmon_1.cnf
```

becomes:

```
/clustercontrol/severalnines.cmon/etc/cmon.d/cmon_1.cnf
```

or if *vaultkv_path_prefix=CC1* becomes:

```
/clustercontrol/CC1/severalnines.cmon/etc/cmon.d/cmon_1.cnf
```

You may list the Vault content using vault list command:

```
vault kv list <mount>/<path>/severalnines.cmon/etc/cmon.d
```

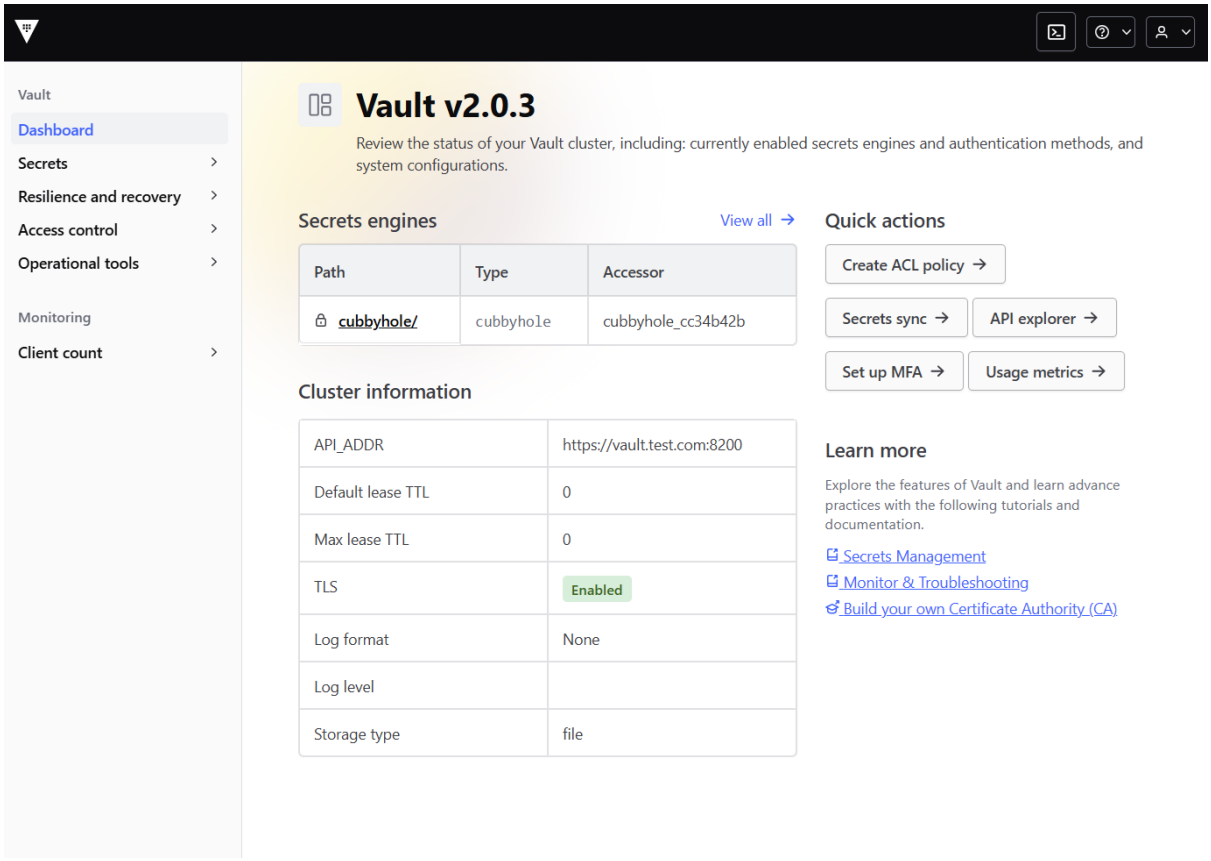
**Example output**

```
$ vault kv list clustercontrol/CC1/severalnines.cmon/etc/cmon.d
Keys
----
cmon_10.cnf
cmon_14.cnf
cmon_15.cnf
cmon_17.cnf
cmon_5.cnf
```

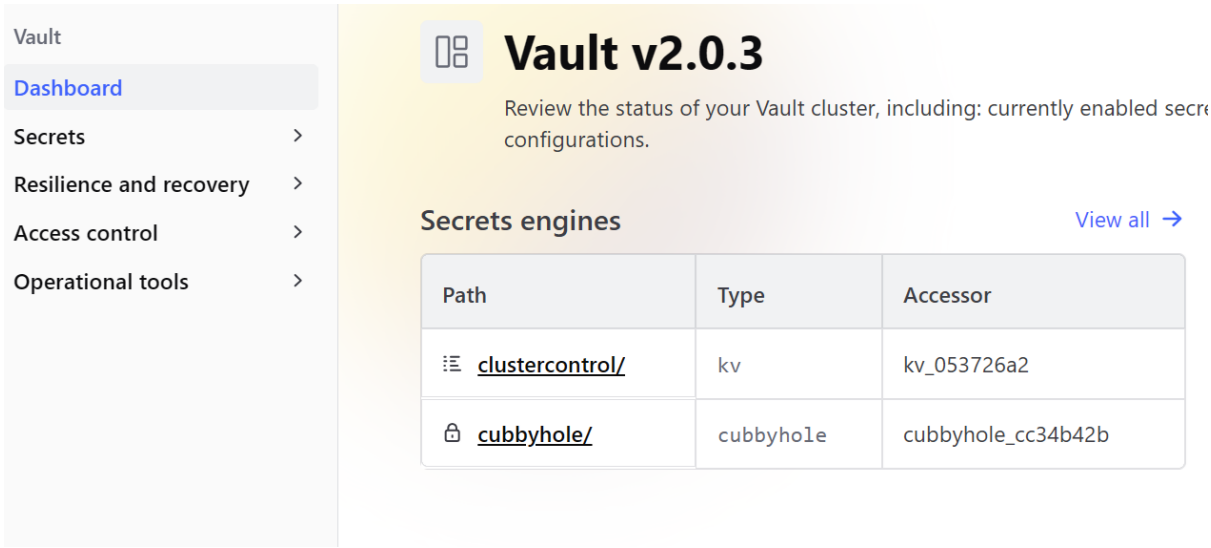
File contents are base64-encoded before being stored, allowing binary files such as certificates and private keys to be safely embedded in JSON.

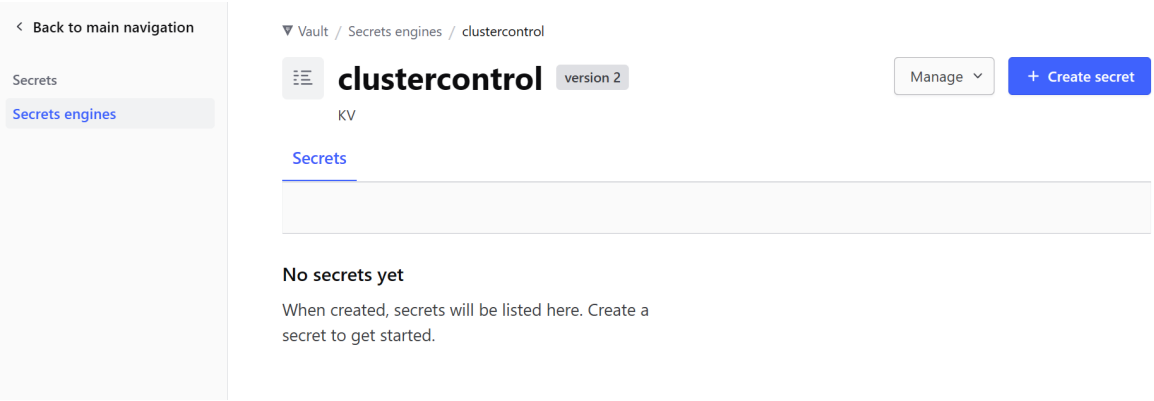
### 2.94.2 Vault GUI

Connect to the Vault GUI (`https://<host_address>:8200/ui`) to check the contents of the Vault.




Before starting ClusterControl, you'll see only the empty mount point:





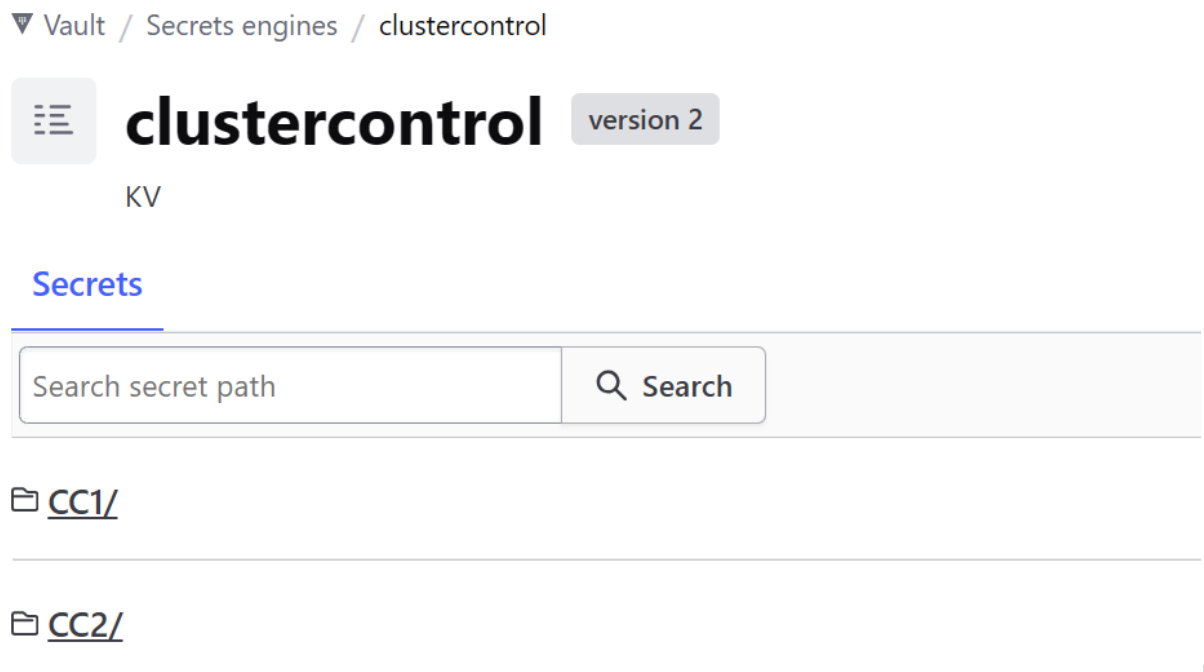

After the ClusterControl server is restarted and linked to Vault, you'll see the content organized in the following tree:

```
    mountpoint_id/path_prefix/severalnines.cmon/etc/cmon.d/CLUSTERID.cnf
    mountpoint_id/path_prefix/severalnines.cmon/var/lib/cmon/ca/RPC_CRT_FILE
    mountpoint_id/path_prefix/severalnines.cmon/var/lib/cmon/ca/RPC_KEY_FILE
    mountpoint_id/path_prefix/severalnines.cmon/var/lib/cmon/cloud_credentials.json
    mountpoint_id/path_prefix/severalnines.cmon/var/lib/cmon/cmon.data
```

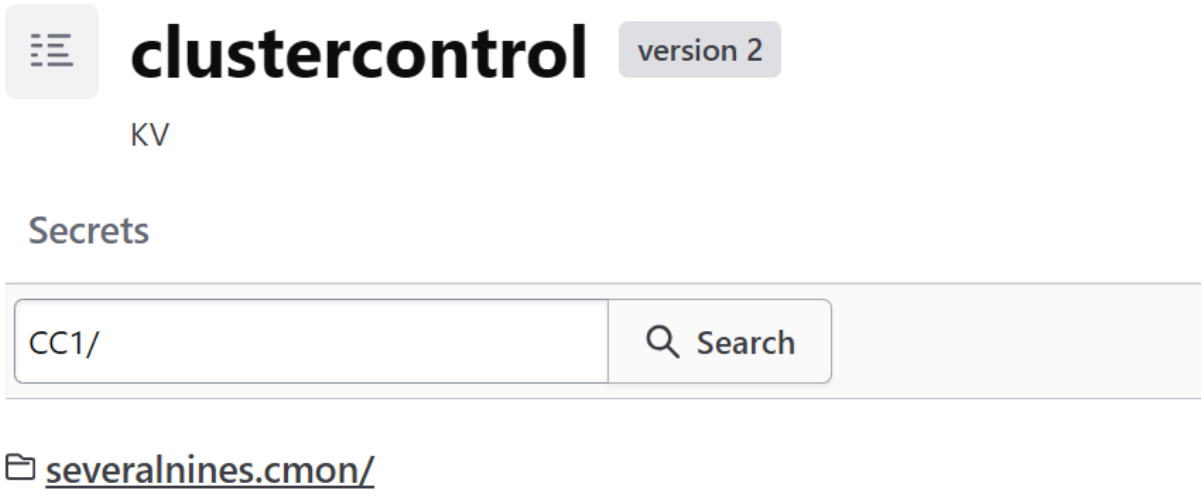
The following example shows the result of migrating 2 ClusterControl servers to the same Vault, one with `path_prefix` set to `CC1` and the other to `CC2`:

```
    clustercontrol/CC1/severalnines.cmon/etc/cmon.d/CLUSTERID.cnf
    clustercontrol/CC1/severalnines.cmon/var/lib/cmon/ca/RPC_CRT_FILE
    clustercontrol/CC1/severalnines.cmon/var/lib/cmon/ca/RPC_KEY_FILE
    clustercontrol/CC1/severalnines.cmon/var/lib/cmon/cloud_credentials.json
    clustercontrol/CC1/severalnines.cmon/var/lib/cmon/cmon.data
    ...
    clustercontrol/CC2/severalnines.cmon/etc/cmon.d/CLUSTERID.cnf
    clustercontrol/CC2/severalnines.cmon/var/lib/cmon/ca/RPC_CRT_FILE
```

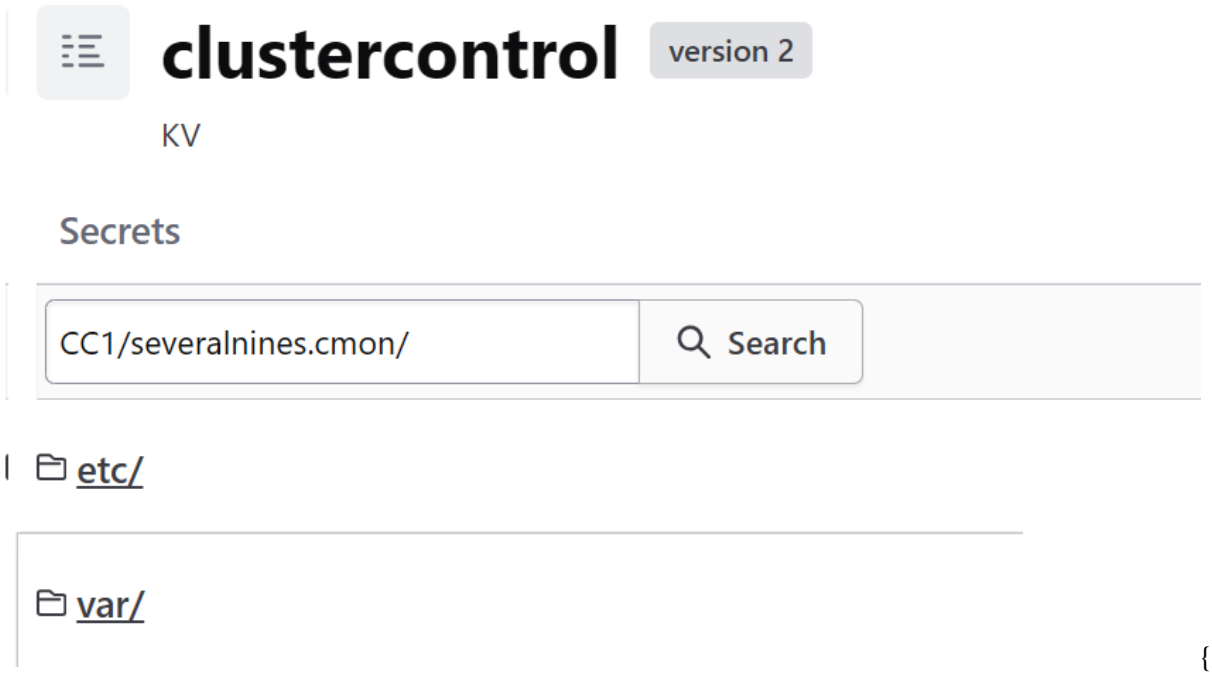
Examining the same in the Vault GUI:




{
width="600" loading=lazy}

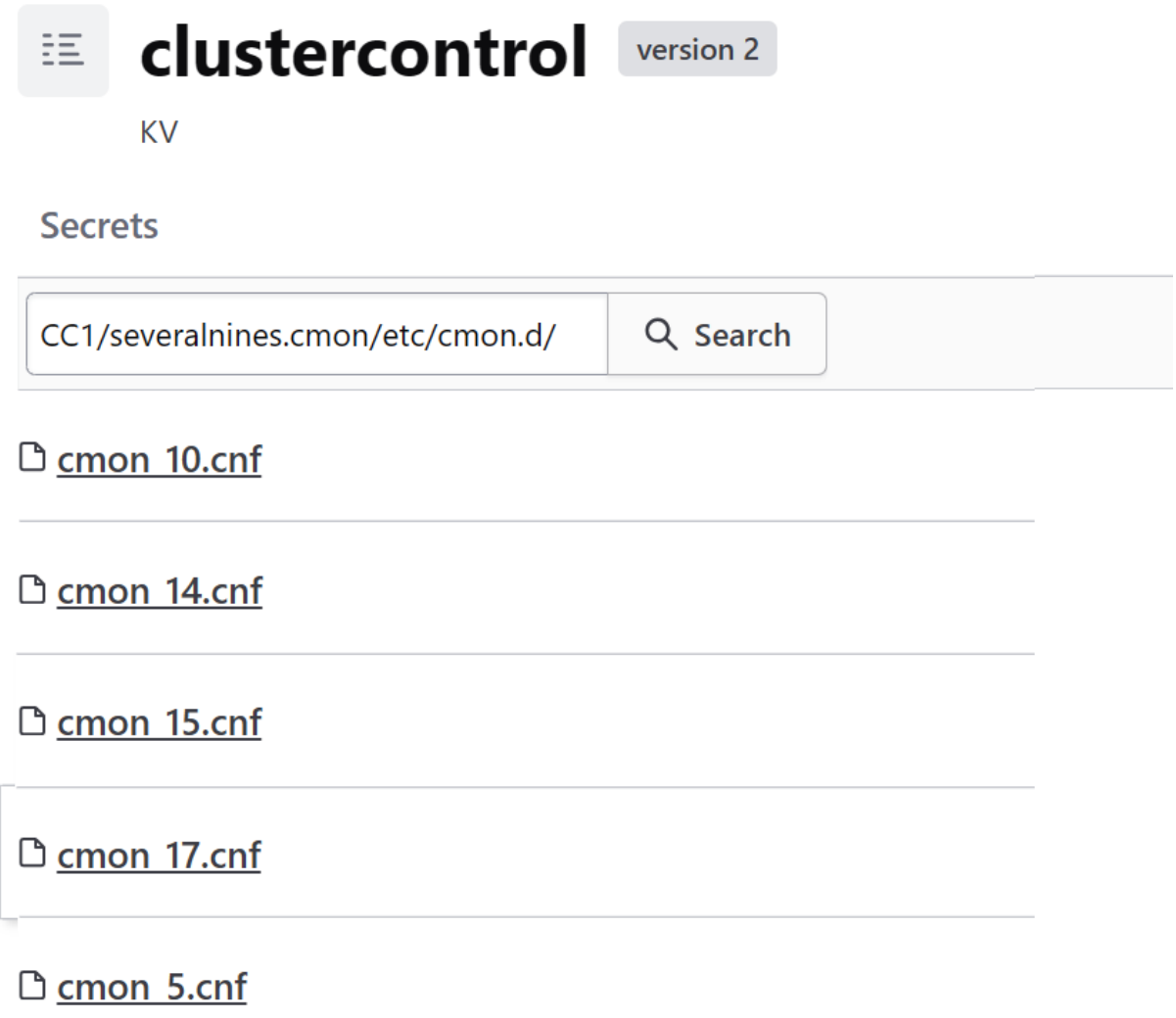



{
width="550" loading=lazy}

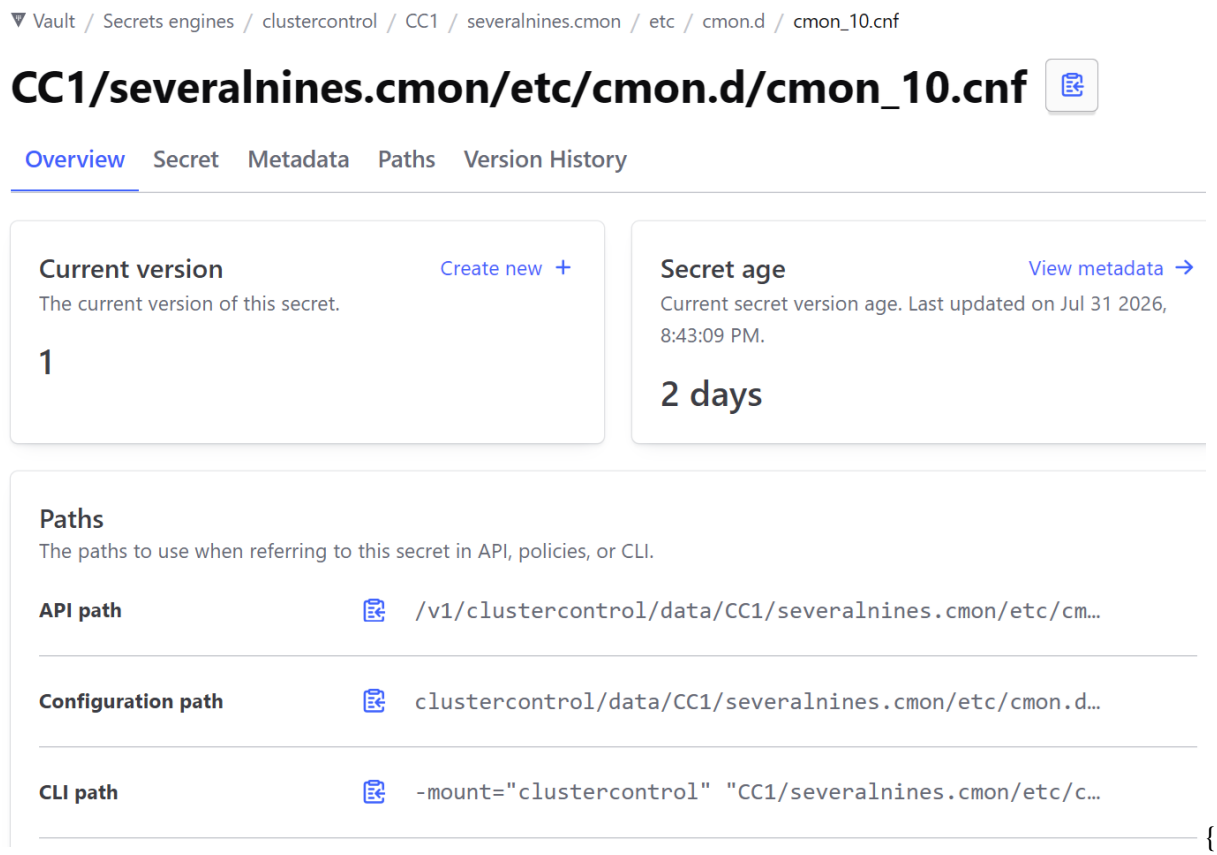



{ width=”550” loading=lazy}

List of ClusterControl instance CC1’s cluster configuration files:

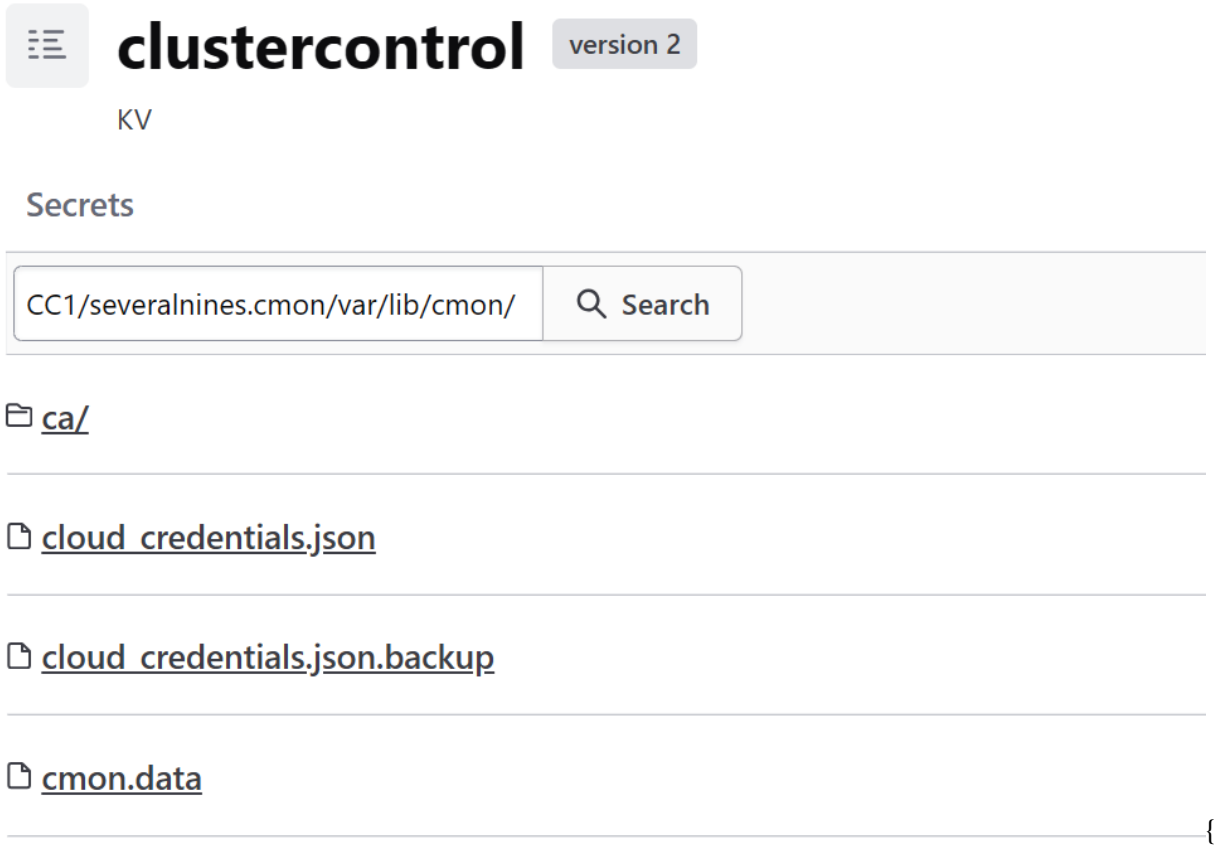



{
width="550" loading=lazy}




{ width=”700” loading=lazy}

Cloud credentials and CA:




{

width=”550” loading=lazy}

## 2.95 Reverting Vault integration

Should you want to revert this process, you should follow these steps:

1. Get all the keys and values from the Vault for all of the clusters.
2. Stop cmon process.
3. Remove ALL Vault parameters from `/etc/cmon.cnf`.
4. Add the keys and values you extracted from the Vault into respective configuration files in `/etc/cmon.d`. You can use the backup copy of the configuration from before Vault has been used but please keep in mind that some additional passwords might have been added in the meantime. Double-check that your old configuration files contain all the data that is stored in the Vault.

## 2.96 Legacy Vault integration

A separate, older Vault integration exists for storing individual cluster credentials (passwords and keys) as key-value pairs. The two systems are independent.

It uses different configuration parameters in `/etc/cmon.cnf` config file : `vault_addr`, `vault_token`, `vault_path`, `vault_auto_migrate`. The new storage secrets engine parameter `vaultkv_*` are intentionally different, ensuring each system operates safely on its own, without affecting the other.

The legacy system is enabled when `vault_token` is present in `/etc/cmon.cnf` file. It adopts a cluster only when secrets already exist in its configured path unless: `vault_auto_migrate=true` is set, in which case secret fields are extracted during the next cluster save.

When new configuration `--secret-storage-engine=vault` is enabled in `/etc/default/cmon`, the legacy credential store is automatically disabled to avoid conflicts.

If the `--secret-storage-engine=vault` option is later removed while `vault_token` remains in `/etc/cmon.cnf`, the legacy integration becomes active again and moves passwords and keys into Vault during the next cluster save.

## 2.97 Limitations

- ClusterControl only supports the KV v2 secrets engine.
- Updating a credential in Vault will not change the underlying credential on the database node. Thus, when changing a password, the account must be updated on the database node(s).
- The cmon-events and cmon-ssh services have yet been migrated to use Vault. Hence the passwords are still present in `/etc/cmon.cnf`.

The Terraform Provider for ClusterControl offers a seamless integration between the powerful infrastructure management tool, Terraform, and the robust database management capabilities of ClusterControl. This provider empowers users to effortlessly provision, configure, and manage database clusters through Infrastructure as Code (IaC), streamlining the deployment process and enhancing the efficiency of managing complex database environments.

With a range of customizable configurations and automation capabilities, this provider simplifies the orchestration of database infrastructure, enabling teams to focus on innovation and development rather than

mundane operational tasks. Leveraging the flexibility and scalability of Terraform alongside the comprehensive management features of Severalnines ClusterControl, organizations can achieve greater agility, reliability, and scalability in their database operations.

This project is hosted in Github, terraform-provider-clustercontrol.

## 2.98 Requirements

| Name | Version |
|---|---|
| Terraform | >= 0.13.x |
| ClusterControl | >= 1.9.8 |

## 2.99 Providers

| Name | Version |
|---|---|
| Terraform ClusterControl Provider | >= 0.1 |

## 2.100 Resources

| Name |
|---|
| clustercontrol_db_cluster |
| clustercontrol_db_cluster_backup |
| clustercontrol_db_cluster_backup_schedule |
| clustercontrol_db_cluster_maintenance |

## 2.101 Quick start

### 2.101.1 Installing and configuring ClusterControl for API access

To get started, you may follow the following steps (if you already have ClusterControl running in your env, skip to #4):

1. *Getting started with ClusterControl*
2. *ClusterControl requirements*
3. *Install ClusterControl*
4. Configure ClusterControl – Enable ClusterControl for API access by editing `/etc/default/cmon` with a text editor and set the `RPC_BIND_ADDRESSES` as shown below:

```
RPC_BIND_ADDRESSES="10.0.0.5,127.0.0.1"
```

Where `10.0.0.5` is the private IP of the ClusterControl host. Restart the ClusterControl service to apply the changes:

```
sudo systemctl restart cmon
```

5. Run a quick test to make sure you can access ClusterControl via its REST API (using curl or Postman):

```
curl -k 'https://10.0.0.5:9501/v2/clusters' -XPOST -d '{"operation":
↪"getAllClusterInfo", "authenticate": {"username":
↪"CHANGE-ME","password": "CHANGE-ME"}}'
```

Where `username` and `password` are valid login credentials for ClusterControl. The output should be a load of JSON text returned with details of the clusters managed by ClusterControl.

### 2.101.2 Setting up the Terraform provider for ClusterControl

In your environment, create a `terraform.tfvars` file to store your ClusterControl secrets. In the file, add the following secrets:

```
cc_api_url="https://<cc-host-private_ip>:9501/v2"
cc_api_user="CHANGE-ME"
cc_api_user_password="CHANGE-ME"
```

Create a file (e.g `main.tf`) to store the ClusterControl Terraform provider configuration and the resources you will create. In the file, add the following configuration:

```
terraform {
  required_providers {
    clustercontrol = {
      source = "severalnines/clustercontrol"
      version = "0.2.16" // refer to terraform registry for latest version
    }
  }
}

variable "cc_api_url" {
  type = string
}

variable "cc_api_user" {
  type = string
}

variable "cc_api_user_password" {
  type = string
}

provider "clustercontrol" {
  cc_api_url = var.cc_api_url
  cc_api_user= var.cc_api_user
  cc_api_user_password= var.cc_api_user_password
}
```

The above Terraform code will install the Terraform Provider for ClusterControl, import the secrets from the `terraform.tfvars` and configure the provider for use across the environment.

Run `terraform init` to initialize the directory and install the provider. You can also verify the correctness of the configuration with `terraform validate` command.

With your ClusterControl Terraform provider configuration validated, you can now deploy database clusters.

### 2.101.3 Deploying database clusters using Terraform

To deploy a database cluster with the ClusterControl Terraform provider, you use the *clustercontrol_db_cluster* resource. The following example `clustercontrol_db_cluster` resource schema deploys a PostgreSQL cluster:

```
resource "clustercontrol_db_cluster" "test-postgresql" {

    db_cluster_create = true
    db_cluster_name = "PostgreSQL Cluster"
    db_cluster_type = "pg-replication"
    db_version = "15"
    db_vendor = "postgresql"
    db_admin_user_password = var.cc_api_user_password
    db_host {
        hostname = "test-primary"
    }
    db_host {
        hostname = "test-replica"
    }
    ssh_key_file = "/root/.ssh/id_rsa"
    ssh_user = "root"
    db_deploy_agents = true
    disable_firewall = true
    disable_selinux = true
    db_install_software = true

}
```

In the above resource:

- The `db_cluster_type`, `db_vendor` and `db_version` attributes define the type of cluster, vendor and version respectively.
- `db_cluster_create` tells the provider to create a new cluster. If you are importing an already existing cluster, you will use the `db_cluster_import` attribute.
- The `db_host` blocks are the list of nodes in the cluster. The hostname of the DB node can also be its IP address as well. You can configure your `db_host` node configurations with its nested schema.
- ClusterControl uses passwordless SSH to deploy and manage the database cluster automatically. The `ssh_user` attribute holds the SSH user ClusterControl will use to SSH to the DB nodes from the ClusterControl node, and `ssh_key_file` the path to the private key file for the Sudo user on the ClusterControl node.

- The `db_deploy_agents` attribute set to "true", automatically deploys prometheus and other relevant agents for observability after setting up the initial database cluster.
- If the nodes are new without any database package pre-installed, this resource has the `db_install_software` attribute set to "true". This tell ClusterControl to install all the database packages it needs from their respective repos, and `disable_firewall` and `disable_selinux` attributes disables security configurations on the node OS to enable ClusterControl download and installs the DB packages.

To apply the above resource run the following commands:

```
terraform plan    # to preview the configuration changes
terraform apply   # to apply the configuration
```

After a few minutes, the cluster creation will complete and you should see a final output as below.

```
clustercontrol_db_cluster.test-postgresql: Creation complete after 9m20s␣
↪[id=1]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

You can verify that the cluster is using the *ClusterControl GUI* or the *ClusterControl CLI (s9s-tools)* which both come with your ClusterControl installation.

For more examples, see the examples documentation on deploying other types of database clusters – MySQL/MariaDB replication or Galera with ProxySQL, MongoDB replica set and/or sharded, Redis Sentinel, etc. Navigate to the docs directory for generated documentation on the Terraform provider plugin for ClusterControl.

### 2.101.4 Destroying a deployed database cluster

After navigating to the appropriate directory which was used to deploy a cluster:

```
terraform destroy
```

ClusterControl supports integration with directory services like Active Directory, FreeIPA, and OpenLDAP authenticatio using lightweight directory access protocol (LDAP). This allows users to log into ClusterControl by using their corporate credentials instead of a separate password. LDAP groups can be mapped onto ClusterControl user groups to apply privileges to the entire group. This ensures that ClusterControl authorizes the logged-in user based on the group assigned. It supports up to the LDAPv3 protocol based on RFC2307.

To integrate with a directory service, one has to perform the following steps:

1. Fill up the ***LDAP Settings*** configuration wizard. See *LDAP Settings*.
2. Save the settings. At this point, LDAP is saved but not activated because no group mapping has been created yet.
3. Create at least one group mapping entry by going to ***Map LDAP Group***. See *Map LDAP Group*.
4. Enable the LDAP authentication by toggling on the ***LDAP Settings → Enable LDAP Authentication***.
5. Log in to ClusterControl by using the value of LDAP ***Username Attributes*** with the respective password. The user will be authenticated and redirected to the ClusterControl dashboard page based on the assigned group.

6. From this point, both ClusterControl native authentication and LDAP authentications are enabled.

**Warning**

Attention

The last entry of the group mapping can not be deleted while LDAP authentication is enabled and activated. See *Delete mapping group*.

## 2.102 LDAP settings

| Field | Description |
|---|---|
| *General Settings* | |
| **Enable LDAP Authentication** | Enables LDAP authentication. The native authentication in ClusterControl will also work. Refer to the steps mentioned above, on enabling LDAP authentication. |
| **LDAP URI** | Enter the LDAP or LDAPS Uniform Resource Identifier (URI), with the port number (if applicable). An example is `ldaps://ad.s9s.com:636`. For LDAPS, provide the certificates and key files under **SSL/TLS Settings** section. |
| **Login DN** | The Distinguished Name is used to bind the LDAP server. This user requires read access to all LDAP users and group entries to work correctly. ClusterControl must perform an LDAP search using the DN before any user can log in. This field is case-sensitive. An example is `cn=Administrator,cn=Users,dc=s9s,dc=com`. |
| **Login DN Password** | The password for Login DN. |
| **User Base DN** | The Distinguished Name (DN) to locate the users' information. This field is case-sensitive. An example is `cn=Users,dc=s9s,dc=com`. |
| **Group Base DN** | The Distinguished Name (DN) to locate the group information. ClusterControl does not support LDAP users that do not belong to at least one LDAP group. An example is `ou=Groups,dc=s9s,dc=com`. |
| *Advanced* | |
| **Prefill for** | Fill up all required fields based on common settings for Active Directory or OpenLDAP. Clicking on **Reset** will clear all textfields. |
| **User Base Filter** | Filter the object class of the LDAP users. If empty, all object classes will be returned. |
| **Username Attribute** | The LDAP attributes which hold the username, separated by a comma (whitespace value is not allowed). For Active Directory, this is commonly `sAMAccountName` and `uid` for OpenLDAP. When authenticating with ClusterControl (via ClusterControl GUI login page, ClusterControl CLI or RPC API), use the value of this attribute. If more than one attribute is specified, ClusterControl will attempt to look up all of them with the first non-empty reply (in the particular order) used for the login username. |
| **Real Name Attribute** | The LDAP attributes which hold the full name of the user, separated by a comma. For Active Directory, this is commonly `displayName` and `cn` for OpenLDAP. This value of this attribute will be used as the first and last name of the user. If more than one attribute is specified, ClusterControl will attempt to look up all of them with the first non-empty reply (in the particular order) used for the user's full name. |
| **Email Attribute** | The LDAP attributes which hold the email address of the user, separated by a comma. For Active Directory, this is commonly `userPrincipalName` and `mail` for OpenLDAP. This value of this attribute will be used as the email address of the user. If more than one attribute is specified, ClusterControl will attempt to look up all of them with the first non-empty reply |

## 2.103 Map LDAP group

ClusterControl requires at least one LDAP group mapping entry to be defined before the LDAP authentication can be activated. Otherwise, ClusterControl would only save the LDAP settings without activating them.

To create a group mapping:

1. Click on the ***Map LDAP Group*** button.
2. Choose an existing team from the dropdown and specify the value of the LDAP's ***Group Name Attributes*** that you want to be mapped with the ClusterControl team. For single-tier RDN, specify the attribute value for example "DBA". ClusterControl will prepend the **Group name attribute** configured under the **LDAP Settings** with the value specified, and append it with the **Group base DN** value. The final constructed DN will be `{Group name attribute}={value specified}, {Group base DN}`. For multi-tier RDNs, specify the attribute name and value (for multiple RDNs, delimited by a comma) for example `cn=DBA,cn=IT,ou=org1`. ClusterControl will append the value specified with the Group base DN value under the LDAP Settings. The final constructed DN will be `{value specified},{Group base DN}`.
3. After creating the first entry, you may proceed to activate the LDAP authentication by going to ***LDAP Settings → Enable LDAP Authentication***.

All LDAP configurations and mappings will be stored inside a configuration file at `/etc/cmon-ldap.cnf`. It is recommended to configure LDAP settings and group mappings via the ClusterControl GUI because any changes to this file will require a reload to the controller process, which is triggered automatically when configuring LDAP via the GUI.

**Note**

Note It is also possible to modify the configuration file directly, however, the `cmon` service must be reloaded manually by using the `systemctl restart cmon` command, or `service cmon restart`.

## 2.104 Delete mapping group

Click on **Delete** of the entry in the list to delete it. The last entry of the group mapping can not be deleted while LDAP authentication is enabled and activated (the button will be greyed out). To delete the last entry, set **Enable LDAP Authentication** to OFF and you will notice the **Remove Group** button is now clickable, allowing you to remove the last group mapping entry and effectively deactivating LDAP authentication for ClusterControl.

Configures third-party notifications on events triggered by ClusterControl. This feature allows users to integrate ClusterControl into your organization's communication channels, incident response systems, and workflow applications. To prevent users from receiving too many notifications, ClusterControl integrations also allow users to send out only specific critical or warning alerts (see *Warning Events* and *Critical Events*).

**See also**

Introducing the ClusterControl Alerting Integrations.

## 2.105 Supported integrations

Supported services are:

| Service type | Service provider |
|---|---|
| Incident management | PagerDuty Splunk On-Call (formerly known as VictorOps) OpsGenie ServiceNow ilert |
| Messaging platform | Slack Telegram |
| Others | Webhook |

## 2.106 Notification behavior

Within ClusterControl internal states, there are 3 types of events:

- CREATED - An alarm was raised.
- CHANGED - Something changed in the alarm event.
- ENDED - The alarm was resolved.

However, by default, the ClusterControl Notification process (`cmon-events`) will only listen to 2 events - **CREATED** and **ENDED** events - and pass them to the configured services. This can be configured with `allowed_events` CLI parameter.

The notification behavior is depending on the type of services that receives the event, as shown below:

| Service | Notification behavior |
|---|---|
| **Email** | CREATED - Will send an email ENDED - Will send an email |
| **Incident management** | CREATED - Will create an incident ENDED - Will resolve an incident |
| **Messaging platform** | CREATED - Will send a message ENDED - Will send a message |
| **Webhook** | CREATED - Will send a HTTP POST request to the webhook ENDED - Will send a HTTP POST request to the webhook See *Webhook integration* for examples. |

**Warning**

Attention

If you already have an alarm raised and then you create, for example, a Slack (or any other) integration, you will never see events for the alarms already raised/created before you created the Slack integration. You will only see alarms created after the integration is set up.

## 2.107 Set up notification

Every supported service has different requirements and configurations as shown in the following subsections.

### 2.107.1 Slack

Slack is a cloud-based collaboration platform designed for teams to communicate and work together in real time. For Slack integration, the following information are needed:

- **Slack workspace URL** - The webhook URLs for your workspace. Commonly it looks like this: `https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX`
- **Channel** - The Slack's channel name.

The following are the steps to get the required information:

1. Create a Slack app by going to Your Apps.
2. Go to ***Features → Incoming Webhooks*** and toggle on **Activate Incoming Webhooks**.
3. Click on **Add New Webhook to Workspace** and choose a Slack channel from the workspace in the dropdown and click **Allow**. Take note on the channel name that you choose.
4. Copy the generated **Webhook URL**. This is the Slack workspace URL that will be used in the integration.

**Tip**

See Slack documentation - Sending messages using incoming webhooks.

To configure Slack integration:

#### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Notification Services → Add new integration → Slack***.
2. In the **Service Configuration** section, specify the following:
   - **Integration name**: Give a name to this integration.
   - **Workspace URL**: The webhook URL of your workspace, taken from the Slack app page.
   - **Channel**: The channel that ClusterControl will post the events, prefixed it with a hash tag (#).
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

### 2.107.2 Telegram

Telegram is a cloud-based instant messaging app available on mobile, desktop, and web that lets you send text and voice messages, make voice and video calls, and share photos, videos, and files of any type. For Telegram integration, the following information are needed:

- **Telegram bot token** - The telegram bot API authentication token. Commonly it looks like this: `999999999:AAHfiqksKZ8WmR2zSjiQ7_v4TMAKdiHm8T0`.
- **Channel** - The Telegram's channel name.

The following are the steps to get the required information:

1. Open Telegram and create a new bot by contacting `@BotFather`.
2. Give the bot a unique name and copy the bot token. This is the Telegram bot token that is required.
3. Create a new Telegram channel (or use an existing channel).
4. Add the created bot to the channel as Administrator.
5. Allow your bot to send messages to the channel.
6. Copy channel ID from the channel link. For example, if the link is `t.me/mypowerfulchannel`, the channel ID is `mypowerfulchannel`.

**Tip**

See Telegram documentation - Bots.

To configure Telegram integration:

#### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Notification Services → Add new integration → Telegram***.
2. In the **Service Configuration** section, specify the following:
   - **Integration name**: Give a name to this integration.
   - **Telegram BOT token**: The telegram bot API authentication token, provided by Telegram bot `@BotFather`.
   - **Channel**: The channel that ClusterControl will post the events, prefixed it with an alias (`@`).
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

### 2.107.3 ServiceNow

ServiceNow is a cloud-based platform for digital workflow automation and enterprise service management. For ServiceNow integration, the following information are needed:

- **Username** - The instance username.
- **Password** - The password for the instance user.
- **Service** - The Service name that you want to monitor.
- **Configuration item** - The ServiceNow components that need to be tracked.
- **Instance** - The instance name.

**Tip**

See ServiceNow documentation.

To configure ServiceNow integration:

#### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Notification Services → Add new integration → ServiceNow***.
2. In the **Service Configuration** section, specify the following:
   - **Integration name**: Give a name to this integration.
   - **Username** - The instance username.
   - **Password** - The password for the instance user.
   - **Service** - The Service name that you want to monitor.
   - **Configuration item** - The ServiceNow components that need to be tracked.
   - **Instance** - The instance name.
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

### 2.107.4 PagerDuty

PagerDuty is an incident management platform for IT operations, enabling teams to detect and respond to service disruptions and outages in real time. For PagerDuty integration, the following information is needed:

- **Service key** - Integration key for PagerDuty Events API v2. Commonly it looks like this: `aaaaa75294fe496b8ba9a87f71bddddd`.

The following are the steps to get the required information:

1. Login to your PagerDuty account.
2. Create a new service by going to ***Services → Service Directory → New Service***.
3. Give the Service a name and generate an (or select an existing) escalation policy.
4. In the **Integration** section, choose "Events API V2". Generate a new **Integration Key** for this integration and this is the **Service key** that is required.

**Tip**

See PagerDuty documentation - Services and Integrations.

To configure PagerDuty integration:

#### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Notification Services → Add new integration → PagerDuty***.
2. In the **Service Configuration** section, specify the following:
   - **Integration name**: Give any name to this integration.
   - **Service Key**: The PagerDuty Events API v2 integration key of the chosen service.
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

### 2.107.5 Splunk On-Call (formerly known as VictorOps)

Splunk On-Call (formerly known as VictorOps), is an incident management platform for IT operations, enabling teams to detect and respond to service disruptions and outages in real time. For Splunk On-Call integration, the following information is needed:

- **REST Integration URL** - The Splunk On-Call Service API Endpoint you get when you enable the REST integration. Commonly it looks like this: `https://alert.victorops.com/integrations/generic/<YOUR_REST_ENDPOINT_KEY>/alert/<YOUR_ROUTING_KEY>`.

The following are the steps to get the required information:

1. Login to your Splunk On-Call/VictorOps account.
2. Create a new integration by going to ***Settings → Integrations → 3rd Party Integrations → REST – Generic***.
3. Click on the **Enable** button to generate an endpoint destination URL.

4. Routing keys in Splunk On-Call can be set up and associated by clicking on ***Settings → Routing Keys***. Routing key must be included as part of the **REST integration URL**.

**Tip**

See Splunk On-Call documentation - REST Endpoint Integration Guide.

To configure Splunk On-Call integration:

#### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Notification Services → Add new integration → VictorOps***.
2. In the **Service Configuration** section, specify the following:
   - **Integration name**: Give any name to this integration.
   - **REST Integration URL**: The Splunk On-Call integration URL for REST API. The URL shall include the REST endpoint key and routing key.
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

### 2.107.6 OpsGenie

Opsgenie is a cloud-based incident management and on-call scheduling platform (part of the Atlassian suite) that centralizes alerts from your monitoring, ticketing, and collaboration tools. For Opsgenie integration, the following information are needed:

- **Region** - The Opsgenie data-center your account lives in, so your HTTP calls go to the correct endpoint. It is either US or EU.
- **Teams** - The Opsgenie team identifiers or names that should receive and handle the alerts generated by this integration.
- **API key** - The unique "GenieKey" token created when you add an API integration under ***Settings → Integrations → API***.

The following are the steps to get the required information:

1. Login to your OpsGenie account.
2. Create a new integration by going to ***Settings → Integrations → API***.
3. Click **Continue**, expand **Steps to configure the integration**, then copy the **API key**.
4. Click **Turn on integration** to enable it.

**Tip**

See Opsgenie Support - Create an API Integration.

To configure OpsGenie integration:

#### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Notification Services → Add new integration → OpsGenie***.
2. In the **Service Configuration** section, specify the following:
   - **Region**: Choose whether US or EU region.
   - **Teams**: The Splunk On-Call integration URL for REST API. The URL shall include the REST endpoint key and routing key.
   - **API Key**: The unique "GenieKey" token created when you add the API integration.
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

### 2.107.7 ilert

ilert is an AI-first company offering an all-in-one incident management tool for alerting, on-call management, and incident communication to help companies increase their digital uptime. B2C and B2B companies from across the globe, including well-known brands such as IKEA, Lufthansa Systems, and NTT Data, trust ilert to empower their operations teams and ensure everything is running smoothly. For ilert integration, the following information is needed:

- **Alert source URL** - The ilert API alert source URL generated in the ClusterControl integration page for ilert. Commonly it looks like this: `https://api.ilert.com/api/v1/events/clustercontrol/xxxxxxxxxxxxxxxxxxxxxxxxxxxx`.

The following are the steps to get the required information:

1. Login to your ilert account.
2. Go to ***Alert sources → Alert sources*** and click ***Create new alert source***.
3. Search for "ClusterControl" in the search field, click the ClusterControl tile, and then **Next**.
4. Give your alert source a name, optionally assign teams, and click **Next**.
5. Select an escalation policy by creating a new one or assigning an existing one.
6. On the final page, an ClusterControl URL will be generated. Use this information for the **URL** when you configure ClusterControl integration.

**Tip**

See ilert Documentation - ClusterControl Integration.

To configure ilert integration:

### ClusterControl GUI

1. Go to *ClusterControl GUI → Settings → Notification Services → Add new integration → ilert*.
2. In the **Service Configuration** section, specify the following:
   - **Integration name**: Give any name to this integration.
   - **URL**: The ilert ClusterControl integration URL generated from the alert source integration dashboard.
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

## 2.108 Webhook integration

A webhook is a lightweight API that powers one-way data sharing triggered by events. Together, they enable applications to share data and functionality, and turn the web into something greater than the sum of its parts. APIs and webhooks both allow different software systems to sync up and share information. For webhook integration, the following information is needed:

- **URL** - The HTTP endpoint of the service that should receive the notifications. Make sure the service is properly set up to handle webhook events.

For webhook integration, ClusterControl will raise an alarm by sending the JSON data using the HTTP POST method to the configured endpoint. The webhook endpoint should receive the following example event if a new alarm is created:

```
{
  "id": 470,
  "status": "CREATED",
  "component": "Node",
  "hostname": "192.168.20.62",
  "title": "Server disconnected",
  "message": "PostgreSQL server on 192.168.20.62:5432 disconnected: Connect␣
↪failure: Watchdog: Failed connection to postgres@192.168.20.62:5432.␣
↪timeout expired\n",
  "recommendation": "Check node status on UI and error log of failed␣
↪server.",
```

(continues on next page)

(continued from previous page)

```
  "severity": "CRITICAL"
}
```

The above event was created and sent out to the webhook endpoint after ClusterControl has detected that one of our database servers (192.168.20.62) was down. After the server came back online, ClusterControl would then send another event with the same alarm ID (470) with a different status "ENDED", indicating the above alarm has ended and the node is back operational:

```
{
  "id": 470,
  "status": "ENDED",
  "component": "Node",
  "hostname": "192.168.20.62",
  "title": "Server disconnected",
  "message": "PostgreSQL server on 192.168.20.62:5432 disconnected: Connect␣
↪failure: Watchdog: Failed connection to postgres@192.168.20.62:5432.␣
↪timeout expired\n",
  "recommendation": "Check node status on UI and error log of failed␣
↪server.",
  "severity": "CRITICAL"
}
```

Commonly, the "ended" event has an almost identical response text with the "created" event except for the "status" value.

**Tip**

You may use https://webhook.site/ to test out the webhook integration with ClusterControl.

To configure webhook integration:

### ClusterControl GUI

1. Go to ***ClusterControl GUI → Settings → Notification Services → Add new integration → Webhook***.
2. In the **Service Configuration** section, specify the following:
   - **Integration name**: Give any name to this integration.
   - **URL**: The HTTP endpoint of the service that should receive the notifications.
3. Then click on **Test credentials**. Ensure you receive a test notification from ClusterControl before proceeding to the next step.
4. In the **Notification settings** section, specify the following:
   - **Clusters** - Choose one or more clusters to be notified. Multiple values are accepted.
   - **Events** - Choose one or more ClusterControl events that will trigger the notification. Multiple values are accepted. Details on the events are described in *Warning Events* and *Critical Events*.
5. Click **Finish** to complete the configuration.

## 2.109 Events

The following table lists out all event groups supported by ClusterControl:

| Event | Description |
|---|---|
| All Events | All ClusterControl events including warning and critical events. |
| All Warning Events | All ClusterControl warning events, e.g. cluster degradation, network glitch. See *Warning Events*. |
| All Critical Events | All ClusterControl critical events, e.g. cluster failed, host failed. See *Critical Events*. |
| Network | Network-related events, e.g. host unreachable, SSH issues. |
| CMON Database | Internal CMON database-related events, e.g. unable to connect to CMON database, `datadir` mounted as read-only. |
| Mail | Mail system-related events, e.g. unable to send mail, mail server unreachable. |
| Cluster | Cluster-related events, e.g. cluster failure, cluster degradation, time drifting. |
| Cluster Configuration | Cluster configuration events, e.g. SST account mismatch. |
| Cluster Recovery | Recovery events, e.g. cluster or node recovery failures. |
| Node | Node-related events, e.g. node disconnected, missing GRANT, failed to start HAProxy, failed to start NDB cluster nodes. |
| Host | Host-related messages, e.g. CPU/disk/RAM/swap exceeds thresholds, memory full. |
| Database Health | Database health-related events, e.g. memory usage of MySQL servers, connections, missing primary key. |
| Database Performance | Alarms for long-running transactions, replication lag, and deadlocks. |
| Software Installation | Software installation-related events, e.g. license expiration. |
| Backup | Backups related events, e.g. backup failed. |

### 2.109.1 Warning Events

**Note**

Note ClusterControl uses the same alarms' code name for other database clusters which produces a similar result. For example, if a PostgreSQL slave is lagging behind, the alarms internal code name is also called "MySqlReplicationLag". However, the actual alarm's response will have proper PostgreSQL relevant texts. This is handled internally.

| Area | Alarms | Severity | Description |
|---|---|---|---|
| Node | MySqlReplicationLag | Warning | MySQL replication slave lag, default 10 seconds. |
| | MySqlReplicationBroken | Warning | The SQL thread has stopped. For PostgreSQL, it means the slave gets disconnected from the master. |
| | CertificateExpiration | Warning | SSL certificate expiration time (<=31 days, >7 days). |
| | MySqlAdvisor | Warning | Raised by `wsrep_sst_method.js` and `wsrep_node_name.js` advisors. |
| | MySqlTableAnalyzer | Warning | Raised by `schema_check_nopk.js` advisor. |
| | StorageMyIsam | Warning | Raised by `schema_check_myisam.js` advisor. |
| | MySqlIndexAnalyzer | Warning | Raised by `schema_check_dupl_index.js` advisor. |
| | MySqlReplicationLooseServer | Warning | A slave is found but its master can't be determined, or it is not part of the cluster. |
| Host | HostSwapV2 | Warning | If a configurable number of pages has been swapped in/out during a configurable period of time. Default 20 pages in 10 minutes. |
| | HostSwapping | Warning | >5% swap space has been used. |
| | HostCpuUsage | Warning | >80%, <90% CPU used. |
| | HostRamUsage | Warning | >80%, <90% RAM used. |
| | HostDiskUsage | Warning | >80%, <90% disk space used on a monitored_mountpoint. |
| | ProcessCpuUsage | Warning | >95 % CPU used on average by a process for 15 minutes. |
| Back | BackupFailed | Warning | The backup job fails. |
| Recovery | GaleraWsrepMissing | Warning | `wsrep_cluster_address` or `wsrep_provider` is missing. |
| | GaleraSstAut | Warning | SST settings (user/pass are wrong). |
| Network | HostFirewall | Warning | The host is not responding to ping after 3 cycles. |
| | HostSshSlow | Warning | It takes 6-12 seconds to SSH into a host. |
| Cluster | ClusterTimeDrift | Warning | Time drift between ClusterControl and database nodes. |
| | ClusterLicenseExpire | Warning | The license is about to expire. |
| | ClusterInconsistentView | Warning | The load balancer or ClusterControl sees a different set of working nodes (master is down from ClusterControl point-of-view, while load balancer or the slave reports the master working.) |

### 2.109.2 Critical Events

**Note**

Note ClusterControl uses the same alarms' code name for other database clusters which produces a similar result. For example, if a PostgreSQL server goes down, the alarms internal code name is also called "MySqlDisconnected". However, the actual alarm's response will have proper PostgreSQL relevant texts. This is handled internally.

| Area | Alarms | Severity | Description |
|---|---|---|---|
| Node | MySqlDisconnected | Critical | The database server cannot be reached. |
| | MySqlGrantMissing | Critical | Node does not have the correct privileges set for the cmon user. |
| | MySqlLongRunningQuery | Critical | If queries are running for too long time. Only used if configured, by default it is not. |
| | ProcFailedRestart | Critical | A process (HAProxy, ProxySQL, Garbd, MaxScale) could not be restarted after a failure. |
| | CertificateExpiration | Critical | (<= 7 days), SSL Certificates expiration time. |
| | MySqlReplicationMultiMaster | Critical | Multiple writable masters detected. |
| | MySqlReplicationFailoverDataLoss | Critical | When automatic failover promotes a replica despite unapplied relay log events lost to a mid-event-group source failure. |
| Host | HostSwapV2 | Critical | If a configurable number of pages has been swapped in/out during a configurable period of time. Default 20 pages in 10 minutes. |
| | HostSwapping | Critical | >20% swap space has been used. |
| | HostCpuUsage | Critical | >90% CPU used. |
| | HostRamUsage | Critical | >90% RAM used. |
| | HostDiskUsage | Critical | >90% disk space used on a `monitored_mountpoint`. |
| | ProcessCpuUsage | Critical | >99 % CPU used on average by a process for 15 minutes. |
| Backu | BackupVerificationFailed | Critical | Backup verification fails. |
| Recovery | GaleraWsrepMissing | Critical | `wsrep_cluster_address` or `wsrep_provider` is missing, and still missing after 20 sample cycles (which are ~100 seconds in this case) |
| | GaleraClusterSplit | Critical | There is a split brain. |
| | ClusterRecoveryFail | Critical | Recovery has failed. |
| | GaleraConfigProblem1 | Critical | A configuration issue preventing the node to start. |
| | GaleraNodeRecoveryFail | Critical | Automatic recovery has failed 3 consecutive times. |
| | ReplicationFailoverBlacklistError | Critical | In the case of automatic failover, the only possible candidate is blacklisted, then this alarm is raised with critical severity. |
| Network | HostUnreachable | Critical | The host is not responding to ping after 3 cycles. |
| | HostSshFailed | Critical | Please check SSH access to the host. The host may also be down. |
| | HostSshAuth | Critical | Please check whether the configured SSH key is properly configured and can be authenticated on the host. |
| | HostSudoError | Critical | `sudo` command error on the host. |
| | HostSshSlow | Critical | It takes more than 12 seconds to SSH into a host. |

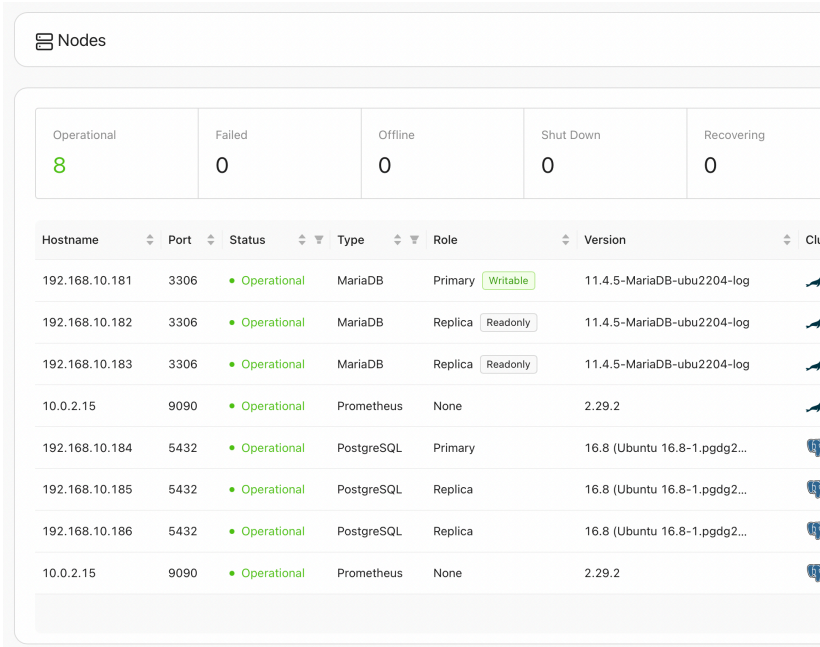
## 2.110 Setting up dedicated Prometheus node

When deploying the first cluster, ClusterControl sets up a Prometheus instance on the ClusterControl node. That instance serves as a time-series database to keep the data collected from nodes by exporters. That Prometheus instance, by default, is reused for all other clusters that will be deployed in ClusterControl.

In some cases this setup may be not enough:

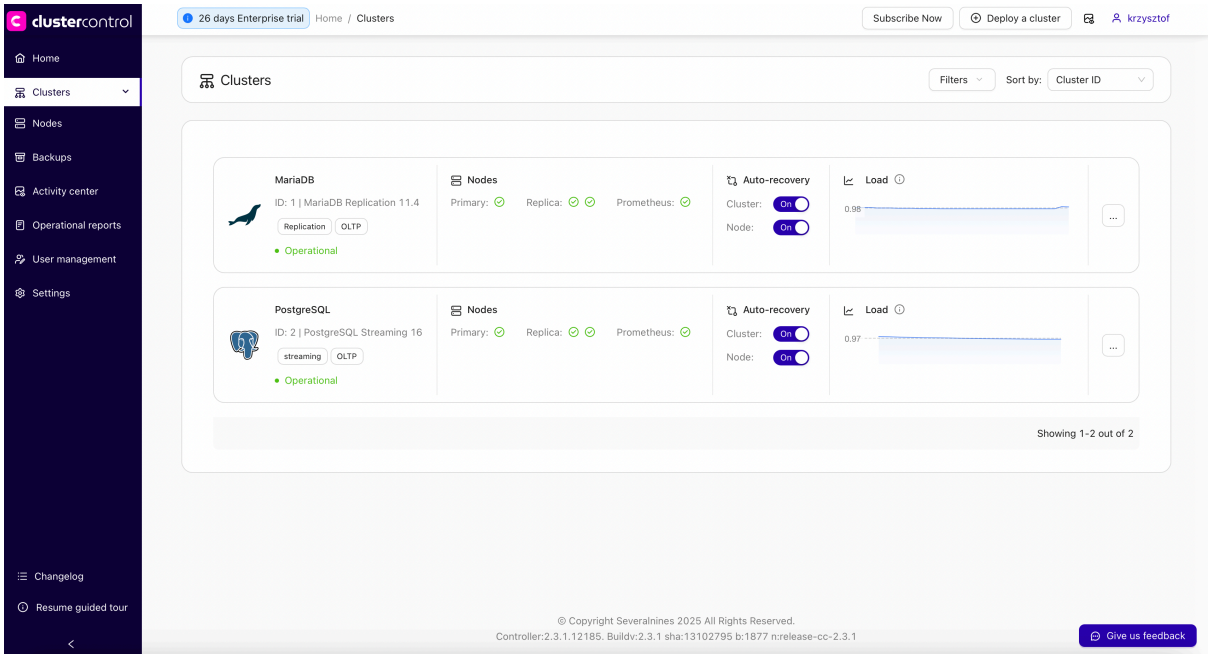
- You may have very large cluster that is generating significant amount of metrics
- You may have many clusters and, in total, the amount of data is big
- You may want to increase the retention period from default 15 days to something longer, causing more data to be stored

In all those cases you may find that you could use a dedicated node for Prometheus - faster, with more disk space available for data storage. Such setup is possible in ClusterControl.

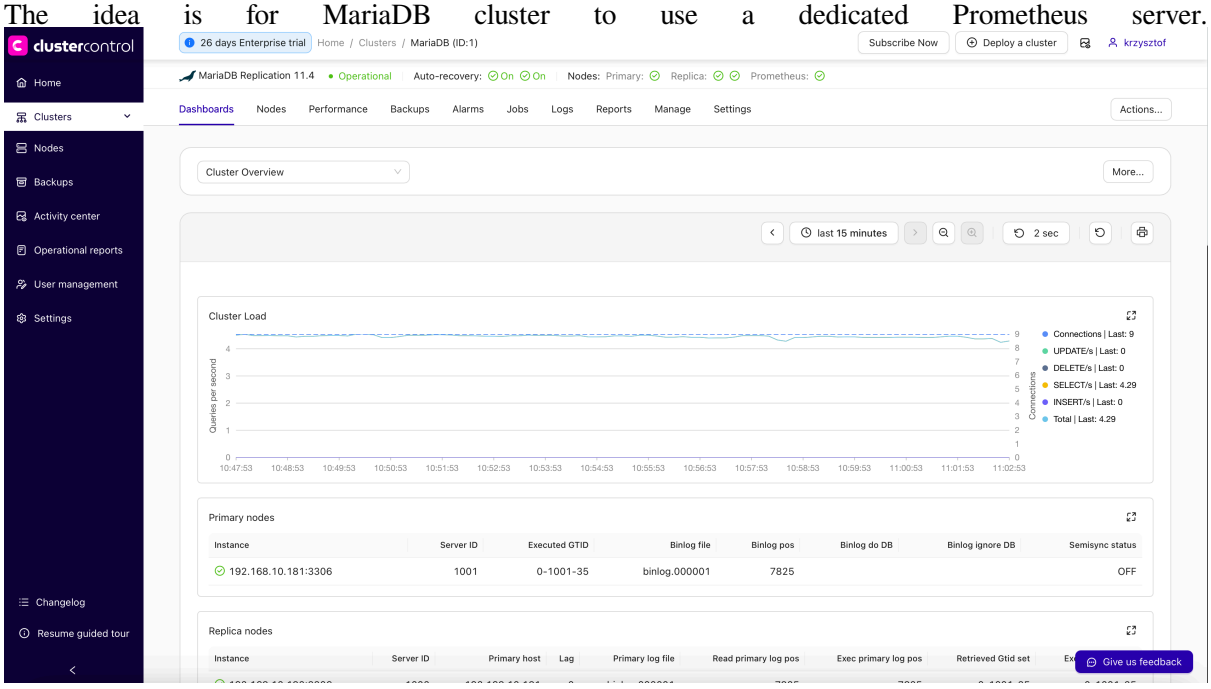



How to do this? Assume that there are two clusters:

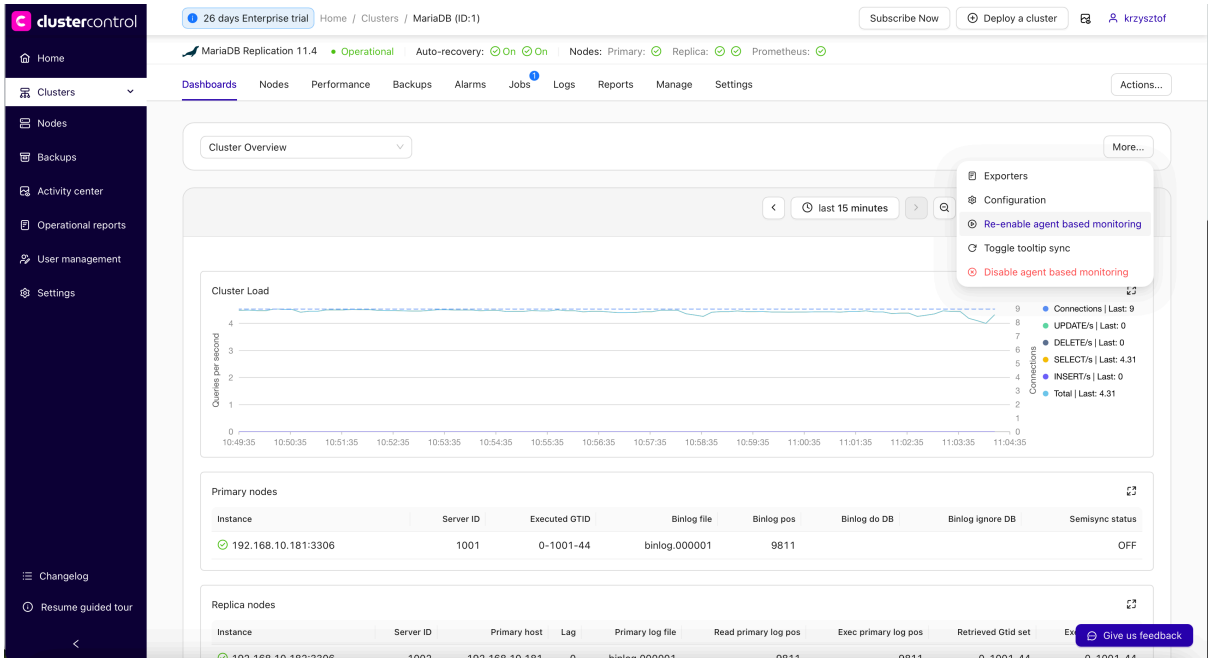
They are sharing the same Prometheus server on host 10.0.2.15 and port 9090.



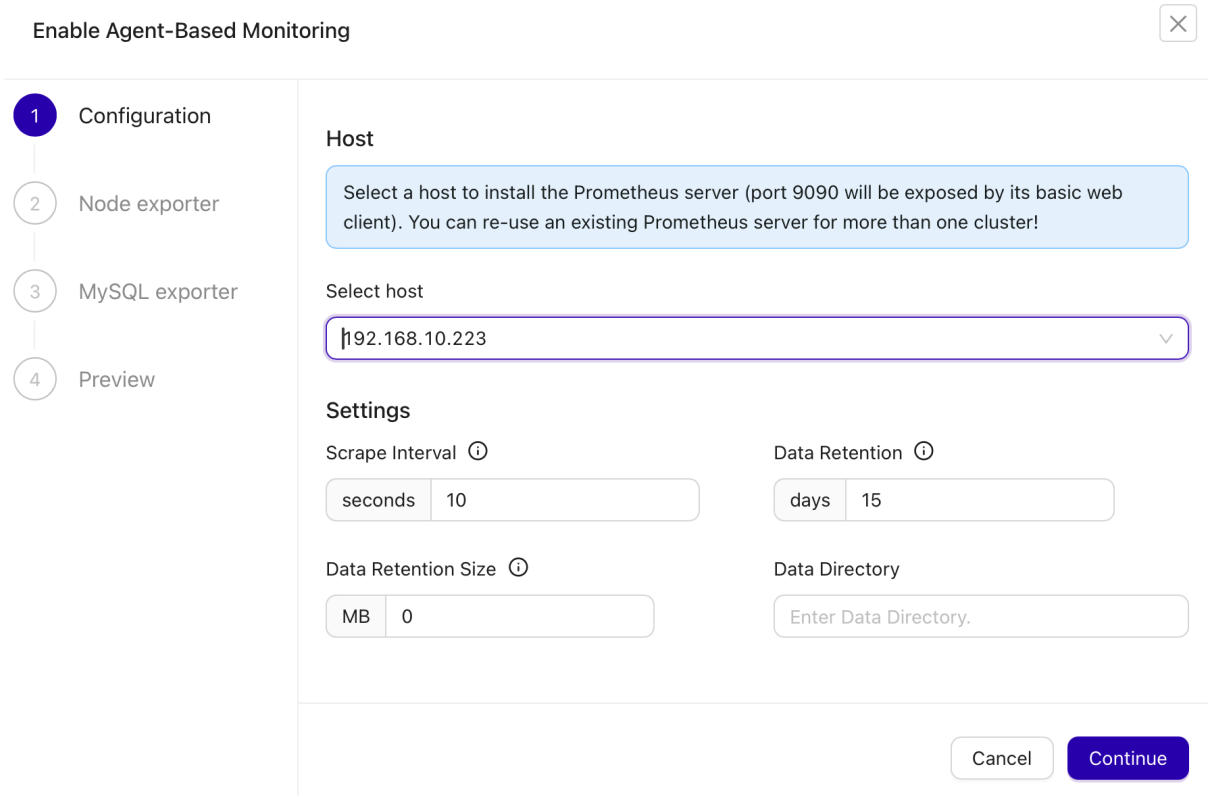

The idea is for MariaDB cluster to use a dedicated Prometheus server.




To do that, open MariaDB cluster and go to Dashboards tab. From there you pick re-enable and reconfigure agent based monitoring.




More→Re-enable agent based monitoring. This opens a window to perform the reconfiguration.




There are settings to configure:

### 2.110.1 Host Selection

Determines on which host the Prometheus server will be installed. You can ether pick existing Prometheus installation or put in a new hostname or IP address for a new Prometheus installation.

### 2.110.2 Scrape Interval

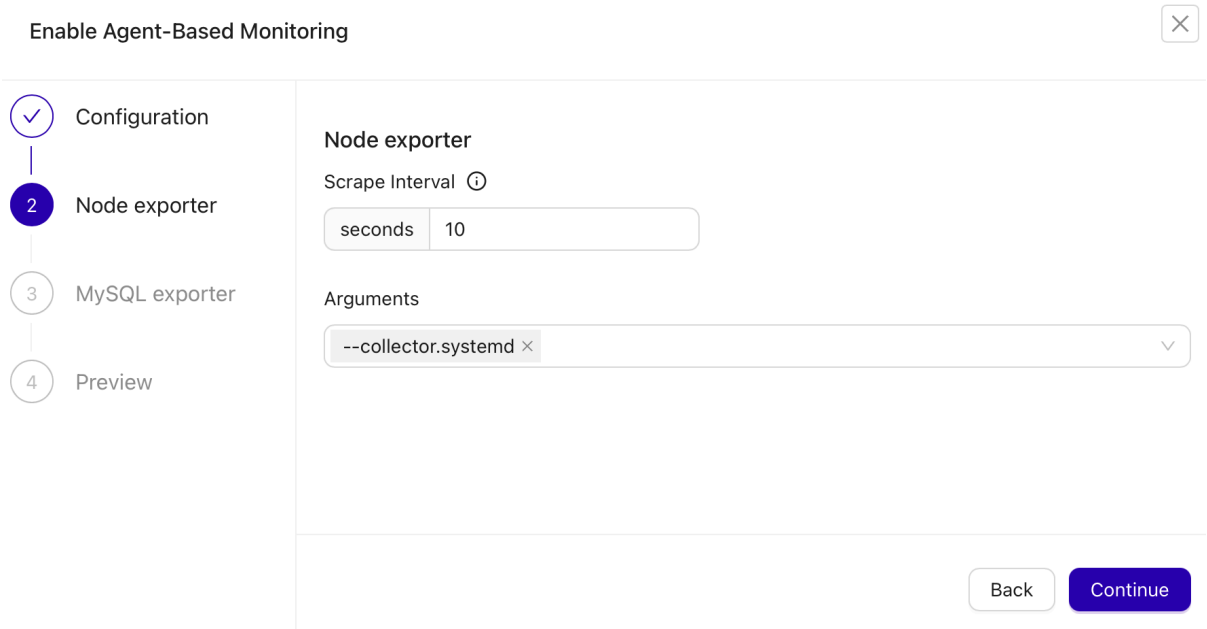
Defines how frequently Prometheus collects (scrapes) metrics from the targets. Default is set to 10 seconds.

### 2.110.3 Data Retention Size

Indicates the maximum amount of storage (MB) allocated for metrics data. Default (0 MB) means no limit.

### 2.110.4 Data Directory

Specifies the location on disk where Prometheus will store metrics data. Optional field (can be left unspecified).



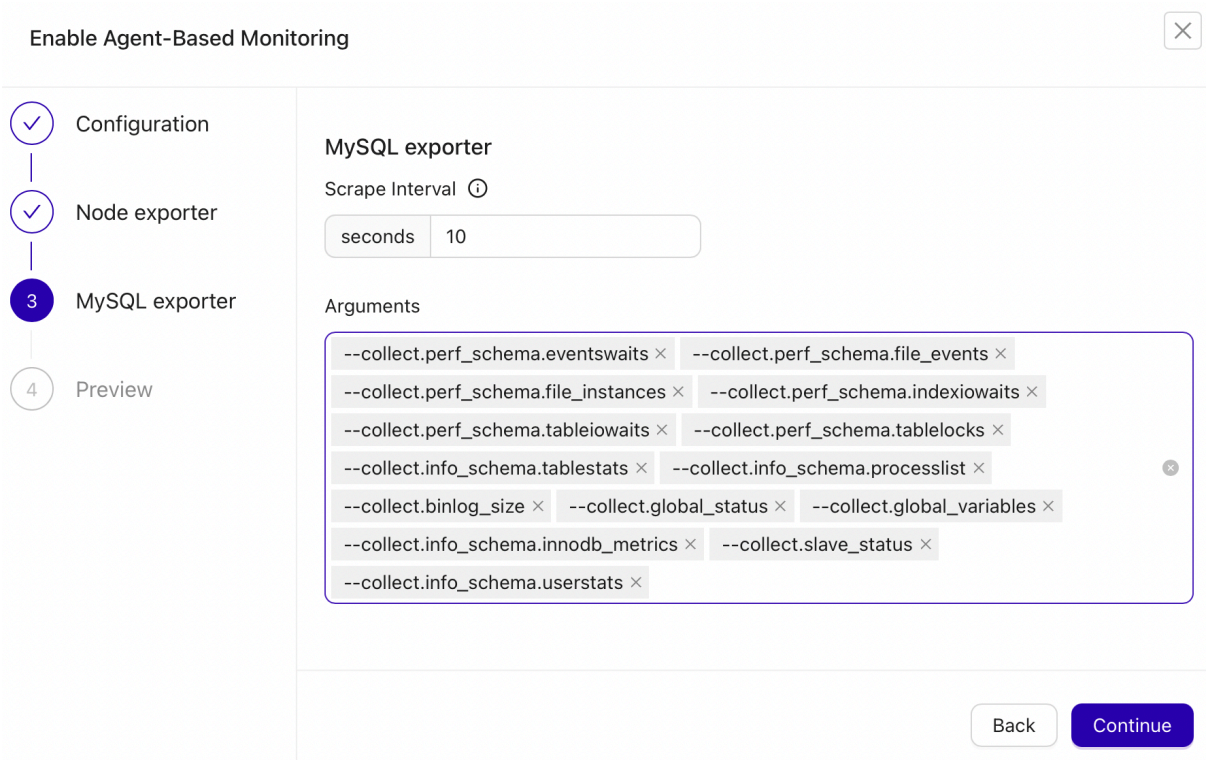

In this step you can configure settings related to node exporter.

### 2.110.5 Scrape Interval

Frequency at which Prometheus retrieves system metrics from the node exporter (10 seconds by default).

### 2.110.6 Arguments

Custom parameters for Node Exporter functionality, for instance: `--collector.systemd` enables collecting metrics from the systemd service manager.




Here are settings related to MySQL exporter

### 2.110.7 Scrape Interval

How often Prometheus scrapes metrics from the MySQL exporter (10 seconds).

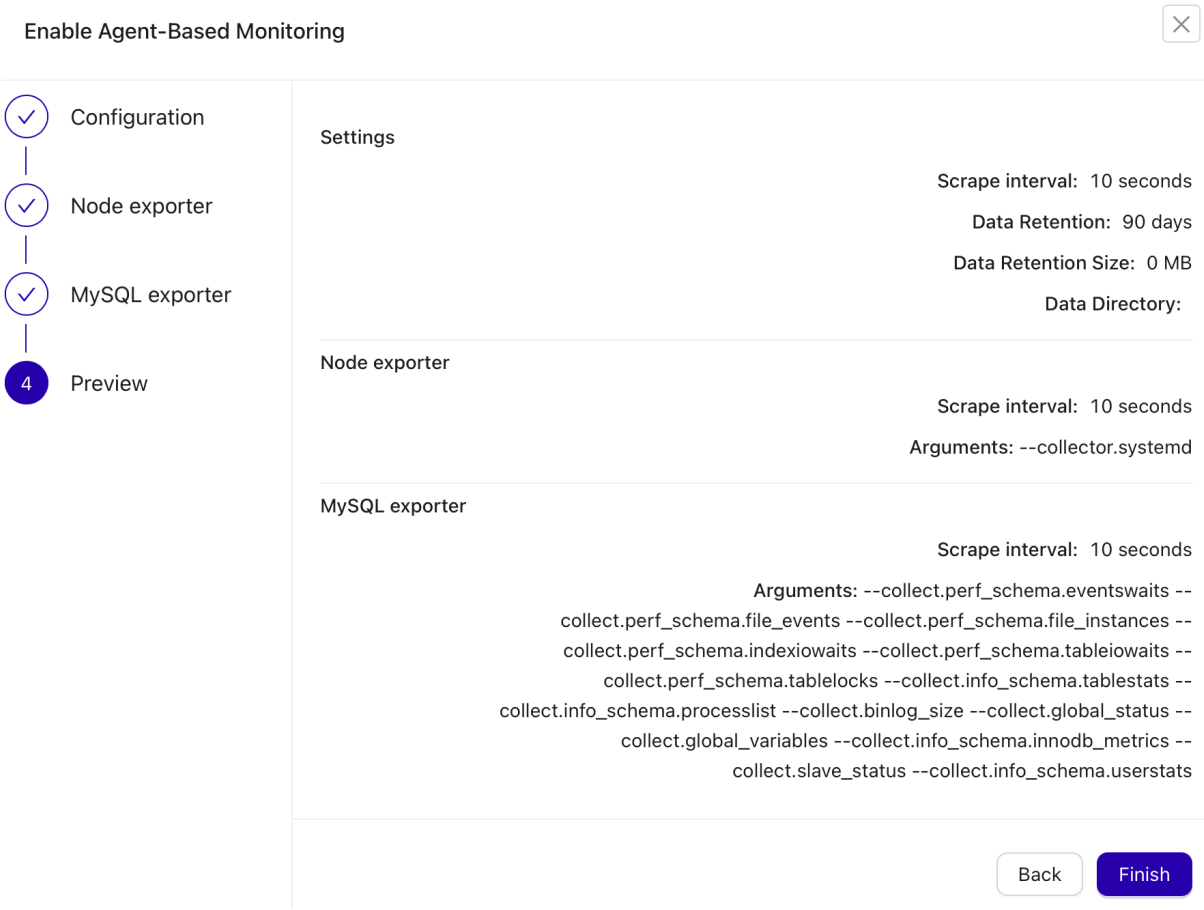
### 2.110.8 Arguments

Additional parameters to specify exactly which MySQL metrics should be collected, such as:

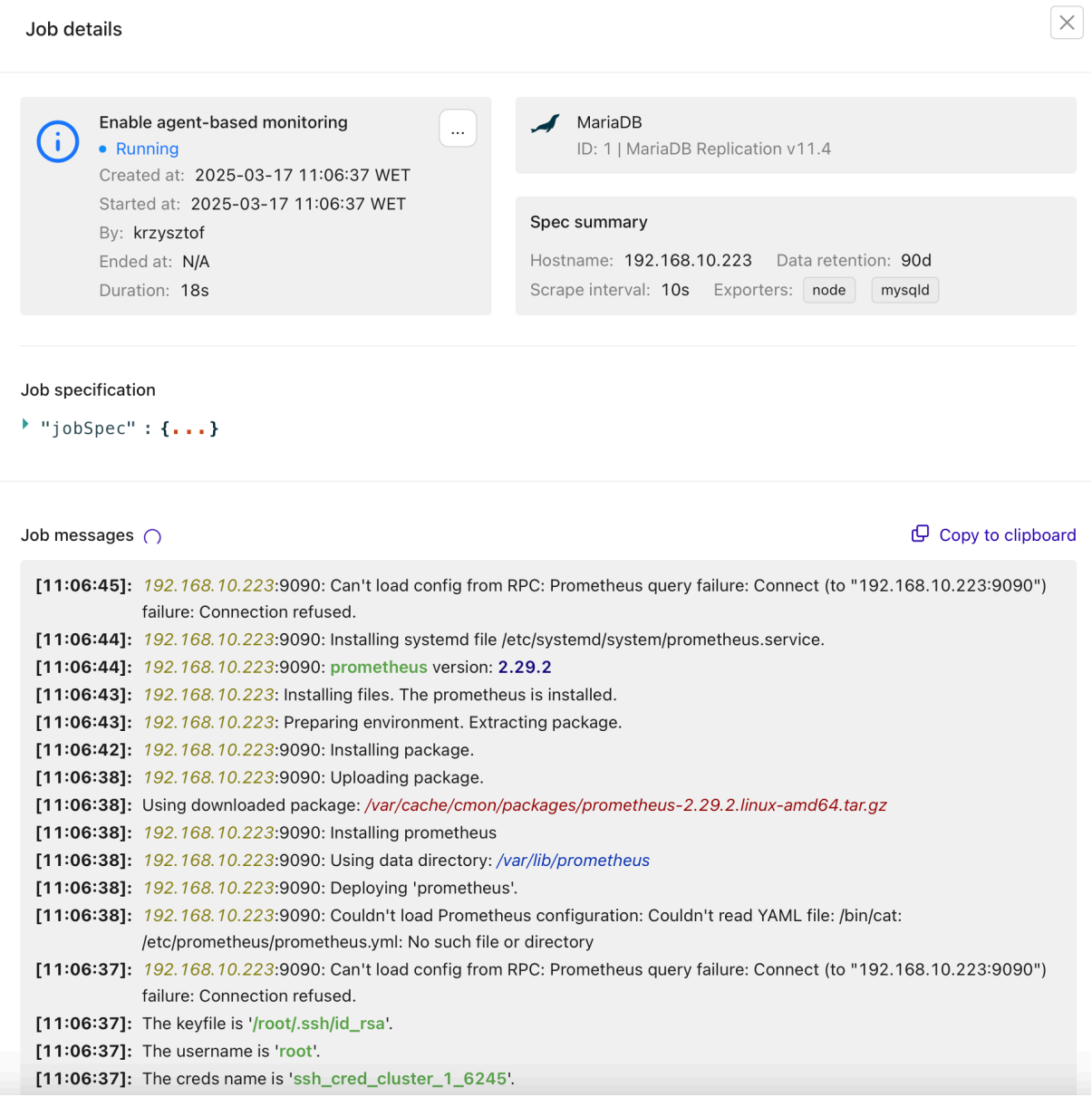
- `--collect.perf_schema.eventswaits` (Wait events)
- `--collect.perf_schema.file_events` (File operation events)
- `--collect.perf_schema.indexiowaits` (Index I/O waits)
- `--collect.perf_schema.tableiowaits` (Table I/O waits)
- `--collect.info_schema.processlist` (Active MySQL processes)
- `--collect.binlog_size` (Binary log size)
- `--collect.global_status` (Global status variables)
- `--collect.global_variables` (Global MySQL variables)
- `--collect.slave_status` (Replication slave status)

and other schema-related metrics.

Once all settings are defined, verify that everything is exactly as you want it to be in the Preview:




After the job is started, you can track it in the Activity center:




This section provides information is your go-to resource for diagnosing and resolving issues that may arise within the ClusterControl platform.

Note that this troubleshooting guide only covers the latest version of ClusterControl. We highly advise you to upgrade to the latest version before troubleshooting any kind of issues related to ClusterControl due to the following reasons:

- ClusterControl has a short release cycle if compared to other software applications (commonly a new major release every quarter of the year).
- We regularly release maintenance build of the latest stable version (if there is). See Release Notes.
- ClusterControl has to keep up with all the latest changes and modifications introduced by supported database and application vendors.
- Some of the issues you encountered probably have been fixed in the latest maintenance patch or major release.

To upgrade ClusterControl to the latest version, see *Upgrading ClusterControl*.

## 2.111 Logging

ClusterControl consists of a number of components that write to their own log file. These files reside on the ClusterControl node. By default, `cmon` runs without the debug option enabled. See *Debugging* on how to get the run in debug mode.

If you encounter any problems with ClusterControl, it is highly recommended to examine the related log files.

| Log Type | Default location (RedHat/CentOS) | Default location (Debian/Ubuntu) |
|---|---|---|
| CMON process log | `/var/log/cmon.log or /var/log/cmon_<cluster_id>.log` | `/var/log/cmon.log` or `/var/log/cmon_<cluster_id>.log` |
| CMON cloud log | `/var/log/cmon-cloud.log` | `/var/log/cmon-cloud.log` |
| CMON events log | `/var/log/cmon-events.log` | `/var/log/cmon-events.log` |
| CMON audit log | `/var/log/cmon_audit.log` | `/var/log/cmon_audit.log` |
| CMON upgrade log | `/var/log/cmon_upgrade.log` | `/var/log/cmon_upgrade.log` |
| CMON discovery log | `/var/log/cmon_discovery.log` | `/var/log/cmon_discovery.log` |
| Apache error log | - `/var/log/httpd/ssl_error_log` - `/var/log/httpd/error_log` | `/var/log/apache2/error.log` |
| Apache access log | - `/var/log/httpd/access_log` - `/var/log/httpd/ssl_access_log` | - `/var/log/apache2/access.log` - `/var/log/apache2/ssl_access.log` |

## 2.112 Debugging

The debugging section describes more advanced troubleshooting in ClusterControl. It's usually used for cases related to `cmon` crashes and requires in-depth investigation.

### 2.112.1 Debugging ClusterControl Controller (CMON)

In case if you encounter a CMON crash, please install the debuginfo package and the necessary packages as shown below.

### 2.112.2 Install debugging components

#### Red Hat/CentOS

1. Enable the debug repo under `/etc/yum.repos.d/CentOS-Debuginfo.repo` and set `enabled=1`.
2. Install yum utilities:

```
yum -y install yum-utils
```

3. Install ClusterControl debuginfo and gdb:

```
yum -y install clustercontrol-controller-debuginfo gdb
```

4. Then, run:

```
debuginfo-install clustercontrol-controller
```

#### Debian/Ubuntu

1. Install ClusterControl debuginfo package and gdb:

```
apt-get install clustercontrol-controller-dbg gdb
```

Optionally, you can

2. Install the debugging components' library:

```
apt-get install libstdc++6-4.8-dbg libc6-dbg
```

However, this totally depends on the libstdc++6 version installed. Print the shared object dependencies using `ldd`:

```
ldd /usr/sbin/cmon | grep libstdc
        libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6␣
↪(0x00007ff508001000)
```

Based on the library path, locate the package name that provides this library:

```
dpkg -S /usr/lib/x86_64-linux-gnu/libstdc++.so.6
libstdc++6:amd64: /usr/lib/x86_64-linux-gnu/libstdc++.so.6
```

Then, find the package's version:

```
dpkg -l | grep libstdc++6
ii libstdc++6:amd64 4.9.2-10 amd64 GNU Standard C++ Library v3
```

In this case, we have version "4.9" installed for libstc++6. Finally, install the corresponding debug packages:

```
apt-get install gdb libc6-dbg libstdc++-6-4.9-dbg
```

### 2.112.3 Debugging Steps

Debugging is a program that produces a core dump. It consists of the recorded state of the working memory of a computer program at a specific time, generally when the program has crashed or otherwise terminated abnormally. ClusterControl Controller (CMON) package comes with a crown file installed under `/etc/cron.d/` which will auto-restart if the cmon process is terminated abnormally. Typically, you may notice if the cmon process has crashed by looking at the `dmesg` output.

In such cases, generating a core dump is the only way to backtrace the issue. Make sure you have the debugging components installed as described in the previous section beforehand. On ClusterControl node as the root user, increase the CPU limit, adjust the kernel's core pattern value and run CMON on the foreground:

```
systemctl stop cmon
service cmon stop # for systemd only, to disable /etc/cron.d/cmon
ulimit -c unlimited
echo "/tmp/core.%e.%p.%h.%t" > /proc/sys/kernel/core_pattern
cmon -d
```

Or, if you want to run CMON as the background process, we need to create a unit override file for the cmon service:

```
systemctl edit cmon
```

And add the following lines:

```
[Service]
LimitCORE=infinity  # only when debugging
```

Then, you may start/stop the service using the `systemctl` command:

```
systemctl daemon-reload
systemctl start cmon
```

**Warning**

Warning!

Only set `LimitCORE=infinity` for debugging purposes. To revert back to the standard setting, remove `/etc/systemd/system/cmon.service.d/override.conf` and reload systemd using the `systemctl daemon-reload` command.

When `cmon` crashes, there will now be a core file under `/tmp`. Compress the core dump (gzip is recommended) and attach it to a support ticket so we can take a look and perform the necessary fix. Alternatively, you can send only the backtrace in a support ticket by using the following command:

```
gdb /usr/sbin/cmon /tmp/<corefile>
thread apply all bt full
```

Attach the full output and potentially replace sensitive information with "XXXXXXXXX". Traces may contain password information.

## 2.113 CMON on Foreground

If you would like to run cmon as a foreground process, you can do that by invoking `-d` option:

```
$ service cmon stop
$ CMON_DEBUG=1 cmon -d
```

CMON will enable `LOG_DEBUG` messages and print detailed information on the screen (stdout) as well as `/var/log/cmon.log` or `/var/log/cmon_{cluster ID}.log`. Press ++ctrl++ + C to terminate the process. In some cases, this type of CMON output might be needed to get insight into the problem.

## 2.114 ClusterControl GUI diagnostic

For ClusterControl GUI-related issue, you can use **Diagnostic** feature available at ***ClusterControl GUI → Settings → Diagnostics*** and toggle ON **UI audit data collection**. This allows user to record and capture the necessary data while reproducing an issue in the user interface. Once the issue has been replicated, click on "Download audit data" for analysis or to share with our Support team for troubleshooting and debugging.

**Warning**

Warning!

Remember to switch off the setting afterward to stop audit data collection.

**Note**

Note Note that we are not collecting any personal data during this process.

This section describes on how to troubleshoot the database cluster. It can be a new deployment, import a cluster or the existing database cluster managed by ClusterControl.

## 2.115 New deployment abd import cluster

The new deployment of database cluster through ClusterControl will trigger a new job either it is succesfull or failed. How to check if the deployment of new cluster is failed and what is the reason of the failed can be checked in the jobs.

### ClusterControl GUI

1. In the ClusterControl UI, go to ***Activity Center → Jobs***, check the failed jobs for the deployment cluster.
2. There is an Action menu in the jobs, expand the details for detailed errors.

### ClusterControl CLI

1. Check the history of the job

```
s9s jobs --list --show-failed
```

2. Dig the detailed error of failed job

```
s9s job --log --job-id=<value>
```

The `job-id` can be found from the job list output above.

## 2.116 Existing database cluster

Failure can occur in the database cluster due to various reasons. It may stem from issues in the network, disk failures, or problems within the operating system itself. It is important to know how to monitor and assess what is happening within the database cluster.

Below are steps to check the database cluster in ClusterControl.

### ClusterControl GUI

1. In the ClusterControl UI, go to **Home** page of ClusterControl, it will shown what cluster and nodes that is degraded.
2. Go to the failure Cluster, check the triggered Alarm under the **Alarm**.
3. If the failure are related to the database, check the database log under the ***Logs → System Logs***

### ClusterControl CLI

1. Run the following command in ClusterControl host. The cluster list give information about the list of the cluster including the state (`STARTED`, `FAILURE`, `UNKNOWN`).

   ```
   s9s cluster --list --long
   ```

   The node list show information of which node that is failure in the cluster.

   ```
   s9s nodes --list --cluster-id=<clusterId>
   ```

   The `clusterId` value can be found when executing the cluster list command.

2. Check the triggered alarm in the CLI.

   ```
   s9s alarm --list  --long
   ```

## 2.117 Overview

This section describe the Error Report utility in ClusterControl. It is a compressed file (tarball) containing diagnostic information about the cluster (`cmon` logs, database error logs, and the contents of some important CMON database tables). This is very useful for troubleshooting purposes, especially when working with the Severalnines Support team on a support issue. Attach the generated compressed file to your support request.

## 2.118 Generating an Error Report

Follow the following steps to generate an error report:

### ClusterControl GUI

1. To create a new error report for a cluster, go to ***ClusterControl GUI → Clusters → choose a cluster → Reports***, click the **Create Error Report** button. By default, the generated report is stored locally on the ClusterControl host under `$HOME/s9s_tmp`.
2. There will be an option to mask passwords so that if any are contained in the configuration, ClusterControl will mask them. The default is always ON.
3. Download the tarball file through ***Action → Download***.

### ClusterControl CLI

Execute the following command in ClusterControl host as root/sudo user:

```
s9s_error_reporter -i <cluster_id>
```

The `<cluster_id>` is the identifier of the cluster where the error report is to be generated. Where 1 is the cluster ID of the corresponding cluster. For other clusters, just change the cluster ID accordingly. The cluster ID can be retrieved from ***ClusterControl GUI → Clusters → locate the cluster → ID***, using `s9s cluster --list` command or by looking into the CMON configuration file for the respective cluster inside `/etc/cmon.d`.

**Example**

The following command generates an error report for cluster ID 15:

```
s9s_error_reporter -i 15
```

At the end of the execution, it will print out something like this:

```
Executing tar -C /var/tmp/cmon-003522-de42b82f69aba63c -czf '/
↪root/s9s_tmp/error-report-2025-05-23_090757-cluster16.tar.gz'␣
↪error-report-2025-05-23_090757-cluster16
Please attach /
↪root/s9s_tmp/error-report-2025-05-23_090757-cluster16.tar.gz to the␣
↪support issue.
```

**Tip**

If you have no cluster managed by ClusterControl, you might want to specify `0` instead, where 0 has a special meaning in ClusterControl – collect information about ClusterControl installation, configurations, and logs on the local node only.

Please attach the generated tarball to your support ticket. If the tarball exceeds the 20MB attachment limit of our support system, please upload it to a cloud storage service such as Google Drive, Dropbox, or WeTransfer. Then, share the HTTPS download link in your support ticket. Be aware that the error report might contain sensitive and confidential information, so restrict public access and only share the download URL with us.

**Tip**

We also recommend you to take screenshots showing the area of the problem, e.g, the Overview page from the UI allows us to understand the current state of nodes and clusters.

## 2.119 ClusterControl GUI diagnostic

For ClusterControl GUI-related issue, you can use the **Diagnostic** feature available at ***ClusterControl GUI → Settings → Diagnostics*** and toggle ON **UI audit data collection**. This allows user to record and capture the necessary data while reproducing an issue in the user interface.

After replicating the issue, immediately disable the setting to halt audit data collection. Then, click **Download audit data** for your analysis or to share with our Support team for troubleshooting and debugging. This audit data allows our team to replay the exact error users encountered in their specific configurations and environments, eliminating the need for screen sharing.

**Warning**

Warning!

Remember to switch off the setting afterward to stop audit data collection otherwise it will impact the web browser's performance after a period of time.

**Note**

Note Note that we are not collecting any personal data during this process.

## 2.120 Common Issues

This section covers common issues when configuring and dealing with ClusterControl components, with possible troubleshooting steps and solutions. There is also a community forum available with knowledge base sections for public reference.

If you need further assistance, please contact us via our support channel by submitting a support request or post a new thread in our community help forum.

### 2.120.1 ClusterControl Installation

This section covers common issues encountered during ClusterControl installation and the installer script.

#### Failed to start MySQL Server during ClusterControl installation

***Description:***

During installation, the installer script fails to start the MySQL/MariaDB server on ClusterControl host and returns the following lines:

```
=> Starting database. This may take a couple of minutes. Do NOT press any␣
↪key.
mysqld: unrecognized service
=> Failed to start the MySQL Server. ...
Please contact Severalnines support at http://support.severalnines.com if␣
↪you have installation issues that cannot be resolved.
```

***Troubleshooting steps:***

1. Examine the MySQL error log on possible reasons why does MySQL fails to start. Typically, the log file is located under `/var/log/mysqld.log` or `/var/lib/mysql/error.log`.
2. Try starting the MySQL server manually by using `systemctl` or `service` command.

***Solutions:***

If you are running on the older operating system, ClusterControl might not support the distribution and some issues are expected to show up. See *Requirements* for details. Once the MySQL server is up and running, restart the ClusterControl installer script again:

```
$ ./install-cc
```

## ClusterControl Controller (CMON)

This section covers common issues encountered related to ClusterControl Controller (CMON).

### User '{username}' suspended for previous authentication failures.

***Description***:

When trying to execute an s9s command line, the command returns the current s9s user is suspended due to authentication failures. This happens if the key is no longer recognized by the CMON controller when processing a request.

***Example error***:

When running an s9s command, the following error occurs:

```
$ s9s user --list
User 'admin' suspended for previous authentication failures.
```

***Solution***:

1. Remove all the s9s user configuration files (/etc/s9s.conf, ~/.s9s/s9s.conf):

   ```
   $ mv ~/.s9s/s9s.conf ~/.s9s/s9s.conf.bak
   $ mv /etc/s9s.conf /etc/s9s.conf.bak
   ```

2. Create another user for s9s CLI, in this example, we created a user called dba, under group "admins":

   ```
   $ s9s user --create --generate-key --controller=
   ↪"https://localhost:9501" --group=admins dba
   Grant user 'dba' succeeded
   ```

3. Use it to unlock the "original" suspended user (assuming the suspended user is "admin"):

   ```
   $ s9s user --cmon-user=dba --enable admin
   OK.
   ```

By performing the above, it will clear the "disabled" flag of the user so that the user will be able to log in again. The "suspended" flag will also be cleared, the failed login counter set to 0 and the date and time of the last failed login gets deleted, so users who are suspended for failed login attempts will also be able to log in.

### CMON is unable to restart MySQL using the service command

***Description***:

When scheduling a start/restart job, ClusterControl fails to start the node with the error "Command not found".

***Example error***:

```
galera1.domain.com: Starting mysqld failed: Error: Command not found (host:␣
↪galera1.domain.com): service mysql restart
galera1.domain.com: Starting mysqld
```

***Troubleshooting steps***:

1. SSH into the DB node and check the user's environment path variable:

```
$ ssh -tt -i /home/admin/.ssh/id_rsa root@192.168.10.101 "sudo env | grep␣
↪PATH"
PATH=/usr/local/bin:/bin:/usr/bin
```

2. Look at the `PATH` output.

***Solution***:

- Ensure the `/sbin` the path is there. Otherwise, ClusterControl won't be able to automatically locate and run the "service" command.
- If the `/sbin` the path is not listed in the PATH, add it by using the following command:

```
PATH=$PATH:/sbin
export PATH
```

However, the above won't persist if the user logs out from the terminal. To make it persistent, add those lines into `/home/<SSH user>/.bash_profile` or `/home/<SSH user>/.bashrc`

### CMON always tries to recover failed database nodes during my maintenance window.

***Description***:

By default, CMON is configured to perform recovery of failed nodes or clusters. This behavior can be overridden by disabling automatic recovery or enabling maintenance mode for the node.

***Solution***:

1. Enable maintenance mode for selected nodes (recommended). To enable the maintenance window, go to ***Nodes → select the node → Schedule Maintenance Mode → Enable***. You have to specify the reason and duration of the maintenance window. During this period, any alarms and notifications raised for this node will be disabled.
2. Disable automatic recovery. To disable automatic recovery temporarily, turn off automatic recovery via ***ClusterControl GUI → Clusters → choose a cluster → Actions → Disable Node/Cluster recovery***. you can just click on the 'power' icon for node and cluster. Red means automatic recovery is turned off while green indicates recovery is turned on. This behavior will not persistent if CMON is restarted. To make the above change persistent, disable node or cluster auto-recovery by specifying the following line inside the CMON configuration file of the respective cluster. For example, if you want to disable automatic recovery for cluster ID 1, inside `/etc/cmon.d/cmon_1.cnf`, set the following line: `enable_autorecovery=0`

### CMON process dies with "Critical error (mysql error code 1)"

***Description***:

After starting CMON service, it stops and `/var/log/cmon.log` shows the following error:

```
(ERROR) Critical error (mysql error code 1) occurred - shutting down
```

***Troubleshooting steps***:

1. Run the following command on the ClusterControl host to check if it has the ability to connect to the DB host with current credentials:

```
$ mysql -ucmon -p -h<database node IP> -P<MySQL port> -e 'SHOW STATUS'
```

2. Check GRANT for the cmon user on each database host:

```
mysql> SHOW GRANTS FOR 'cmon'@'<ClusterControl IP address>';
```

***Solution***:

It is not recommended to mix public IP addresses and internal IP addresses. For the GRANT, try to use the IP address that your database nodes use to communicate with each other. If the `SHOW STATUS` statement returns `ERROR 1130 (HY000): Host '<ClusterControl IP address>' is not allowed to connect to this`, the database host is missing the cmon user grant. Run the following command to reset the cmon user privileges:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'cmon'@'<ClusterControl IP>'␣
↪IDENTIFIED BY '<cmon password>' WITH GRANT OPTION;
mysql> FLUSH PRIVILEGES;
```

Where, is ClusterControl IP address and is `mysql_password` value inside the CMON configuration file.

### Job fails with 'host is already in another cluster' error

***Description***:

When deploying a new node, or adding a node into an existing cluster managed by ClusterControl, the deployment fails with the following error:

```
Host 1.2.3.4:nnnn is already in another cluster.
```

***Solution***:

A host can only exist in one cluster at a time. Check if you have an `/etc/cmon.d/cmon_X.cnf` file (where X is an integer) that contains the hostname and remove the file if the corresponding cluster does no exist in the UI (be careful to not remove the wrong `cmon_X.cnf` file):

```
$ rm /etc/cmon.d/cmon_X.cnf
```

Otherwise, delete the host from `server_node`, `mysql_server`, and `hosts` tables:

```
mysql> DELETE FROM cmon.server_node WHERE hostname='1.2.3.4';
mysql> DELETE FROM cmon.mysql_server WHERE hostname='1.2.3.4';
mysql> DELETE FROM cmon.hosts WHERE hostname='1.2.3.4';
```

Restart CMON to load the new changes:

```
$ systemctl restart cmon
```

You may have to execute the deletion several times for each hostname/IP of the cluster you are trying to add.

### Failed to deploy Prometheus service

***Description***:

The deployment of prometheus service failed due to the `prometheus.service` does not exist as follows:

```
Failed to enable prometheus.service: Command exited with return code 1 on␣
↪host xxxx. Command: systemctl --no-pask-password enable prometheus.
↪service stdErr: Failed to enable unit: Unit file prometheus.service does␣
↪not exist. stderr: Failed to enable unit: Unit file prometheus.service␣
↪does not exist
```

***Troubleshooting steps***:

Run the following command to verify the following command in the ClusterControl host to check for the detail why the prometheus won't start:

```
$ journalctl -t setroubleshoot
```

If the result is related to the `[20764]: SELinux is preventing systemd from read access on the file prometheus.service.`, check the status of SELinux.

```
$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      33
```

***Solution***:

In this case, you need to update the parameter `SELINUX` in SELinux configuration for CentOS/Redhat based. The configuration can be found at `/etc/selinux/config`. Change the value become `PERMISSIVE` or `DISABLED`. While for the Debian/Ubuntu based, you need to stop the AppArmor service.

### Unable to authenticate to LDAP server

***Description***:

Unable to log in to ClusterControl using the LDAP user after LDAP Settings is configured.

***Troubleshooting steps***:

1. Make sure that you have mapped ClusterControl's Team with the respective LDAP Group Name under ***ClusterControl → User Management → LDAP → Map LDAP Group***.

2. Verify if the configured ***LDAP Settings*** are still correct. Go to ***ClusterControl → User Management → LDAP → LDAP Settings*** and make sure hit the **Enable LDAP authentication** is toggled ON. When toggling ON, ClusterControl will perform a simple call to the configured LDAP server to verify the connection info. You will see a pop-up error if the connection info is incorrect. Note that Windows Active directory DNs are case-sensitive.
3. For failure LDAP events, ClusterControl will capture the error log under `/var/log/cmon.log`. Examine this log to look for any clues why LDAP authentication fails.
4. On ClusterControl node, try to list out the directory branch by using the admin DN's and password using `ldapsearch` the command (OpenLDAP-clients package is required):

```
$ ldapsearch -H ldaps://ad.company.com -x -b
↪'CN=DBA,OU=Groups,DC=ad,DC=company,DC=com' -D
↪'CN=Administrator,OU=Users,DC=ad,DC=company,DC=com' -W
```

***Solutions***:

ClusterControl supports Active Directory, FreeIPA, and OpenLDAP authentication, see *LDAP Integration* for details. If the above troubleshooting steps do not help you solve the issue, please contact us via our support channel for further assistance.

## 2.121 Known Limitations

ClusterControl is a complex and sophisticated product. It consists of several components and it is recommended to run it on a dedicated host. Here are some noteworthy and known limitations that we have gathered so far:

### 2.121.1 General

- ClusterControl can not reside on the same host as the database nodes that you want to deploy or import. ClusterControl must be running on an independent node away from your database clusters/servers.
- For good long-term support, do not co-locate ClusterControl with other applications outside of ClusterControl's control. For example, do not co-locate ClusterControl with a DNS server, an email server, or on a hosting server controlled by a control panel manager. A dedicated host is recommended.
- Not all functionalities are available on both user interfaces, *ClusterControl GUI* and *ClusterControl CLI*. The command-line client is focused on automation and adoption in management, deployment and scaling operations while the web UI client, focuses more on structural visualization with a guided approach.
- In terms of Linux security, ClusterControl does not manage firewalls (iptables, ufw, firewalld) or any Linux kernel security modules like SELinux or AppArmor. ClusterControl only offers a switch for this feature to be turned on or off while deployment to reduce the risk of deployment errors. The SysAdmin is responsible for these aspects.

### 2.121.2 SSH

- ClusterControl requires the SSH user to be a sudo user without restrictions or simply a root user.
- ClusterControl only supports OpenSSH. Other SSH implementations like Dropbear and TinySSH are not supported.

### 2.121.3 MySQL/MariaDB

- ClusterControl does not support managing multiple MySQL and MariaDB instances running on a single host. This includes multiple MySQL/MariaDB instances running on Docker container or running MySQL/MariaDB instances via mysqld_multi, MySQL Sandbox or DBdployer.
- ClusterControl only supports GTID-based replication (both MySQL's and MariaDB's) for automatic failover and recovery. If you are running on a position-based replication, ClusterControl is still able to manage and monitor it without supporting the automatic failover.
- When importing an existing MySQL/MariaDB server/cluster into ClusterControl, some non-destructive configurations will be appended:
  - CMON database user.
  - Backup database user credentials.
  - Some systemd configuration for MySQL/MariaDB will be overridden e.g, Galera Cluster to allow long SST.
  - Read-only flag (MySQL/MariaDB replication).
- When automatic recovery is enabled, ClusterControl always tries to be as conservative as possible when recovering a cluster or a database node. For example, it will always start/restart node one node at a time (not multiple at once), it will always force one master at a time (MySQL replication) or in case of network flapping, ClusterControl will attempt to recover the cluster once and if it keeps failing instantaneously, it will back off on the second attempt and raise an alarm. ClusterControl provides many hooks for you to extend the recovery and failover behavior, as shown on the *Configuration Options*.

### 2.121.4 Containerization

- Although Severalnines offers ClusterCluster as a Docker image, it is not intended for production usage. ClusterControl product direction is never intended to run on a container environment due to its internal logic and system design. We are maintaining the Docker image on a best-effort basis, and it is not part of the product development projection and pipeline.
- Supporting on database instances running on the Docker container is fairly limited. ClusterControl requires an SSH server to communicate and control the instance, which commonly does not come with the database's Docker image (which defeats the purpose of microservices). You either have to install an SSH server on the container or use a special image that comes with SSH like centos-ssh.

### 2.121.5 Monitoring

- When agent-based monitoring is activated, ClusterControl will control the Prometheus and all exporters' processes to avoid false positives when observing the node and cluster state. Hence, ClusterControl does not support importing an existing Prometheus configured and deployed outside of ClusterControl.

CHAPTER

THREE

# ADMINISTRATION GUIDE

## 3.1 Upgrading ClusterControl

There are several ways to upgrade ClusterControl to the latest version. However, we recommend users to perform an *online upgrade* where the instructions are mostly up-to-date. For details on the latest release, see Release Notes.

**Warning**

Attention

If you are upgrading from ClusterControl v1 (1.9.8 and older), please see *Upgrading from ClusterControl 1.x to 2.x*. If you are running on legacy operating systems with GLIBC < 2.27 (RHEL 7, CentOS 7, Debian 9), you can only upgrade to version ClusterControl 2.1.0. See *Upgrading to v2.1.0 for Legacy Operating Systems*.

**Warning**

Attention

If you are upgrading from ClusterControl v2 (2.0.0 and later), proceed to *Online Upgrade* or *Offline Upgrade*.

### 3.1.1 Online upgrade

This is the recommended way to upgrade ClusterControl. The following upgrade procedures require an internet connection on the ClusterControl node.

**Warning**

Attention

If you are upgrading from ClusterControl v1 (1.9.8 and older), please see *Upgrading from ClusterControl 1.x to 2.x*. If you are running on legacy operating systems with GLIBC < 2.27 (RHEL 7, CentOS 7, Debian 9), you can only upgrade to version ClusterControl 2.1.0. See *Upgrading to v2.1.0 for Legacy Operating Systems*.

### Red Hat/CentOS/Rocky Linux/AlmaLinux

1. Starting with ClusterControl 2.3.2, the web application is now included in the `clustercontrol-mcc` package, which supersedes the `clustercontrol2` package. This new setup eliminates the dependency on the Apache web server. Instead, the web application is served by the `ccmgr` process, part of the `clustercontrol-proxy` package. Consequently, to upgrade, the existing Apache server must be uninstalled using the following commands:

```
systemctl stop httpd
dnf remove httpd
```

2. Clear the repository cache so it will retrieve the latest repository list and perform the upgrade:

```
dnf clean all
dnf install clustercontrol-mcc \
    clustercontrol-controller \
    clustercontrol-ssh \
    clustercontrol-notifications \
    clustercontrol-cloud \
    clustercontrol-clud \
    clustercontrol-proxy \
    clustercontrol-kuber-proxy \
    s9s-tools
```

3. Initialize the new ClusterControl web application to be started on port 443:

```
ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
```

**Warning**

Attention

```
If you see the following output, it means the web application is␣
↪already initialized and you may proceed to the next step:

```bash
$ ccmgradm init --local-cmon -p 443 -f /
↪var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller already exists with this URL.
```
```

**Example**

```
$ ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
```

```
:::{tip}

To use a custom SSL certificate, you must update the `tls_key` and␣
↪`tls_cert` parameters in the `/usr/share/ccmgr/ccmgr.yaml` configuration␣
↪file. Ensure the certificate and key files are owned by the user `s9s_cc`␣
↪and the group `severalnines`.
:::
```

4. Restart all ClusterControl services:

```
systemctl daemon-reload
systemctl restart cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
```

The upgrade is now complete. Verify the new version from the GUI's footer or by using the command `cmon -v`. You should re-login if your ClusterControl GUI session is active.

### Debian/Ubuntu

1. Starting with ClusterControl 2.3.2, the web application is now included in the `clustercontrol-mcc` package, which supersedes the `clustercontrol2` package. This new setup eliminates the dependency on the Apache web server. Instead, the web application is served by the `ccmgr` process, part of the `clustercontrol-proxy` package. Consequently, to upgrade, the existing Apache server and the old `clustercontrol2` package must be uninstalled using the following commands:

```
apt purge apache2 clustercontrol2
```

2. Update the repository list and perform the upgrade by installing the latest version of the following packages:

```
sudo apt update
sudo apt install clustercontrol-mcc \
    clustercontrol-controller \
    clustercontrol-ssh \
    clustercontrol-notifications \
    clustercontrol-cloud \
    clustercontrol-clud \
    clustercontrol-proxy \
    clustercontrol-kuber-proxy \
    s9s-tools
```

3. Initialize the new ClusterControl web application to be started on port 443:

```
ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
```

**Warning**

Attention

```
If you see the following output, it means the web application is␣
↪already initialized and you may proceed to the next step:

```bash
$ ccmgradm init --local-cmon -p 443 -f /
↪var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller already exists with this URL.
```
```

**Example**

```
$ ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
```

```
:::{tip}

To use a custom SSL certificate, you must update the `tls_key` and␣
↪`tls_cert` parameters in the `/usr/share/ccmgr/ccmgr.yaml` configuration␣
↪file. Ensure the certificate and key files are owned by the user `s9s_cc`␣
↪and the group `severalnines`.
:::
```

4. Restart all ClusterControl services:

```
systemctl daemon-reload
systemctl restart cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
```

The upgrade is now complete. Verify the new version from the GUI's footer or by using the command `cmon -v`. You should re-login if your ClusterControl GUI session is active.

### 3.1.2 Offline upgrade

The following upgrade procedures can be performed without an internet connection on the ClusterControl node.

**Warning**

Attention

If you are upgrading from ClusterControl v1 (1.9.8 and older), please see *Upgrading from ClusterControl 1.x to 2.x*. If you are running on legacy operating systems with GLIBC < 2.27 (RHEL 7, CentOS 7,

Debian 9), you can only upgrade to version ClusterControl 2.1.0. See *Upgrading to v2.1.0 for Legacy Operating Systems*.

### Red Hat/CentOS/Rocky Linux/AlmaLinux

1. Download the latest version of ClusterControl-related RPM packages from Severalnines download site and the Severalnines Repository. There are some packages you need to download as explained below:
   - `clustercontrol-mcc` - ClusterControl GUI
   - `clustercontrol-controller` - ClusterControl Controller (CMON)
   - `clustercontrol-notifications` - ClusterControl event module
   - `clustercontrol-ssh` - ClusterControl web-ssh module
   - `clustercontrol-cloud` - ClusterControl cloud module
   - `clustercontrol-clud` - ClusterControl cloud's file manager module
   - `clustercontrol-proxy` - ClusterControl proxy manager and web server
   - `clustercontrol-kuber-proxy` - ClusterControl Kubernetes proxy
   - `s9s-tools` - ClusterControl CLI (s9s) - Download from here.

   **Note**

   Note If you are upgrading from ClusterControl 2.3.3, skip step 2.

2. Starting with ClusterControl 2.3.2, the web application is now included in the `clustercontrol-mcc` package, which supersedes the `clustercontrol2` package. This new setup eliminates the dependency on the Apache web server. Instead, the web application is served by the `ccmgr` process, part of the `clustercontrol-proxy` package. Consequently, to upgrade, the existing Apache server must be uninstalled using the following commands:

```
systemctl stop httpd
dnf remove httpd
```

3. Install using `dnf localinstall` to satisfy all dependencies:

```
dnf localinstall clustercontrol*
dnf localinstall s9s-tools
```

4. Initialize the new ClusterControl web application to be started on port 443:

```
ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
```

**Warning**

Attention

```
If you see the following output, it means the web application is␣
↪already initialized and you may proceed to the next step:

```bash
$ ccmgradm init --local-cmon -p 443 -f /
↪var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller already exists with this URL.
```
```

**Example**

```
$ ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
```

```
:::{tip}

To use a custom SSL certificate, you must update the `tls_key` and␣
↪`tls_cert` parameters in the `/usr/share/ccmgr/ccmgr.yaml` configuration␣
↪file. Ensure the certificate and key files are owned by the user `s9s_cc`␣
↪and the group `severalnines`.
:::
```

5. Restart all ClusterControl services:

```
systemctl daemon-reload
systemctl restart cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
```

The upgrade is now complete. Verify the new version from the GUI's footer or by using the command `cmon -v`. You should re-login if your ClusterControl UI session is active.

### Debian/Ubuntu

1. Download the latest version of ClusterControl-related DEB packages from Severalnines download site and Severalnines Repository. There are some packages you need to download as explained below:
   - `clustercontrol-mcc` - ClusterControl GUI
   - `clustercontrol-controller` - ClusterControl Controller (CMON)
   - `clustercontrol-notifications` - ClusterControl event module
   - `clustercontrol-ssh` - ClusterControl web-ssh module

- `clustercontrol-cloud` - ClusterControl cloud module
- `clustercontrol-clud` - ClusterControl cloud's file manager module
- `clustercontrol-proxy` - ClusterControl proxy manager and web server
- `clustercontrol-kuber-proxy` - ClusterControl Kubernetes proxy
- `s9s-tools` - ClusterControl CLI (s9s) - Download from here.
- `libs9s0` - ClusterControl CLI (s9s) library - Download from here.

**Note**

Note If you are upgrading from ClusterControl 2.3.3, skip step 2.

2. Starting with ClusterControl 2.3.2, the web application is now included in the `clustercontrol-mcc` package, which supersedes the `clustercontrol2` package. This new setup eliminates the dependency on the Apache web server. Instead, the web application is served by the `ccmgr` process, part of the `clustercontrol-proxy` package. Consequently, to upgrade, the existing Apache server and the old `clustercontrol2` package must be uninstalled using the following commands:

```
apt purge apache2 clustercontrol2
```

3. Upload all the packages to the ClusterControl host and install them using `dpkg` command:

```
sudo dpkg -i clustercontrol*.deb
sudo dpkg -i libs9s0*.deb
sudo dpkg -i s9s*.deb
```

4. Initialize the new ClusterControl web application to be started on port 443:

```
ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
```

**Warning**

Attention

```
If you see the following output, it means the web application is␣
↪already initialized and you may proceed to the next step:

```bash
$ ccmgradm init --local-cmon -p 443 -f /
↪var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller already exists with this URL.
```
```

**Example**

```
$ ccmgradm init --local-cmon -p 443 -f /var/www/html/clustercontrol-mcc
ClusterControl Manager - admin CLI v2.2
Controller 127.0.0.1:9501 registered successfully
Changing frontend_path from /app to /var/www/html/clustercontrol-mcc
File /var/www/html/clustercontrol-mcc/config.js updated successfully
Configuration /usr/share/ccmgr/ccmgr.yaml updated successfully
Please restart 'cmon-proxy' service to apply changes
```

```
:::{tip}

To use a custom SSL certificate, you must update the `tls_key` and␣
↪`tls_cert` parameters in the `/usr/share/ccmgr/ccmgr.yaml` configuration␣
↪file. Ensure the certificate and key files are owned by the user `s9s_cc`␣
↪and the group `severalnines`.
:::
```

5. Restart the ClusterControl services:

```
systemctl daemon-reload
systemctl restart cmon cmon-ssh cmon-events cmon-cloud cmon-proxy␣
↪kuber-proxy
```

The upgrade is now complete. Verify the new version from the GUI's footer or by using the command `cmon -v`. You should re-login if your ClusterControl GUI session is active.

### 3.1.3 Upgrading from ClusterControl v1 to v2

ClusterControl v2 features significant changes and improvements across its frontend web UI, user management, monitoring method and backend controller. Consequently, a proper upgrade and migration of several components are essential for its smooth operation.

The easiest way to upgrade from v1 to v2 is to install a new ClusterControl instance with the latest version (see *Quickstart*), and then import all clusters from the old ClusterControl (v1), to the new ClusterControl (v2) using the **Import existing cluster** feature. *Import Database Cluster* for details. After imported, manually move the settings and configurations. For a clear step-by-step guide to perform this method, see this tutorial, *Migrating Existing Clusters from ClusterControl v1 to ClusterControl v2*. However, the drawback is you will lose the monitoring data of the past and settings have to be migrated manually.

If this is not an option, kindly follow the upgrade instructions as explained in this section.

#### Requirements

To upgrade from legacy ClusterControl 1.x (1.9.8 and older) to the latest 2.x version, the following conditions must be met:

### Operating System Compatibility

- ClusterControl 2.2.0 and later requires GLIBC 2.27 or later, which is available on RHEL/Rocky Linux/AlmaLinux ≥ 8, Ubuntu ≥ 20.04, and Debian ≥ 10. This is due to significant changes related to OpenSSL v3 and FIPS 140-2 compliance. See *Operating System*.
- If your system uses an older operating system with GLIBC < 2.27 (RHEL 7, CentOS 7, Debian 9), you can only upgrade to ClusterControl version 2.1.0. See *Upgrading to v2.1.0 for Legacy Operating Systems* for instructions.

### User Management

- ClusterControl v2 introduces a new user management system (User Management v2) where login is based on a username, such as `ccadmin`. If this is the way you log into ClusterControl, you can directly proceed to *Online upgrade* or *Offline upgrade*.
- If you currently log into the ClusterControl GUI v1 with an email address, you are using the legacy user management system. Before upgrading to ClusterControl v2, you must first migrate these existing users to the new User Management v2. Refer to *Migrating ClusterControl users to v2* for instructions. If you log into ClusterControl GUI v1 using a username, skip the user migration step.

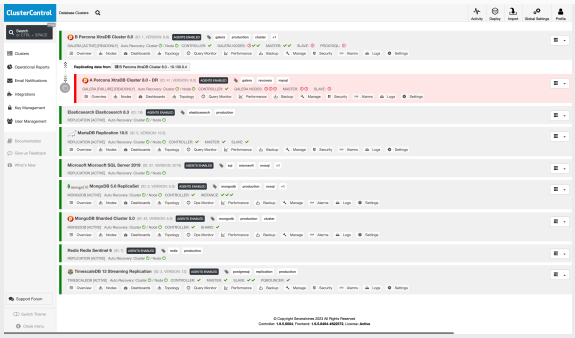
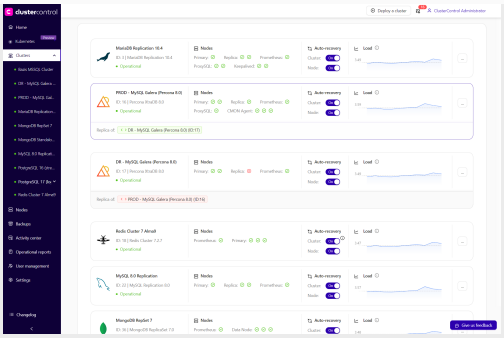
### LDAP Integration

- LDAP in ClusterControl v2 is only compatible with the new User Management v2.
- If you have existing group mappings configured in LDAP v1, you will need to recreate these mappings in LDAP v2 after completing the user migration to v2. Refer to *Migrating LDAP users to v2* for details.

### Monitoring

- ClusterControl v2 defaults to agent-based monitoring using Prometheus and exporters for collecting time-series data (CPU, RAM, disk, etc.).
- After migrating to User Management v2, you can enable agent-based monitoring for each cluster via ***ClusterControl GUI v1 or v2 → choose a cluster → Dashboards → Enable agent-based monitoring***. ClusterControl will then deploy the necessary exporters on each database node and configure the Prometheus targets accordingly. If this is already enabled in your ClusterControl v1, you can directly upgrade the packages to ClusterCotnrol v2.x by following *Online upgrade* or *Offline upgrade*.

To summarize, here are the main difference between ClusterControl v1 and v2:

| Aspect | ClusterControl v1 | ClusterControl v2 |
|---|---|---|
| User management | Users log in using email address, e.g. `admin@example.com` (User management v1) Users log in using username, e.g `ccadmin` (User management v2) | Users log in using username, e.g `ccadmin` (User management v2) |
| Monitoring | Agentless monitoring via SSH sampling Agent-based monitoring using Prometheus and exporters | Agent-based monitoring using Prometheus and exporters |
| LDA | LDAP is handled by ClusterControl GUI | LDAP is handled by ClusterControl controller (cmon) |
| GUI URL | `https://<ClusterControl_host>/clustercontrol` | `https://<ClusterControl_host>` |
| Interface |  |  |

## Migrating monitoring to agent-based monitoring

Starting from ClusterControl v2, time-series monitoring will be performed by Prometheus, thus you need to go to ***ClusterControl GUI v1 → choose a cluster → Dashboards → Enable Agent-Based Monitoring*** to enable Prometheus agent-based monitoring for every cluster. ClusterControl will deploy Prometheus (default to ClusterControl host) with the corresponding exporters on all managed nodes. If you have many clusters and want to simplify this process, you may use ClusterControl CLI with `--deploy-agents` flag.

**Example**

The following commands enable agent-based monitoring for 7 clusters, one cluster at a time:

```
s9s cluster --cluster-id=3 --deploy-agents --log
s9s cluster --cluster-id=16 --deploy-agents --log
s9s cluster --cluster-id=26 --deploy-agents --log
s9s cluster --cluster-id=46 --deploy-agents --log
s9s cluster --cluster-id=56 --deploy-agents --log
s9s cluster --cluster-id=57 --deploy-agents --log
s9s cluster --cluster-id=58 --deploy-agents --log
```

To verify if every cluster has agent-based monitoring enabled:

### ClusterControl GUI

Make sure the monitoring dashboards at ***ClusterControl GUI v1 → choose a cluster → Dashboards*** are populated correctly for every cluster.

### ClusterControl CLI

Check if an instance with a `P` (means Prometheus) role is in the list of every cluster:

```
s9s nodes --list --long | grep ^P
```

**Example**

The following output shows that every cluster has its own Prometheus instance (pay attention at the left-most column):

```
$ s9s nodes --list --long | grep ^P
Po-- 2.29.2         3 MariaDB Replication 10.4          192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        16 PROD - MySQL Galera (Percona 8.0) 192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        17 DR - MySQL Galera (Percona 8.0)   192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        18 Redis Cluster 7 Alma9             192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        22 MySQL 8.0 Replication             192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        36 MongoDB RepSet 7                  192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        37 MongoDB Standalone 7 percona      192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        38 PostgreSQL 16 (streaming)         192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        40 PostgreSQL 17 (logical)_1         192.168.61.210 ␣
 ↪9090 Pro…
Po-- 2.29.2        41 PostgreSQL 17 (logical)_2         192.168.61.210 ␣
 ↪9090 Pro…
```

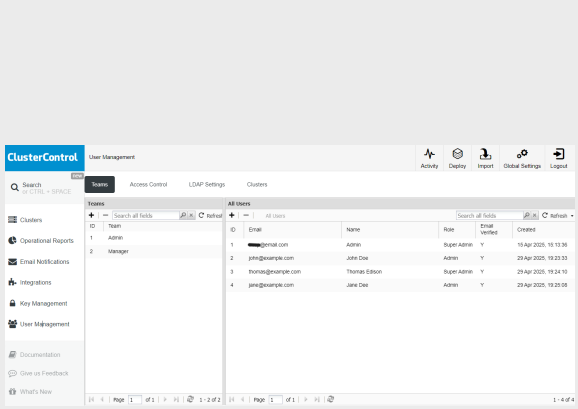
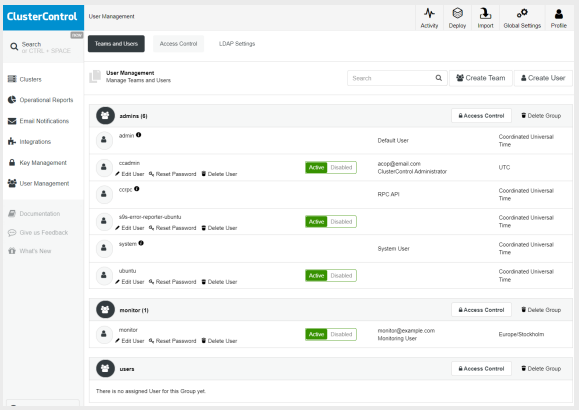
### Migrating ClusterControl users to User Management v2

If you login to ClusterControl GUI v1 using an email address (e.g, `admin@example.com`), it means that you are still using the legacy ClusterControl user management handled by ClusterControl GUI.

The new user management system (also known as **User Management v2**), is handled by ClusterControl Controller (cmon), allowing ClusterControl users to communicate directly with the `cmon` backend service through various interfaces like command-line interface (*ClusterControl CLI*), application programming interface (*ClusterControl RPC API v2*) and *Terraform Provider for ClusterControl*.

If you login to ClusterControl GUI v1 using a username (e.g, `ccadmin`), you can bypass the steps detailed in this section as **User Management v2** is already enabled.

Here are the differences in the user interface between legacy user management and user management v2 in ClusterControl GUI v1 (click to enlarge):

| Legacy user management | User management v2 |
| --- | --- |
|  |  |

Unfortunately, the user and group migration must be performed manually, meaning all existing ClusterControl users and groups must be re-created in the User Management v2. If you have a lot of users, we recommend you to use the ClusterControl CLI method. Here are the steps to migrate ClusterControl users to the new user management v2 using ClusterControl CLI.

1. Create a new ClusterControl super-admin user. In this example, we called it “ccadmin”. This user must belong in the “admins” group:

```
s9s user \
    --create \
    --group=admins \
    --generate-key \
    --new-password=SuPeRs3cr3tP455 \
    --email-address=ccadmin@example.com \
    --first-name=ClusterControl \
    --last-name=Admninistrator \
    --batch \
    ccadmin
```

> **Tip**
>
> Instead of `ccadmin`, you may use your own username. Username `admin` is not available and reserved for ClusterControl internal usage.

2. Create all ClusterControl users and its respective group. In the following example, we created a user called John Doe, with login name “john” and also create a group called “dba”, which John Doe will be part of it:

```
s9s user \
    --create \
    --group=dba \
    --create-group \
    --generate-key \
    --new-password=s3cr3tP455 \
    --email-address=johndoe@example.com \
```

(continues on next page)

(continued from previous page)

```
    --first-name=John \
    --last-name=Doe \
    --batch \
    john
```

Repeat this command for every user. Omit `--create-group` if the group already exists and change the `--new-password` value accordingly.

3. If you want to create a new group, you must pass a username as one of the member of the group. In this example, we create a group called "sysadmin" and also create a default user for this group called "sysadmin":

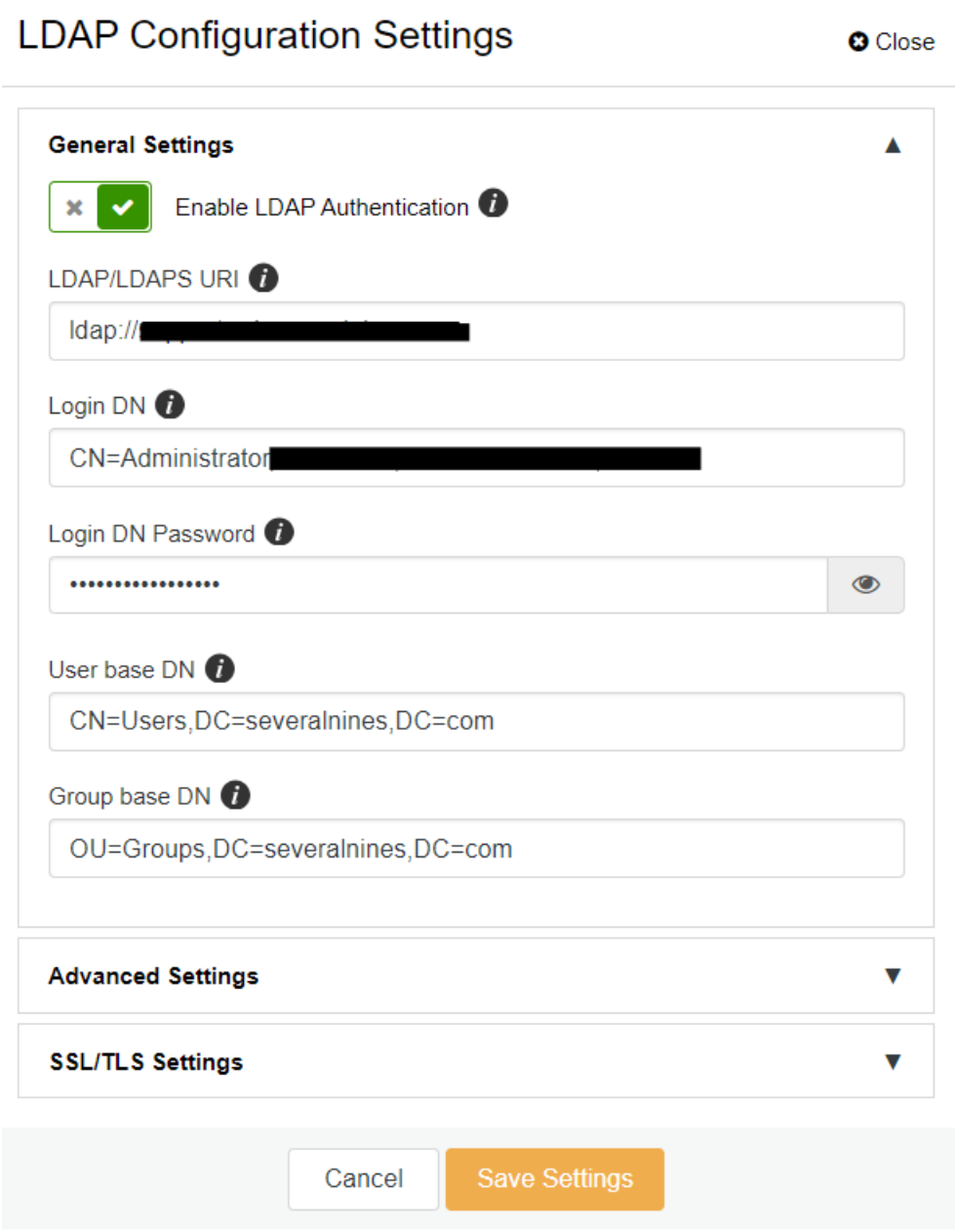
```
s9s user \
    --create \
    --create-group \
    --group=sysadmin \
    sysadmin
```

4. To change user "john" from group "dba" to another group called "sysadmin":

```
s9s user \
    --set-group \
    --group=sysadmin \
    john
```

### Migrating LDAP users to v2

Log into ClusterControl GUI using a username from the admins group. This will activate User Management v2 and allow for the LDAP v2 configuration at ***ClusterControl GUI v1 or v2 → User Management → LDAP → LDAP Settings***. Make sure you see the LDAP v2 similar to below:




To migrate LDAP users to v2, re-configure the **LDAP Settings** accordingly, by using the same credentials and configurations as in the legacy user management together with its group mapping. See *LDAP* section for details.

### 3.1.4 Upgrading to v2.1.0 for Legacy Operating Systems

Starting from ClusterControl v2.2.0 (Sept 2024), ClusterControl requires GLIBC 2.27 and later due to major changes in supporting OpenSSL v3 and FIPS 140-2 requirements. This is only offered in the following operating systems:

- Red Hat Enterprise Linux/Rocky Linux/AlmaLinux ≥ 8
- Ubuntu ≥ 20.04
- Debian ≥ 10

We highly recommend upgrading the operating systems to the supported version shown in the *Operating System* section for long-term support. One method is to prepare a new ClusterControl server on the sup-

ported operating system, *install the latest ClusterControl* (for a new installation, v2 and above, you will only get the new GUI v2), and then import the database clusters into the new ClusterControl v2.

If you want to upgrade ClusterControl running on the legacy operating systems (not recommended, but somehow relevant), the latest ClusterControl packages that can be installed are:

- clustercontrol-controller-2.1.0-10601
- clustercontrol2-2.2.5-1655
- clustercontrol-notifications-2.0.0-344
- clustercontrol-cloud-2.0.0-408
- clustercontrol-clud-2.0.0-408
- clustercontrol-ssh-2.0.0-166

**Info**

For offline upgrade, kindly pre-download the above packages from Severalnines Download Site.

The following is an example of ClusterControl upgrade on a RHEL-based operating system (CentOS 7):

1. Perform the upgrade to the selected version:

```
yum install clustercontrol-controller-2.1.0-10601 \
    clustercontrol2-2.2.5-1655 \
    clustercontrol-notifications-2.0.0-344 \
    clustercontrol-cloud-2.0.0-408 \
    clustercontrol-clud-2.0.0-408 \
    clustercontrol-ssh-2.0.0-166 \
    s9s-tools
```

2. Restart the ClusterControl services:

```
systemctl restart cmon cmon-ssh cmon-events cmon-cloud
```

The upgrade is now complete.

**Note**

Note If you are upgrading from ClusterControl 1.9.8 and older, you will have two ClusterControl GUI (v1 and v2) and both can be accessible as below:

```
* ClusterControl GUI v1 (feature-freeze, stays at v1.9.8, provided by␣
↪package `clustercontrol`):␣
↪`https://<ClusterControl_host>/clustercontrol`.
* ClusterControl GUI v2 (latest, version 2.x, provided by package␣
↪`clustercontrol2`): `https://<ClusterControl_host>:9443/`.

Starting from ClusterControl v2, time-series monitoring will be performed␣
↪by Prometheus, thus you need to go to _**ClusterControl GUI → choose a␣
↪cluster → Dashboards → Enable Agent-Based Monitoring**_ for every␣
↪cluster. ClusterControl will perform Prometheus deployment with the␣
↪corresponding exporters on all managed nodes.
```

### 3.1.5 Activating web-based SSH for ClusterControl GUI v2

**Info**

Follow the instructions in this section if ClusterControl web-based SSH in ClusterControl GUI v2 is not activated (or not working after an upgrade).

Starting from ClusterControl 1.9.8, ClusterControl GUI v2 is now able to perform a web-based SSH using a web socket. ClusterControl will use the operating system user as configured during the cluster deployment/import and pop up a new browser window to access the SSH terminal via ClusterControl.

**Note**

Note The ClusterControl GUI user must have "Manage" privilege to the cluster to use this feature.

To activate the web-based SSH on ClusterControl GUI v2, a modification is required to be performed to the Apache configuration file, to ensure the feature is secure and ClusterControl can exchange HTTP token authentication properly.

1. Ensure you have upgraded to the latest version of the ClusterControl GUI v2 package. The package name and version should be at least clustercontrol2.x86_64 2.1.0-1203.
2. Edit `/etc/httpd/conf.d/cc-frontend.conf` (RedHat-based) or `/etc/apache2/conf.d/sites-available/cc-frontend.conf` (Debian-based) and make sure the following lines exist inside the `<VirtualHost *:9443>` section.

```
        # Proxy settings
        SSLProxyEngine on
        SSLProxyVerify none
        SSLProxyCheckPeerCN off
        SSLProxyCheckPeerExpire off
        SSLProxyCheckPeerName off
        SSLProxyCACertificateFile /var/lib/cmon/ca/cmon/rpc_tls.crt

        <LocationMatch /cc-license>
            ProxyPass https://severalnines.com/service/lic.php
            ProxyPassReverse https://severalnines.com/service/lic.php
        </LocationMatch>

        <LocationMatch /api/v2/>
            ProxyPass https://127.0.0.1:9501/v2/
            ProxyPassReverse https://127.0.0.1:9501/v2/
            Header edit Set-Cookie ^(.*)$ "$1; Path=/"
        </LocationMatch>

        <LocationMatch /api/events-test/>
            ProxyPass http://127.0.0.1:9510/test/
            ProxyPassReverse http://127.0.0.1:9510/test/
```

(continues on next page)

(continued from previous page)

```
        </LocationMatch>


        <Location /cmon-ssh/cmon/ws/>
            RewriteEngine On
            RewriteCond %{REQUEST_URI} ^/cmon-ssh/cmon/ws/(.*)$
            RewriteRule ^(.*)$ ws://127.0.0.1:9511/cmon/ws/%1 [P,L]
        </Location>

        <LocationMatch /cmon-ssh/>
            ProxyPass http://127.0.0.1:9511/
            ProxyPassReverse http://127.0.0.1:9511/
        </LocationMatch>
```

This is an example after we have updated the configuration file on a Rocky Linux 8 system, `/etc/httpd/conf.d/cc-frontend.conf`:

```
<VirtualHost *:9443>
        ServerName 157.230.37.193
        ServerAlias *.severalnines.local

        DocumentRoot /var/www/html/clustercontrol2
        #ErrorLog /var/log/httpd/cc-frontend-error.log
        #CustomLog /var/log/httpd/cc-frontend-access.log combined
        #ErrorLog ${APACHE_LOG_DIR}/cc-frontend-error.log
        #CustomLog ${APACHE_LOG_DIR}/cc-frontend-access.log combined

        # HTTP Strict Transport Security (mod_headers is required)␣
↪(63072000 seconds)
        Header always set Strict-Transport-Security "max-age=63072000"

        SSLEngine on
        SSLCertificateFile /etc/ssl/certs/s9server.crt
        SSLCertificateKeyFile /etc/ssl/private/s9server.key

        <Directory />
                Options +FollowSymLinks
                AllowOverride All
                Require all granted
        </Directory>

        <Directory /var/www/html/clustercontrol2>
                Options +Indexes +Includes +FollowSymLinks -MultiViews
                AllowOverride All

                RewriteEngine On
                # If an existing asset or directory is requested go to␣
↪it as it is
                RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
                RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
```

(continues on next page)

(continued from previous page)

```
                RewriteRule ^ - [L]
                # If the requested resource doesn't exist, use␣
↪index.html
                RewriteRule ^ /index.html
        </Directory>

        # Proxy settings
        SSLProxyEngine on
        SSLProxyVerify none
        SSLProxyCheckPeerCN off
        SSLProxyCheckPeerExpire off
        SSLProxyCheckPeerName off
        SSLProxyCACertificateFile /var/lib/cmon/ca/cmon/rpc_tls.crt

        <LocationMatch /cc-license>
            ProxyPass https://severalnines.com/service/lic.php
            ProxyPassReverse https://severalnines.com/service/lic.php
        </LocationMatch>

        <LocationMatch /api/v2/>
            ProxyPass https://127.0.0.1:9501/v2/
            ProxyPassReverse https://127.0.0.1:9501/v2/
            Header edit Set-Cookie ^(.*)$ "$1; Path=/"
        </LocationMatch>

        <LocationMatch /api/events-test/>
            ProxyPass http://127.0.0.1:9510/test/
            ProxyPassReverse http://127.0.0.1:9510/test/
        </LocationMatch>


        <Location /cmon-ssh/cmon/ws/>
            RewriteEngine On
            RewriteCond %{REQUEST_URI} ^/cmon-ssh/cmon/ws/(.*)$
            RewriteRule ^(.*)$ ws://127.0.0.1:9511/cmon/ws/%1 [P,L]
        </Location>

        <LocationMatch /cmon-ssh/>
            ProxyPass http://127.0.0.1:9511/
            ProxyPassReverse http://127.0.0.1:9511/
        </LocationMatch>



</VirtualHost>
# intermediate configuration
SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite          ␣
↪ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-S
```

(continues on next page)

(continued from previous page)

```
SSLHonorCipherOrder     off
SSLSessionTickets       off

# SSLUseStapling On
# SSLStaplingCache "shmcb:logs/ssl_stapling(32768)"
```

3. Restart the Apache service:

#### For RHEL-based server

```
systemctl restart httpd
```

#### For Debian-based server

```
systemctl restart apache2
```

4. Finally, you must log out from the ClusterControl GUI v2 and re-login again, to activate the new cookie settings for HTTP token authentication.

You can now use the web SSH feature accessible under ***Nodes → Actions → SSH Console***.

## 3.2 Users and Teams

ClusterControl comes with an advanced user management system, managed by ClusterControl Controller (cmon) service. All users' activities are logged and accessible via the ***Audit Log***, available under the ***Activity center*** section.

### 3.2.1 Users

Every operation initiated by ClusterControl clients (GUI or CLI) is associated with a user. A user must be authenticated with the controller service before performing any operation. ClusterControl supports private/public key and password authentication. A user can be disabled or enabled by anyone in the "admins" team.

**Info**

For security reason, a user will be disabled (suspended) after 3 consequent login failures over a predefined short time.

#### Create user

A ClusterControl user must belong to a primary team (primary team is the first group the user belongs to). Once created, the user can be assigned to more than one group if necessary.

**Note**

Note The username "admin" is reserved for ClusterControl internal usage. It is a system user and is not intended to be used by anyone.

### ClusterControl GUI

To create a user, go to ***Create user or team → Create user*** and fill up all the following information:

| Field | Description |
|---|---|
| *Details* | |
| **First name** | The first name of the user. |
| **Last name** | The last name of the user. |
| **Username** | The username of the user. In CC v2, the username shall be used when logging in to the login page, in contrast to CC v1 (old GUI) where you had to use e-mail address and password combination in order to login. |
| **Password** | The user's password. |
| **Email** | Email address of the user. This should be a valid email address (despite not mandatory), for password reset and receiving alerts via email. |
| **Timezone** | Choose a timezone from the dropdown list. This timezone will reflect the date/time presentation of the ClusterControl events and monitoring data points. |
| *Team* | |
| **Team** | Choose a team from the dropdown. This is considered the primary group for this user. A user can be assigned to multiple groups (teams). See *Create Team*. |

### ClusterControl CLI

- List all users:

```
s9s user --list --long
```

**Example**

```
$ s9s user --list --long
A ID UNAME      GNAME  EMAIL REALNAME
-  1 system     admins -     System User
-  2 nobody     nobody -     -
A  3 dba        users  -     -
-  4 remote_dba users  -     -
```

- Create an admin user called "dba" with private/public key authentication only (for local usage):

```
s9s user --create \
    --generate-key \
    --controller="https://localhost:9501" \
    --group=admins \
    dba
```

- Create a read-only user called "readeruser" with a same-name group and generate both key and password:

```
s9s user \
    --create \
    --group=readergroup \
    --create-group \
    --generate-key \
    --new-password=s3cr3tP455 \
    --email-address=reader@somewhere.com \
    --first-name=Reader \
    --last-name=User \
    --batch \
    readeruser
```

**Warning**

Attention

Starting from ClusterControl 1.9.7 (September 2023), upon logging in to ClusterControl GUI v2, make sure you use the `username` you have assigned or set during the **User Registration** page. Whereas in ClusterControl GUI v1, you can use e-mail address when logging in to the login page, but in CC v2, make sure you use the `username` and `password` combination in order to login to the home page.

### Edit user

Use the **Edit** action to modify existing user's information, re-assign the user to another primary team or force the user to change password:

### ClusterControl GUI

1. Login to ClusterControl GUI as a user in the "admins" team.
2. Go to ***ClusterControl GUI → User management → Users →*** find the user **→ *Actions → Edit***.
3. Update all information in the corresponding fields and click **Save**.

### ClusterControl CLI

- Set new first name, last name and email address for user "john":

```
s9s user \
    --set \
    --first-name=John \
    --last-name=Doe \
    --email-address=johndoe@example.com \
    --batch \
    john
```

- Set a new group for user "devops":

```
s9s user \
    --set-group \
    --group=infra_team \
    devops
```

**Info**

System users' information like "admin", "ccrpc", "system" and "nobody" are immutable.

#### Change password

Only users in the "admins" team can change password of other users. For users that belong to a non-admins team, they can only change their own password.

To change password for other users:

#### ClusterControl GUI

1. Login to ClusterControl GUI as a user in the "admins" team.
2. Go to ***ClusterControl GUI → User management → Users →*** find the user **→** ***Actions → Change Password***.
3. Speficy the new password in the **New password** field and click **Change**.

#### ClusterControl CLI

- Set new password for user "devops":

```
s9s user \
    --change-password \
    --new-password="MynewP455w0rd" \
    devops
```

**Info**

System users' password like "admin", "ccrpc", "system" and "nobody" are immutable.

### 3.2.2 Teams

Teams group users. Each team may consist of zero or more users. A single user may be a part of one team. Teams may be assigned to zero or more clusters.

#### Create team

When creating a team, the primary team where the user who creates it belongs, will be the team owner. Only the team owner is allowed to delete or modify the team. Only the owner's team is allowed to delete the cluster. All clusters created by a specific user also belong to the same team.

To create a team, go to ***Create user or team → Create team*** and fill up all the following information:

| Field | Description |
|---|---|
| *Details* | |
| **Team name** | The name of the team or group. |
| *Users* | |
| **Users** | Select one or more users from the list. You can always add users later. The selected users will be moved from their current teams to this new team. |
| *Permissions* | |
| **Change controller configuration** | Toggle on to allow users on this team to change all cluster configurations of this controller. |
| **Change LDAP settings** | Toggle on to allow users to change LDAP settings under the User Management page. |
| **Manage users and teams** | Toggle on to allow users to create, edit, and delete ClusterControl users and teams. |
| **Deploy clusters** | Toggle on to allow users to deploy a new cluster or import an existing cluster into ClusterControl. |
| **Clusters permission level** | Choose one of the permissions for all clusters. This permission can be adjusted later for each cluster. If you want to have fine-grained permission per cluster, choose Custom. See *Cluster-Level permissions*. |

**Note**

Note By default ClusterControl comes with two teams: “admins” and “users”. There is also a hidden team for internal usage called “nobody”.

### Cluster-level permissions

ClusterControl uses access control to restrict access to clusters and their respective deployment, management, and monitoring features. Access control defines privileges and permissions for a specific team. This ensures that only authorized user requests are allowed.

There are 3 types of access levels for clusters:

| Permission | Description |
|---|---|
| Manage | Allows viewing the cluster and its properties such as jobs, backups, charts, metrics, and settings of the cluster. It also allows changing the settings of the cluster and managing (clone, create, delete, abort ) jobs on the specific cluster.It does not allow the creation or deletion of the cluster, which is only permissible to the team's owner. |
| View | Allows viewing the cluster and its properties such as jobs, backups, charts, metrics, and settings of the cluster.The user can not modify the cluster. |
| No access | Access to the resource is denied. |
| Custom | Allows specific permission levels for each cluster. |

#### Edit team

Edit team allows administrator to add or remove users into the team, change global permissions and access level for every cluster managed by ClusterControl. Only the team owner can change the underlying team permission. For example, if user "dba" created a group called "infra", only user "dba" is able to edit "infra" team.

#### ClusterControl GUI

1. Login to ClusterControl GUI as a user who owns the team.
2. Go to ***ClusterControl GUI → User management → Teams →*** find the team **→** ***Actions → Edit***.
3. Configure the users, permission and access level sections and click **Save**.

#### ClusterControl CLI

- Set a new group for user "devops":

```
s9s user \
    --set-group \
    --group=infra_team \
    devops
```

- Add user "myuser" into group admins:

```
s9s user \
    --add-to-group \
    --group=admins \
    myuser
```

### 3.2.3 LDAP

ClusterControl supports integration with directory services like Active Directory, FreeIPA, and OpenLDAP authentication. This allows users to log into ClusterControl by using their corporate credentials instead of a separate password. LDAP groups can be mapped onto ClusterControl user groups to apply

privileges to the entire group. This ensures that ClusterControl authorizes the logged-in user based on the group assigned. It supports up to the LDAPv3 protocol based on RFC2307.

To integrate with a directory service, one has to perform the following steps:

1. Fill up the ***LDAP Settings*** configuration wizard. See *LDAP Settings*.
2. Save the settings. At this point, LDAP is saved but not activated because no group mapping has been created yet.
3. Create at least one group mapping entry by going to ***Map LDAP Group***. See *Map LDAP Group*.
4. Enable the LDAP authentication by toggling on the ***LDAP Settings → Enable LDAP Authentication***.
5. Log in to ClusterControl by using the value of LDAP ***Username Attributes*** with the respective password. The user will be authenticated and redirected to the ClusterControl dashboard page based on the assigned group.
6. From this point, both ClusterControl native authentication and LDAP authentications are enabled.

**Warning**

Attention

The last entry of the group mapping can not be deleted while LDAP authentication is enabled and activated. See *Delete mapping group*.

### LDAP settings

| Field | Description |
|---|---|
| *General Settings* | |
| **Enable LDAP Authentication** | Enables LDAP authentication. The native authentication in ClusterControl will also work. Refer to the steps mentioned above, on enabling LDAP authentication. |
| **LDAP URI** | Enter the LDAP or LDAPS Uniform Resource Identifier (URI), with the port number (if applicable). An example is `ldaps://ad.s9s.com:636`. For LDAPS, provide the certificates and key files under **SSL/TLS Settings** section. |
| **Login DN** | The Distinguished Name is used to bind the LDAP server. This user requires read access to all LDAP users and group entries to work correctly. ClusterControl must perform an LDAP search using the DN before any user can log in. This field is case-sensitive. An example is `cn=Administrator,cn=Users,dc=s9s,dc=com`. |
| **Login DN Password** | The password for Login DN. |
| **User Base DN** | The Distinguished Name (DN) to locate the users' information. This field is case-sensitive. An example is `cn=Users,dc=s9s,dc=com`. |
| **Group Base DN** | The Distinguished Name (DN) to locate the group information. ClusterControl does not support LDAP users that do not belong to at least one LDAP group. An example is `ou=Groups,dc=s9s,dc=com`. |
| *Advanced* | |
| **Prefill for** | Fill up all required fields based on common settings for Active Directory or OpenLDAP. Clicking on **Reset** will clear all textfields. |
| **User Base Filter** | Filter the object class of the LDAP users. If empty, all object classes will be returned. |
| **Username Attribute** | The LDAP attributes which hold the username, separated by a comma (whitespace value is not allowed). For Active Directory, this is commonly `sAMAccountName` and `uid` for OpenLDAP. When authenticating with ClusterControl (via ClusterControl GUI login page, ClusterControl CLI or RPC API), use the value of this attribute. If more than one attribute is specified, ClusterControl will attempt to look up all of them with the first non-empty reply (in the particular order) used for the login username. |
| **Real Name Attribute** | The LDAP attributes which hold the full name of the user, separated by a comma. For Active Directory, this is commonly `displayName` and `cn` for OpenLDAP. This value of this attribute will be used as the first and last name of the user. If more than one attribute is specified, ClusterControl will attempt to look up all of them with the first non-empty reply (in the particular order) used for the user's full name. |
| **Email Attribute** | The LDAP attributes which hold the email address of the user, separated by a comma. For Active Directory, this is commonly `userPrincipalName` and `mail` for OpenLDAP. This value of this attribute will be used as the email address of the user. If more than one attribute is specified, ClusterControl will attempt to look up all of them with the first non-empty reply |

### Map LDAP group

ClusterControl requires at least one LDAP group mapping entry to be defined before the LDAP authentication can be activated. Otherwise, ClusterControl would only save the LDAP settings without activating them.

To create a group mapping:

1. Click on the ***Map LDAP Group*** button.
2. Choose an existing team from the dropdown and specify the value of the LDAP's ***Group Name Attributes*** that you want to be mapped with the ClusterControl team. For single-tier RDN, specify the attribute value for example "DBA". ClusterControl will prepend the **Group name attribute** configured under the **LDAP Settings** with the value specified, and append it with the **Group base DN** value. The final constructed DN will be `{Group name attribute}={value specified},{Group base DN}`. For multi-tier RDNs, specify the attribute name and value (for multiple RDNs, delimited by a comma) for example `cn=DBA,cn=IT,ou=org1`. ClusterControl will append the value specified with the Group base DN value under the LDAP Settings. The final constructed DN will be `{value specified},{Group base DN}`.
3. After creating the first entry, you may proceed to activate the LDAP authentication by going to ***LDAP Settings → Enable LDAP Authentication***.

All LDAP configurations and mappings will be stored inside a configuration file at `/etc/cmon-ldap.cnf`. It is recommended to configure LDAP settings and group mappings via the ClusterControl GUI because any changes to this file will require a reload to the controller process, which is triggered automatically when configuring LDAP via the GUI.

**Note**

Note It is also possible to modify the configuration file directly, however, the `cmon` service must be reloaded manually by using the `systemctl restart cmon` command, or `service cmon restart`.

### Delete mapping group

Click on **Delete** of the entry in the list to delete it. The last entry of the group mapping can not be deleted while LDAP authentication is enabled and activated (the button will be greyed out). To delete the last entry, set **Enable LDAP Authentication** to OFF and you will notice the **Remove Group** button is now clickable, allowing you to remove the last group mapping entry and effectively deactivating LDAP authentication for ClusterControl.

## 3.3 Backup and Restore

Starting from ClusterControl 1.7.1, you can back up the ClusterControl server and restore it (together with metadata about your managed databases) onto another server using ClusterControl CLI. It backs up the ClusterControl application as well as all of its configuration data.

There are 4 options available under the `s9s backup` command:

| Flag | Description |
|---|---|
| `--save-controller` | Saves the state of the controller into a tarball. |
| `--restore-control` | Restores the entire controller from a previously created tarball (created by using `--save-controller`). |
| `--save-cluster-in` | Saves the information the controller has about one cluster. |
| `--restore-cluster` | Restores the information the controller has about a cluster from a previously created archive file. |

**Note**

Note These options manage backups of the ClusterControl controller and its metadata, not backups of the databases it monitors. To back up and restore a managed database cluster, see *Backup & Restore* in the User Guide.

### 3.3.1 Prerequisites

Before saving or restoring, make sure of the following:

- The source and target ClusterControl hosts run the same `s9s`/`cmon` version. Restoring across different minor versions may work for some cluster types but is not guaranteed, and increases the chance of hitting the known issues below.
- The SSH key ClusterControl uses to reach the database nodes exists at the same path on the target host. For example, if the key file is located under `/root/.ssh/id_rsa` on the source, make sure the same path and file exist on the target.
- The target host has working, passwordless SSH key-based access to every node in every cluster contained in the backup. Missing SSH access to even one node causes the restore job to fail during the grant step (see *Known issues and limitations*).
- `--restore-controller` requires a target host with no existing cluster configuration. If `/etc/cmon.d/` already contains a `cmon_<id>.cnf` file for one of the clusters being restored, the job fails immediately. Remove any conflicting cluster (`s9s cluster --drop --cluster-id=<id>`) and its leftover config file before retrying.

### 3.3.2 Back up all clusters

To back up the ClusterControl controller with all clusters together with their metadata, run the following command on the ClusterControl node as root user (or with sudo):

```
s9s backup \
    --save-controller \
    --backup-directory=$HOME/ccbackup \
    --output-file=controller.tar.gz \
    --log
```

The `--output-file` must be a filename or physical path (if you want to omit the `--backup-directory` flag), and the file must not exist beforehand. ClusterControl does not replace the output file if it already exists. By specifying the `--log` flag, it waits until the job is executed and shows the job logs in the terminal. The same logs can be accessed via ***ClusterControl GUI → Activity center → Jobs → Save Controller***.

The save controller job performs the following procedures:

1. Retrieve the controller configuration and export it to JSON.
2. Export the CMON database as a MySQL dump file.
3. For every database cluster:
    1. Retrieve the cluster configuration and export it to JSON.

**Note**

Note In the output, you may notice the job found is *N + 1* cluster, for example `Found 3 cluster(s) to save` even though you only have two database clusters. This includes cluster ID 0, which carries special meaning in ClusterControl as the global initialized cluster. It does not belong to the `CmonCluster` component, which is the database cluster under ClusterControl management.

### 3.3.3 Back up an individual cluster

To back up an individual cluster with its metadata, use the `--cluster-id` option to specify the cluster ID, on the ClusterControl node as root user (or with sudo):

```
s9s backup \
    --save-cluster-info \
    --cluster-id=2 \
    --backup-directory=$HOME/ccbackup \
    --output-file=cc-replication-2.tar.gz \
    --log
```

### 3.3.4 Restore an individual cluster to another server

Restoring an individual cluster's metadata is intended for moving one cluster's management from an old ClusterControl controller to a new one, not for restoring a cluster while it is still registered elsewhere.

1. Copy the backup file to the target host:

    ```
    scp cc-replication-2.tar.gz root@<new_server>:~/
    ```

2. On the target host, run the restoration. Do **not** pass `--cluster-id`; ClusterControl allocates a new cluster ID automatically from the archive contents:

    ```
    s9s backup \
        --restore-cluster-info \
        --input-file=$HOME/cc-replication-2.tar.gz \
        --debug \
        --log
    ```

3. Verify the cluster restored correctly:

    ```
    s9s cluster --list --long
    ```

### 3.3.5 Restore the entire controller to another server

Use this to migrate the controller and every cluster it manages onto a new host in a single operation, for example when replacing aging hardware or moving to a new operating system.

1. Install the same ClusterControl version on the new host. See *Installation*.
2. Make sure the same SSH key ClusterControl uses to reach the database nodes exists at the same path on the new host, and that it has working access to **every** node in **every** cluster contained in the backup (see *Prerequisites*).
3. Perform the restoration:

```
s9s backup \
    --restore-controller \
    --input-file=$HOME/controller.tar.gz \
    --debug \
    --log
```

4. Wait a moment, then verify the clusters restored correctly:

```
s9s cluster --list --long
```

> **Tip**
>
> If the cluster list comes back empty right after the restore, wait about 30 seconds and run the command again. `cmon` is still restarting.

### 3.3.6 Known issues and limitations

The following behavior has been confirmed when restoring onto a target running the same ClusterControl version:

- **The monitoring/Prometheus host is not repointed to the new controller.** Both `--restore-cluster-info` and `--restore-controller` correctly update the controller host entry to the new server's address, but the Prometheus monitoring host entry keeps pointing at the old server's IP. Update this manually, or follow *Migrating ClusterControl to Another Server*, which installs Prometheus fresh on the new host as part of the procedure.
- **`s9s cluster --drop` is the correct option to unregister a cluster**, not `--remove-cluster` (which does not exist and returns `unrecognized option`). Dropping a cluster removes it from ClusterControl only; the database itself keeps running.
- **`--restore-cluster-info` only completes successfully when the target controller has no other cluster already registered.** Every cluster shares the same controller-side Prometheus/monitoring host entry; if the target already manages another cluster, restoring a second one fails while trying to recreate that shared monitoring host object (`Host is already part of some other cluster`), and the job rolls back everything it created. Always restore onto a target host with no existing clusters.
- Dropping a cluster may log `Failed to purge cluster data on dcps db: Unknown database 'dcps'`. This is informational only and does not affect the drop operation; the `dcps` database is an optional component not present on all installations.

## 3.4 Configuration Files

A single `cmon` process is capable of managing one or more database clusters. Each of the clusters requires one exclusive configuration file residing in the `/etc/cmon.d/` directory. For instance, the default `cmon` configuration file is located at `/etc/cmon.cnf`, and commonly used to store the default (minimal) configuration for the `cmon` process to run.

Example of the CMON main configuration file located at `/etc/cmon.cnf`:

```
# Content of /etc/cmon.cnf

mysql_port=3306
mysql_hostname=127.0.0.1
mysql_password=cm0nP4ss
hostname=10.0.0.196
logfile=/var/log/cmon.log
rpc_key=390faeffb8166277a4f25336a69efa50915635a7
```

For the first cluster (cluster_id=1), the configuration options should be stored inside `/etc/cmon.d/cmon_1.cnf`. For the second cluster, it would be `/etc/cmon.d/cmon_2.cnf` with `cluster_id=2` respectively, and so on. The following shows an example content of `cmon` cluster's configuration file located at `/etc/cmon.d/cmon_4.cnf`:

```
# Content of /etc/cmon.d/cmon_4.cnf

cluster_id=4
cmon_user=cmon
created_by_job=1
db_stats_collection_interval=30
enable_query_monitor=1
galera_vendor=percona
galera_version=3.x
group_owner=1
host_stats_collection_interval=60
hostname=10.0.0.196
logfile=/var/log/cmon_4.log
mode=controller
monitored_mountpoints=/var/lib/mysql/
monitored_mysql_port=3306
monitored_mysql_root_password='7XU@Wy4nqL9'
mysql_bindir=/usr/bin/
mysql_hostname=127.0.0.1
mysql_password='cm0nP4ss'
mysql_port=3306
mysql_server_addresses=10.0.0.99:3306,10.0.0.253:3306,10.0.0.181:3306
mysql_version=5.6
name='Galera Cluster'
os=redhat
osuser=root
owner=1
pidfile=/var/run
basedir=/usr
```

(continues on next page)

(continued from previous page)

```
repl_password='9hHRgQLSsZz3Vd4a'
repl_user=rpl_user
rpc_key=3V0RaV6dE8KSyClE
ssh_identity=/root/.ssh/id_rsa
ssh_port=22
type=galera
vendor=percona
```

An example of `cmon` configuration file hierarchy is as follows:

| Example cluster | Configuration file | Cluster identifier | Log file location |
|---|---|---|---|
| Default configuration | `/etc/cmon.cnf` | N/A | `logfile=/var/log/cmon.log` |
| Cluster #1 (Galera) | `/etc/cmon.d/cmon_1.cnf` | `cluster_id=1` | `logfile=/var/log/cmon_1.log` |
| Cluster #2 (MongoDB) | `/etc/cmon.d/cmon_2.cnf` | `cluster_id=2` | `logfile=/var/log/cmon_2.log` |
| Cluster #N (cluster type) | `/etc/cmon.d/cmon_N.cnf` | `cluster_id=N` | `logfile=/var/log/cmon_N.log` |

**Note**

Note It's highly recommended to separate `cmon` logging for each cluster to its own log file. In the above example, we can see that `cluster_id` and `logfile` are two important configuration options to distinguish the cluster.

The ClusterControl Controller will import the configuration options defined in each configuration file into the CMON database during the process starts up. Once loaded into the runtime configuration, `cmon` uses all of these information to manage the database clusters accordingly.

## 3.5 Runtime configuration

The runtime configuration is considered the active configuration and can be viewed via ClusterControl GUI by going to ***ClusterControl → choose a database cluster → Settings → Runtime Configuration*** or by using the ClusterControl CLI:

```
s9s cluster --list-config --cluster-id=1
```

For list of all configuration options, see *Configuration Options*.

## 3.6 Applying configuration changes

It is worth noting that ClusterControl configuration options can be set in multiple ways. This can be done by the following approaches:

- Editing the file `/etc/cmon.d/cmon_X.cnf`, where X is the cluster id of your cluster. *Although this approach is not recommendable and strongly discouraged.*

- Via the web UI by going to ***ClusterControl → choose a database cluster → Settings → System settings***. Then search for the desired parameter you plan to change and click on the pencil icon to modify the value.
- Using *s9s cluster* command. For example:

```
s9s cluster \
    --change-config \
    --cluster-id=25 \
    --opt-name=replication_pre_failover_script \
    --opt-value="/root/check_before_failover.sh"
```

There are certain variables that upon changes via runtime, shall reflect immediately. However, it is strongly recommended that you have to restart the `cmon` daemon whenever you apply changes to the configuration options.

For list of all configuration options, see *Configuration Options*.

## 3.7 Securing ClusterControl

This guide provides ways to secure your ClusterControl installation.

### 3.7.1 Firewall and security group

Depending on the deployment options, ClusterControl might disable firewalls to minimize the possibilities of failure during the cluster deployment. Once the process is completed, it is important to secure the ClusterControl node and the database cluster. We recommend isolating your database infrastructure from the public Internet and allowing only known hosts or networks to connect to the database cluster.

See *Firewall and Security Groups* for the full list of ports ClusterControl requires to be open.

### 3.7.2 SSH

SSH is very critical for ClusterControl. It must be possible to SSH from the ClusterControl server to the other nodes in the cluster without a password, thus the database nodes must accept the SSH port configured in the CMON configuration file. The following best practices are recommended:

- Permit very few people in the organization to access the servers. The fewer the better.
- Lock down SSH access so it is not possible to SSH into the nodes from any other server than the ClusterControl server.
- Lock down the ClusterControl server so that it is not possible to SSH into it directly from the outside world.

### 3.7.3 File permission

CMON configuration and log files contain sensitive information, such as the `mysql_password` value or the sudo password used for SSH privilege escalation. Ensure the CMON configuration file (e.g., `/etc/cmon.cnf` and `/etc/cmon.d/cmon_{clusterid}.cnf`, if present) has permission 700. CMON log files (e.g., `/var/log/cmon.log` and `/var/log/cmon_{clusterid}.log`) should have permission 740. Both the configuration and log files should be owned by root.

### 3.7.4 HTTPS

By default, the installation script installs and configures a self-signed certificate for ClusterControl GUI. You can access it by pointing your browser to `https://{ClusterControl_host}/`. If you would like to use a properly signed SSL certificate, replace the `tls_cert` and `tls_key` path inside `/usr/share/ccmgr/ccmgr.yaml` and restart the ClusterControl Proxy service. Details at *ClusterControl Proxy → Configure Custom SSL Certificates*.

### 3.7.5 Access control and identity

Securing the host is only part of the picture. ClusterControl also provides application-level controls for managing who can access the system and what they're allowed to do once they're in:

- **Role-based access control** ^==Enterprise==^{: .badge-enterprise}: Group users into teams with defined permissions instead of sharing a single admin account. See *Users and Teams*.
- **LDAP/Active Directory authentication** ^==Enterprise==^{: .badge-enterprise}: Authenticate users against your corporate directory, with LDAP groups mapped to ClusterControl permissions, instead of maintaining separate local passwords. See *LDAP*.
- **Audit logging**: Record logins, queries, data modifications, and schema changes on the managed database nodes for security and compliance purposes. Currently supported for MariaDB Replication/Galera, PostgreSQL, and TimescaleDB. See *Audit Logs*.

### 3.7.6 Secrets and encryption

- **HashiCorp Vault integration** ^==Enterprise==^{: .badge-enterprise}: Store and load database and cluster credentials from an external Vault server instead of in plaintext inside the CMON configuration files described under *File permission* above. See HashiCorp Vault Integration.
- **Certificate management** ^==Enterprise==^{: .badge-enterprise} ^==Advanced==^{: .badge-advanced}: Generate, import, and organize a CA and server/client certificates from the GUI, then assign them to a cluster to enable SSL/TLS encryption between the controller and the database nodes. See *Certificates*.
- **Backup encryption**: Enable the encryption option when scheduling a backup to have ClusterControl auto-generate and manage an AES-256 encryption key for it. See *Create Backup*.

ClusterControl can be deployed in a couple of different ways for redundancy and high availability:

1. *Secondary standby* - Acts as a hot standby in case the primary ClusterControl host goes down.
2. *CMON Controller Pool* - A scalable pool of controllers designed for demanding, multi-site and high-growth environments.
3. *ClusterControl Replication* - A high-availability deployment using replication with a primary-secondary architecture.

## 3.8 Secondary Standby

It is possible to have more than one ClusterControl server to monitor the same cluster. This is useful if you have a multi-datacenter cluster and you may need to have ClusterControl on the remote site to monitor and manage the alive nodes if the connection between them goes down. However, ClusterControl servers must be configured to be working in active-passive mode to avoid race conditions when recovering failed nodes or clusters.

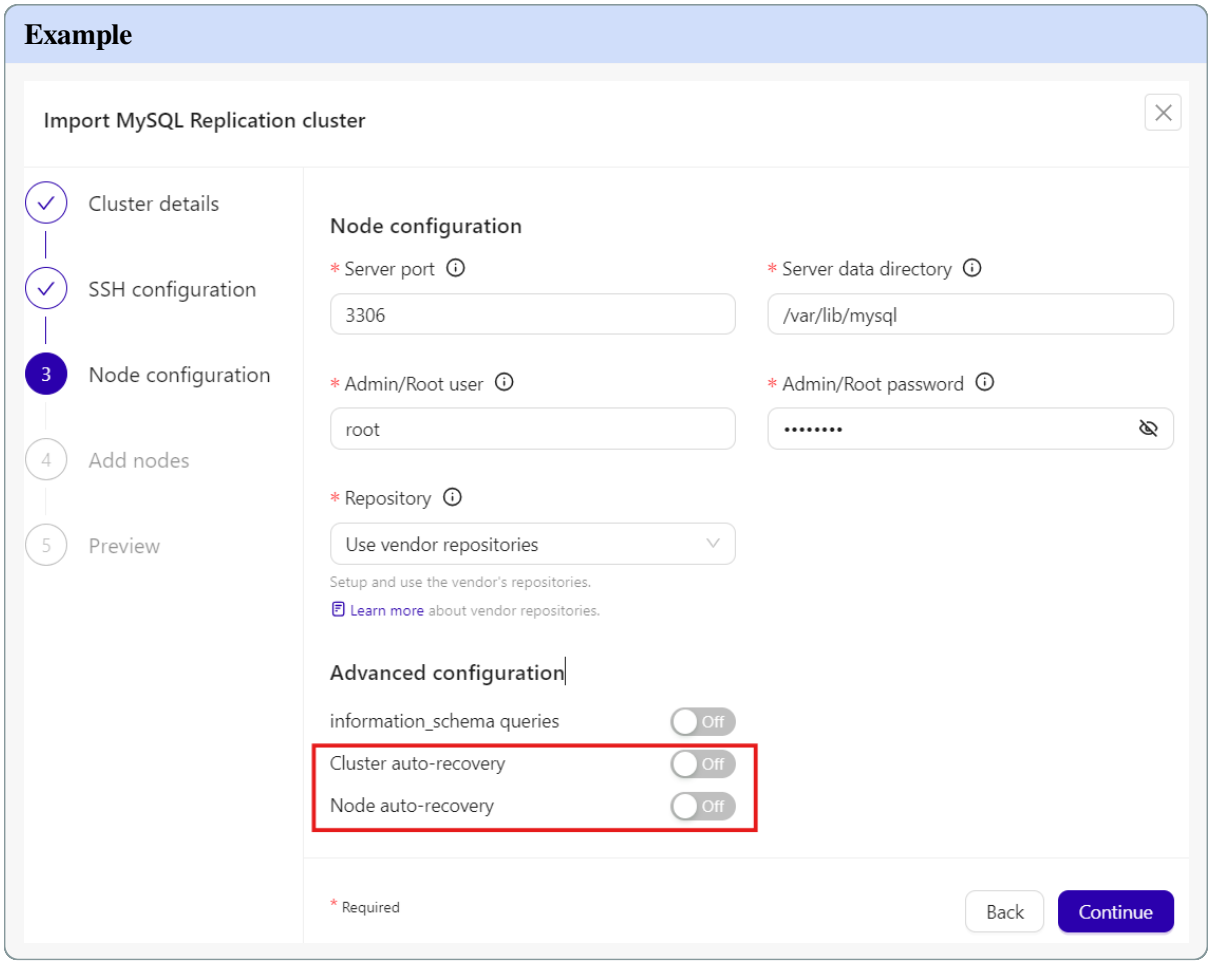
In the active mode, the ClusterControl node acts as a primary controller, where it performs automatic recovery and management activities. Therefore the primary controller **Cluster/Node Auto Recovery** must be set to on. The secondary ClusterControl node however must be configured with **Cluster/Node Auto Recovery** turned off.

### 3.8.1 Installing standby server

The steps described in this section must be performed on the secondary ClusterControl server.

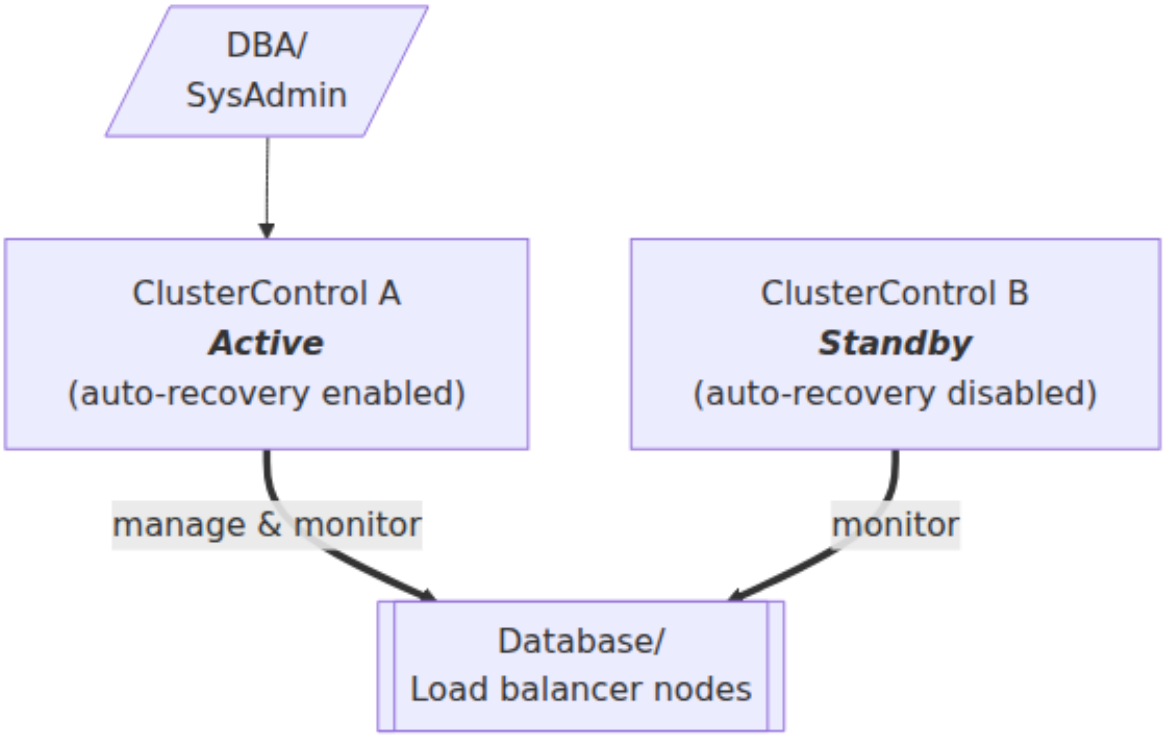
1. Install ClusterControl as explained on the *Quickstart*.
2. Import the same cluster via ***ClusterControl GUI → Deploy a cluster → Import a database cluster***. Ensure to toggle off **Cluster auto-recovery** and **Node auto-recovery** in the **Node configuration** section. Repeat this step if you want to import more than one cluster.

**Example**




3. Set up the cluster configuration accordingly to follow similar settings with the primary ClusterControl (backup schedules, alerting configuration, user roles, etc).

At this point, the architecture should be looking like this:




Nothing should be performed on the primary side. The primary ClusterControl server shall perform automatic recovery in case of node or cluster failure. Use the secondary ClusterControl server for monitoring purposes only. For management and recovery purposes like rebuilding the replication, resyncing the node, backup and restore, perform those activities on the primary ClusterControl server.

**Info**

You don't need an additional ClusterControl license for multiple ClusterControl instances. You can apply the same license as your primary ClusterControl server onto the secondary server. The license is bounded on the number of database/load balancer nodes it manages.

### 3.8.2 Activating the secondary standby

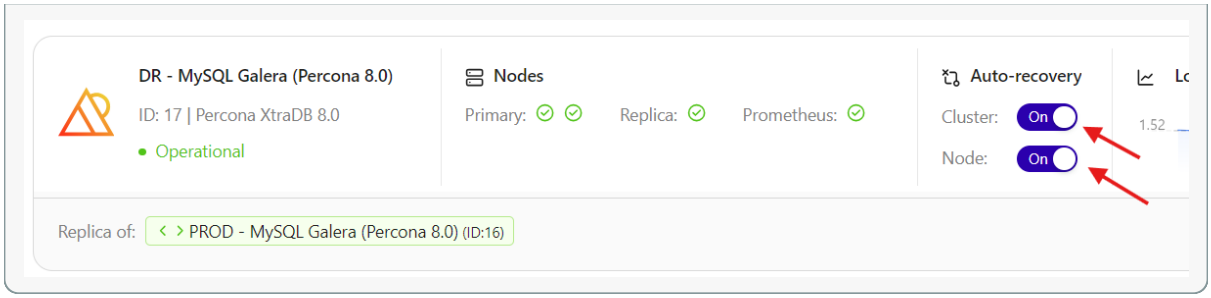
If you want to make the standby server runs in the active mode, you must do the following:

1. If the primary ClusterControl server is still alive, stop the primary ClusterControl controller services, or shutdown the server. To stop all ClusterControl processes, run the following command on the primary ClusterControl server:

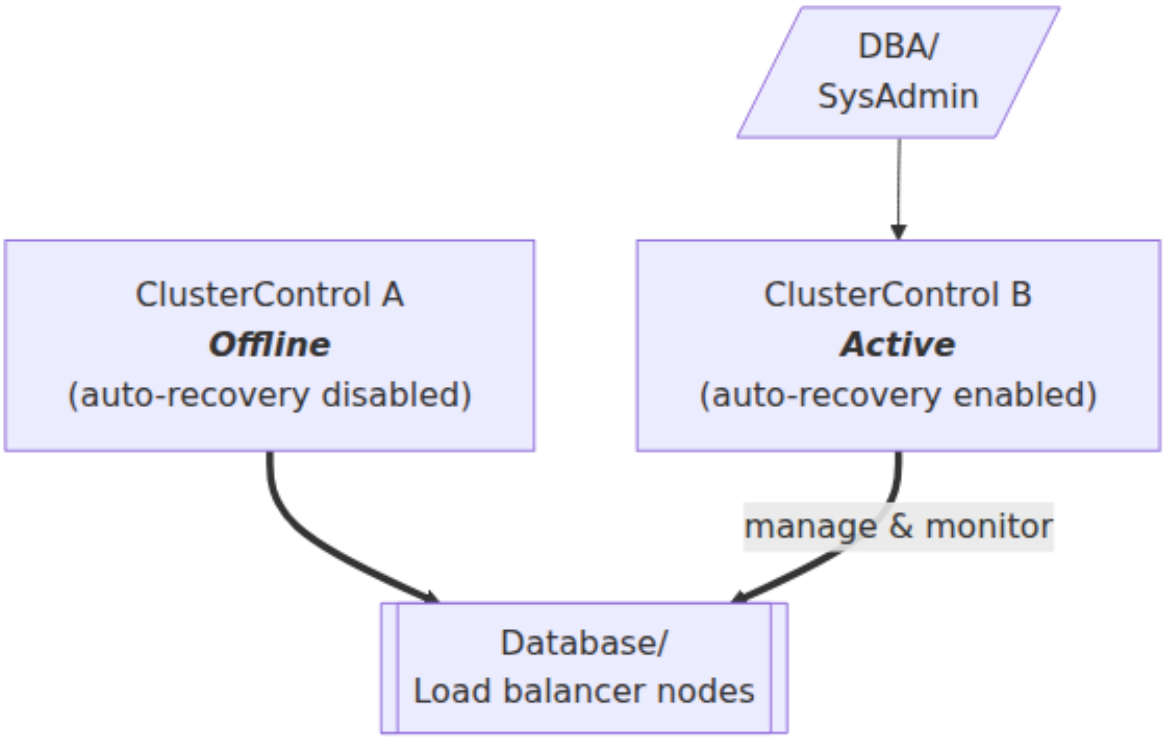
```
systemctl stop cmon cmon-cloud cmon-ssh cmon-events
```

2. Toggle on **Cluster auto-recovery** and **Node auto recovery** on the secondary ClusterControl server.

**Example**




At this point, the standby server has taken over the primary role and you can perform the management activities on the database nodes or clusters. We can visualize the current topology as below:



**Warning**

Attention

Do not let two or more ClusterControl instances perform automatic recovery to the same cluster at a given time.

## 3.9 Scalable Controllers Pool

The Scalable Controllers Pool removes the operational limits and failure risk of relying on a single CMON controller in larger ClusterControl environments. As the number of managed clusters grows, one controller can become a capacity bottleneck and a single failure domain for cluster monitoring, orchestration, and job handling.

### 3.9.1 How it works

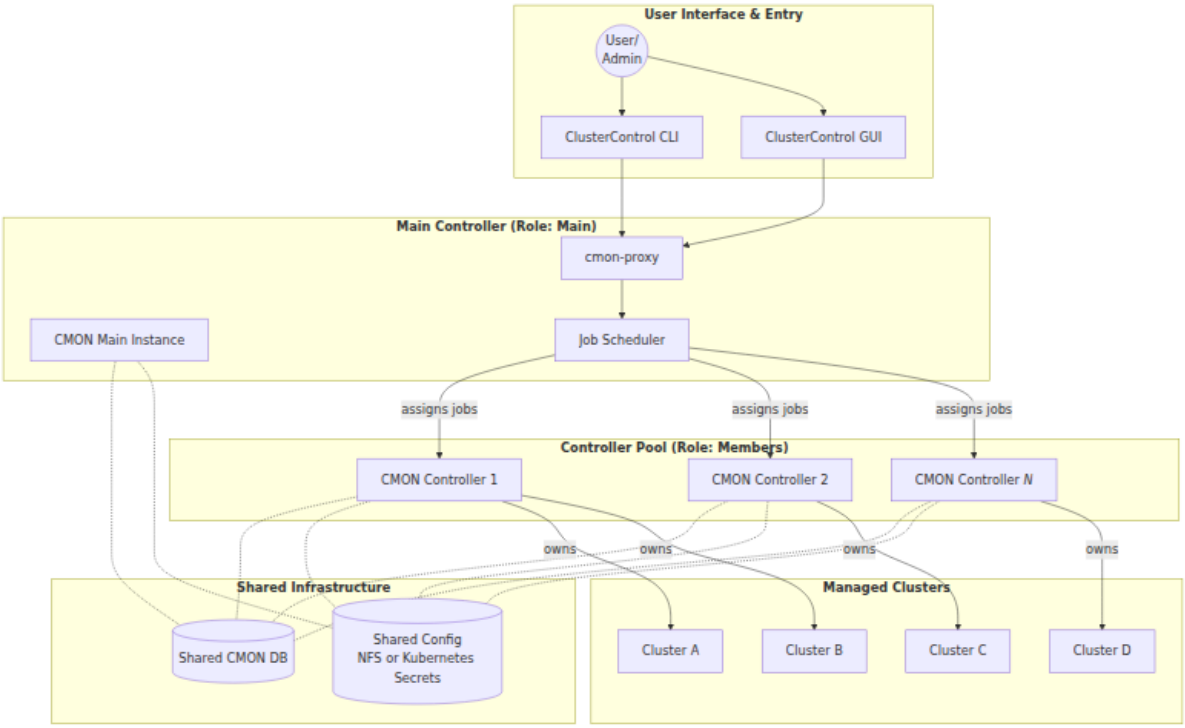
The pool lets multiple CMON controllers coordinate cluster ownership: controller capacity scales horizontally, and ownership moves automatically when controllers are added, removed, restarted, or become unavailable.

The pool works identically regardless of the underlying infrastructure. Controllers can run on virtual machines, bare-metal servers, or cloud-hosted instances using NFS or Vault as the shared configuration backend, or on Kubernetes using Kubernetes Secrets. The only requirement is that every controller in the pool connects to the same CMON database and shares the same configuration backend.

All pool members share a common source of truth:

- All pool members connect to the **same CMON database**.
- All pool members must use a **shared configuration backend**.
- Each controller publishes its presence and status.
- Clusters are owned by one controller at a time, and ownership can move when controllers are added, removed, restarted, or become stale.

Here's a diagram on how the controller pool works:




### 3.9.2 Key runtime concepts

- **Controllers table**: stores controller identity, endpoint, status, and heartbeat information.
- **Cluster assignment table**: tracks which controller owns which cluster.
- **Refresh cycle**: each controller refreshes status and adjusts ownership.
- **Staleness threshold**: if a controller stops refreshing in time, peers can reclaim its clusters.
- **Fair-share ownership**: cluster ownership is distributed across active controllers, subject to controller capacity limits.

Operationally, a controller only needs to manage the subset of clusters that it currently owns. When ownership changes, the controller loads or unloads cluster state accordingly.

### 3.9.3 Shared configuration requirement

Shared configuration is a hard requirement in pool mode. ClusterControl supports the following backends:

- Kubernetes Secrets for Kubernetes-native environments.
- NFS or shared filesystem for VM and non-Kubernetes deployments.
- Vault as a secure backend direction for non-Kubernetes/shared deployments.

### 3.9.4 Deployment and configuration model

Minimum prerequisites:

- All controllers must connect to the **same CMON database**.
- Each controller must have a unique **hostname:port** identity.
- All controllers must use the **same shared configuration backend**.

- For Kubernetes shared-config deployments, controllers are started with `--pool --k8s`.

### 3.9.5 Example startup pattern for non-Kubernetes environment shared config

```
# /etc/default/cmon
EXTRA_OPTS=--pool
```

### 3.9.6 Example startup pattern for Kubernetes-backed shared config

```
# /etc/default/cmon
EXTRA_OPTS=--pool --k8s
```

In multi-VM or external-controller setups using Kubernetes Secrets as the shared configuration store, the documented pattern also includes Kubernetes API endpoint and certificate environment variables so that each controller can access the same secret store.

```
KUBERNETES_SERVICE_HOST=hostA
KUBERNETES_SERVICE_PORT=16443
KUBERNETES_SERVICE_CA_CERT_PATH=/etc/certs/ca.crt
KUBERNETES_CLIENT_CERT_PATH=/etc/certs/client.crt
KUBERNETES_CLIENT_KEY_PATH=/etc/certs/client.key
```

### 3.9.7 Non-Kubernetes deployment model

The feature is also intended for non-Kubernetes environments. In those cases, the shared configuration backend is typically a shared filesystem such as NFS, while controllers still point to the same CMON database. This makes the feature applicable to:

- Traditional VM deployments
- Physical server or bare-metal deployments
- Cloud VMs or instances outside Kubernetes

### 3.9.8 Capacity and cold-start hardening

The following settings control how many clusters a controller may own and how quickly a new or restarted controller claims them.

| Configuration key | Meaning | Default behavior described |
|---|---|---|
| `controllers_pool_start_interva` | Refresh cycles before a starting controller becomes active | 2 cycles |
| `controllers_pool_max_clusters` | Per-controller ownership cap | `-2` auto, `-1` unlimited, `0` inactive, `>0` explicit cap |
| `controllers_pool_clusters_per_` | RAM-based multiplier for auto capacity | 10 clusters per GB |
| `controllers_pool_cluster_claim` | Clusters claimed per refresh cycle during slow start | Auto, about 20% of max capacity |
| `controllers_pool_soft_alarm_se` | Warning threshold for unassigned clusters | 75 seconds |
| `controllers_pool_hard_alarm_se` | Critical threshold for unassigned clusters | 150 seconds |

These controls help avoid a cold-start controller claiming too many clusters at once, starving heartbeats, and triggering reassignment storms.

### 3.9.9 Managing the Controller Pool

#### Enabling the controller pool

Enabling the controller pool can be done from either the GUI or the `s9s` CLI.

#### ClusterControl GUI

1. In the left-side navigation bar, click **Controller pool**. If pool mode is not yet enabled, the UI notifies you and shows a **Restart CMON in pool mode** button.
2. Click **Restart CMON in pool mode**.
3. In the **Allowed controllers network** field, specify the CIDR block or blocks controllers are allowed to connect from (for example, `192.168.40.0/24, 10.120.0.0/24, 10.8.0.0/24`). Click **Enable**.

   Enabling pool mode adds a `granted_controllers_network_mask` entry to `/etc/cmon.cnf` (for example, `granted_controllers_network_mask=192.168.40.0/24,192.168.121.0/24`). The controller you enabled pool mode on keeps its existing clusters and is assigned the `main` role.

#### ClusterControl CLI

The `s9s` CLI adds a `pool-controllers` mode to manage the controller pool. To enable pool mode:

```
s9s pool-controllers --set-pool-mode --conf-storage=nfs --
↪granted-network-mask=192.168.40.0/24
```

- `--set-pool-mode` enables the controller pool.
- `--conf-storage=nfs` sets the shared configuration backend. Use `k8s` for Kubernetes Secrets instead. Only `nfs` and `k8s` are allowed.
- `--granted-network-mask=192.168.40.0/24` sets the CIDR block controllers are allowed to connect from.

### Prerequisites (ClusterControl 2.5.0+)

From ClusterControl 2.5.0, the first controller of a pool needs two prerequisites before pool mode can be enabled on it:

- **CC DB cluster**: cmon's own database made highly available, as a MySQL InnoDB Cluster behind a local MySQL Router. cmon connects through the Router, and other controllers and additional database instances join the cluster later. The target version is Oracle MySQL 8.4 LTS. An existing MySQL 8.0 cmon database is accepted, but the bootstrap logs a warning that MySQL 8.0 has reached end of life and that you should move to 8.4 LTS.
- **CC configuration storage**: an OpenBao secret store that all controllers of the pool share.

Enabling pool mode no longer sets these up for you. Each one is created by its own job, so you can verify every stage before you restart cmon in pool mode. Until both are in place, enabling pool mode on the first controller fails, and the missing prerequisites are listed. There is nothing to set up when the controller runs in Kubernetes mode or when other controllers are already in the pool.

### ClusterControl GUI

In the left-side navigation bar, click **Controller pool**. Every missing prerequisite is shown with an explanation of what it is for and a button that starts the job to set it up. Once both prerequisites are ready, the page continues with the enable wizard described in *Enabling the controller pool*.

### ClusterControl CLI

1. Check which prerequisites are still missing. This command is read-only and works while pool mode is off. It prints the commands to run, in order:

   ```
   s9s pool-controllers --pool-readiness
   ```

2. If cmon's database is MariaDB, migrate it to Oracle MySQL first, because the CC DB cluster needs Oracle MySQL:

   ```
   s9s pool-controllers --migrate-db
   ```

   The automatic migration is supported only on RHEL-family hosts (EL8, EL9 and EL10). On Debian and Ubuntu it is refused, and `--pool-readiness` shows the reason. Follow *Migrating the CMON database to MySQL on Debian and Ubuntu* instead.

3. Bootstrap the CC DB cluster. This job does not restart cmon, and it does nothing if the cluster already exists:

   ```
   s9s pool-controllers --bootstrap-db --log
   ```

4. Install and bootstrap OpenBao on a host of your choice:

   ```
   s9s pool-controllers --add-openbao --nodes=HOST --log
   ```

5. Run `--pool-readiness` again. When it reports ready, enable pool mode:

   ```
   s9s pool-controllers --set-pool-mode
   ```

If you run cmon's database HA or the configuration storage yourself, skip the matching check with `--no-require-db-cluster` or `--no-require-config-storage`, added to `--set-pool-mode`.

**Warning**

The migration stops cmon

`--migrate-db` runs in the background, and **cmon is stopped for several minutes**. While it is stopped, managed clusters get no monitoring, no auto-recovery and no scheduled backups. Plan a maintenance window. The migration is refused while cmon has running jobs, so wait for them to finish first. Until cmon is back, `--pool-readiness` cannot connect. If the migration fails before cmon is running on MySQL, it rolls back to MariaDB and restarts cmon. Check `journalctl -u cmon-db-migration` before you retry.

### Migrating the CMON database to MySQL on Debian and Ubuntu

On Debian and Ubuntu, `--migrate-db` is not available, so you have to move cmon's database from MariaDB to Oracle MySQL 8.4 LTS by hand. Run every step below as `root` on the controller host.

1. **Check that the procedure applies.** `s9s pool-controllers --pool-readiness` reports the DB backend as `mariadb` and says the migration is not supported on this host. cmon, and with it auto-recovery, stays down for the whole procedure, so plan a maintenance window. Take a backup of cmon's database before you start.

2. **Record the current setup and back up the configuration.** Note the database user cmon logs in as (`cmon_user`, or its alias `cmondb_user`, in `/etc/cmon.cnf`; `cmon` if neither is set), the `mysql_*` keys (hostname, port, socket and password), the hosts of that database user, and the installed MariaDB packages. The following steps use the `CMON_USER` shell variable, so run them in the same shell:

```
grep -E '^(cmon_user|cmondb_user|mysql_)' /etc/cmon.cnf
CMON_USER=$(sed -n -E
↪'s/^(cmon_user|cmondb_user)[[:space:]]*=[[:space:]]*//p' /
↪etc/cmon.cnf | tr -d "\"' " | head -n 1)
CMON_USER=${CMON_USER:-cmon}
mariadb -u"$CMON_USER" -p -e "SELECT user, host FROM mysql.user WHERE␣
↪user = '$CMON_USER'"
dpkg -l | grep -i mariadb
cp -a /etc/cmon.cnf /root/cmon.cnf.mariadb
cp -a /etc/cmon.d /root/cmon.d.mariadb
```

3. **Stop the ClusterControl services:**

```
systemctl stop cmon cmon-cloud cmon-events cmon-ssh
```

4. **Dump the cmon databases and save the row counts.** Leave `dcps` out if that database does not exist:

```
mariadb-dump -u"$CMON_USER" -p --single-transaction --routines --
↪triggers \
  --databases cmon dcps > /root/cmon-mariadb.sql
```

   Recent MariaDB versions add a first line containing `enable the sandbox mode`, which MySQL cannot parse. Delete it. Then check that the dump uses no MariaDB-only `uca1400` collations. The count must be `0`. If it is not, stop here, because this procedure does not cover that case:

```
sed -i '1{/enable the sandbox mode/d}' /root/cmon-mariadb.sql
grep -c uca1400 /root/cmon-mariadb.sql
```

Save the per-table row counts to compare after the import:

```
mariadb -u"$CMON_USER" -p -N -e "SELECT CONCAT('SELECT ''',␣
↪table_schema, '.', table_name, ''', COUNT(*) FROM \`', table_schema,
↪'\`.\`', table_name, '\`;') FROM information_schema.tables WHERE␣
↪table_schema IN ('cmon', 'dcps') AND table_type = 'BASE TABLE' ORDER␣
↪BY 1" > /root/cmon-count.sql
mariadb -u"$CMON_USER" -p -N < /root/cmon-count.sql > /
↪root/cmon-counts.mariadb.txt
```

5. **Stop MariaDB and archive its data and configuration:**

```
systemctl stop mariadb
systemctl disable mariadb
tar -C /var/lib -czpf /root/mariadb-datadir.tgz mysql
cp -a /etc/mysql /root/etc-mysql.mariadb
```

6. **Remove the MariaDB server.** Use `remove`, not `purge`, so that a rollback is possible. Simulate the removal first with `apt-get -s`, and do not continue if it would also remove ClusterControl packages. Then move the old data directory aside:

```
apt-get -s remove mariadb-server mariadb-client
apt-get remove -y mariadb-server mariadb-client
mv /var/lib/mysql /var/lib/mysql.mariadb
```

7. **Install Oracle MySQL 8.4 LTS** from repo.mysql.com. This uses the same keyring as the repository ClusterControl sets up for MySQL Shell and MySQL Router. Check that repo.mysql.com has a suite for your release. On Ubuntu 26.04 you can skip the repository and install the distribution's own MySQL 8.4 packages:

```
curl -fsSL https://repo.mysql.com/RPM-GPG-KEY-mysql-2025 | \
  gpg --batch --yes --dearmor -o /usr/share/keyrings/mysql.gpg
. /etc/os-release
echo "deb [signed-by=/usr/share/keyrings/mysql.gpg]␣
↪https://repo.mysql.com/apt/${ID}/ ${VERSION_CODENAME} mysql-8.4-lts"␣
↪\
  > /etc/apt/sources.list.d/mysql-8.4-lts-community.list
apt-get update
apt-cache policy mysql-server
apt-get -s install 'mysql-server=8.4.*'
DEBIAN_FRONTEND=noninteractive apt-get install -y 'mysql-server=8.4.*'
mysql -uroot -e "SELECT VERSION()"
```

Before you install, check that `apt-cache policy` lists an `8.4` version from repo.mysql.com. Also check that the `apt-get -s` dry run installs 8.4 packages only and removes no ClusterControl packages. After the install, `SELECT VERSION()` must return `8.4.x`.

> **Note**
>
> Note ClusterControl writes `/etc/apt/preferences.d/s9s-repo-mysql-com-pin`, which gives repo.mysql.com priority 1 so that only MySQL Shell and MySQL Router come from it. If that file exists, you must keep the explicit `=8.4.*` version, or apt installs the distribution's MySQL instead. If the dry run still resolves a dependency to a distribution package, add that package to the command with `=8.4.*` as well.

8. **Recreate the cmon database users** with the password from `mysql_password` in `/etc/cmon.cnf`. Create one user for every host recorded in step 2, at least `localhost` and `127.0.0.1`. Keep MySQL's default authentication plugin. Replace `<cmon_user>` with the value of `$CMON_USER` and `<cmon_password>` with that password:

```
CREATE USER '<cmon_user>'@'localhost' IDENTIFIED BY '<cmon_password>';
GRANT ALL ON *.* TO '<cmon_user>'@'localhost' WITH GRANT OPTION;
CREATE USER '<cmon_user>'@'127.0.0.1' IDENTIFIED BY '<cmon_password>';
GRANT ALL ON *.* TO '<cmon_user>'@'127.0.0.1' WITH GRANT OPTION;
```

   Run these statements with `mysql -uroot`.

9. **Import the dump and compare the row counts.** The diff must be empty:

```
mysql -uroot < /root/cmon-mariadb.sql
mysql -uroot -N < /root/cmon-count.sql > /root/cmon-counts.mysql.txt
diff /root/cmon-counts.mariadb.txt /root/cmon-counts.mysql.txt
```

10. **Point cmon at MySQL.** In `/etc/cmon.cnf`, and in every `/etc/cmon.d/cmon_*.cnf` that sets these keys, set:

```
mysql_hostname=127.0.0.1
mysql_port=3306
mysql_socket=/var/run/mysqld/mysqld.sock
```

11. **Start the services and verify:**

```
systemctl start cmon cmon-cloud cmon-events cmon-ssh
s9s pool-controllers --pool-readiness
```

    The DB backend must now be `mysql`, with no migration required.

12. **Continue with the remaining prerequisites** in the GUI or the CLI: bootstrap the CC DB cluster (`--bootstrap-db`), set up OpenBao (`--add-openbao`), then enable pool mode (`--set-pool-mode`). See *Prerequisites (ClusterControl 2.5.0+)*.

> **Warning**
>
> Rollback
>
> If a step fails before cmon is running on MySQL, go back to MariaDB:
>
> ```
> systemctl stop cmon cmon-cloud cmon-events cmon-ssh
> systemctl stop mysql
> systemctl disable mysql
> ```

```
mv /var/lib/mysql /var/lib/mysql.oracle
mv /etc/mysql /etc/mysql.oracle
mv /var/lib/mysql.mariadb /var/lib/mysql
cp -a /root/etc-mysql.mariadb /etc/mysql
rm /etc/apt/sources.list.d/mysql-8.4-lts-community.list
apt-get update
apt-get remove -y mysql-server
DEBIAN_FRONTEND=noninteractive apt-get install -y -o Dpkg::Options::=--
↪force-confold \
  mariadb-server mariadb-client
cp -a /root/cmon.cnf.mariadb /etc/cmon.cnf
rm -rf /etc/cmon.d && cp -a /root/cmon.d.mariadb /etc/cmon.d
systemctl enable --now mariadb
systemctl start cmon cmon-cloud cmon-events cmon-ssh
```

Reinstall the MariaDB packages and versions you recorded in step 2 (for example `mariadb-server=VERSION`). If they came from the MariaDB repository and not from the distribution, install them from that repository again. The MariaDB data and configuration are back in place before the packages are reinstalled, so the installation reuses them. If `/var/lib/mysql.mariadb` is missing or damaged, restore the data directory from the archive instead, with `tar -C /var/lib -xzpf /root/mariadb-datadir.tgz`.

### Adjusting controller capacity at runtime

Set the maximum number of clusters a controller may own with `--set-max-clusters-capacity` (CLI) or through the GUI. This is only available once the controller pool is enabled. See *Capacity and cold-start hardening* for what the `-2`/`-1`/`0`/`>0` values mean.

### ClusterControl GUI

1. In the left-side navigation bar, click **Controller pool**.
2. Choose a controller and click its ellipsis button (**…**).
3. Choose **Set capacity**.
4. In the **Max clusters** field, enter the maximum number of clusters this controller may own. Setting the capacity lower than the number of clusters currently owned causes the controller to abandon the excess clusters. To move clusters back onto this controller later, raise the capacity again.
5. Click **Apply**.

### ClusterControl CLI

```
s9s pool-controllers \
  --set-max-clusters-capacity=40 \
  --controller="https://HOST:9501" \
  --cmon-user=USER \
  --password=PASS
```

- `--set-max-clusters-capacity=40` sets this controller's capacity to 40 clusters.
- `--controller="https://HOST:9501"` targets the controller to change.

- `--cmon-user` and `--password` are your ClusterControl credentials. Omit them if they are already set in your `S9S_USER_CONFIG` file.
- Add `--force` if lowering the capacity would require the controller to abandon clusters it already owns.

### Starting or stopping a controller

Starting and stopping a controller works the same way as starting or stopping the `cmon` daemon via systemd. You can only start a stopped controller, and only stop a running one.

### ClusterControl GUI

1. In the left-side navigation bar, click **Controller pool**.
2. Choose a controller and click its ellipsis button (**…**).
3. Choose **Stop** or **Start**.
4. Confirm the action in the prompt that appears.

### ClusterControl CLI

```
s9s pool-controllers \
  --start \
  --controller-id SID
```

- `--start` starts the controller's `cmon` daemon. Use `--stop` to stop a running controller instead.
- `--controller-id SID` is the controller's `SID`, not its runtime `ID`. See *View the controller pool* for the difference.

**Note**

Note A controller with the `main` role cannot be stopped, only controllers with the `member` role.

### Removing a controller from the pool

Removing a controller applies only to a controller that is already stopped.

### ClusterControl GUI

1. In the left-side navigation bar, click **Controller pool**.
2. Choose a controller and click its ellipsis button (**…**).
3. Choose **Remove**.
4. In the prompt, choose one of two options:
   - **Unregister controller only**: Unregisters the controller from the pool but leaves the `cmon` daemon and its installation untouched.
   - **Unregister and uninstall controller**: Unregisters the controller and uninstalls its packages, ensuring the daemon is fully stopped.

#### ClusterControl CLI

```
s9s pool-controllers \
  --remove-controller \
  --controller-id SID
```

- `--remove-controller` removes the controller from the pool.
- `--controller-id SID` is the controller's `SID`, not its runtime `ID`. See *View the controller pool* for the difference.

Add `--uninstall` to also remove the `cmon` packages from that host:

```
s9s pool-controllers --remove-controller --controller-id SID --uninstall
```

### Upgrading a controller's cmon version

#### ClusterControl GUI

1. In the left-side navigation bar, click **Controller pool**.
2. Choose a controller and click its ellipsis button (**…**).
3. Choose **Upgrade**.
4. Confirm the action in the prompt that appears.

#### ClusterControl CLI

The controller must be stopped before it can be upgraded.

```
s9s pool-controllers \
  --update-cmon \
  --controller-id SID
```

- `--update-cmon` upgrades the controller to the latest `cmon` version available in the Severalnines public repository.
- `--controller-id SID` is the controller's `SID`, not its runtime `ID`. See *View the controller pool* for the difference.

This also works against a specific pool member when run from the main controller.

## 3.9.10 Inspecting the controller pool with the CLI

Beyond the management operations above, the `s9s` CLI can also inspect pool status and ownership through related `cluster` and `node` commands.

### View the controller pool

```
$ s9s pool-controllers --list
```

This shows controller SID, ID, hostname, port, status, role, and assigned clusters.

```
$ s9s pool-controllers --list
SID ID HOSTNAME      PORT STATUS ROLE   COUNT/MAX CLUSTERS
13  14 192.168.40.10 9500 active member      2/36 [2, 9]
7   24 192.168.40.5  9500 active main        2/36 [5, 10]
11  31 192.168.40.3  9500 active member      1/38 [1]
9   32 192.168.40.4  9500 active member      1/38 [8]
```

The **ROLE** field identifies which controller is your **main** controller, host `192.168.40.5` in this example.

The columns show:

- **SID**: **Static controller ID** (persistent config identifier). Use this for operations like `--start`, `--stop`, `--remove-controller`.
- **ID**: **Runtime/dynamic controller instance ID** (can change with runtime/re-registration).
- **HOSTNAME**: Controller host/IP.
- **PORT**: Controller RPC/API port (typically `9500`/`9501` depending setup).
- **STATUS**: Current state (e.g., `active`, `starting`, `stopped`, `restarting`, etc.).
- **ROLE**: Pool role (`main` = primary config/db holder, `member` = regular pool member).
- **COUNT/MAX**: `owned_clusters / max_cluster_capacity` for that controller (e.g., `2/36` means owns 2 clusters, cap is 36).
- **CLUSTERS**: Cluster IDs currently assigned to that controller (e.g., `[2, 9]`).

In the example above, the controller at `192.168.40.10` owns clusters `2` and `9`, out of a capacity of `36`.

### Identifying ownership from the cluster perspective

Combine the result of `s9s pool-controllers --list` with `s9s cluster --list` to see which cluster names and details belong to a given controller:

```
$ s9s cluster --list --long --controller="https://192.168.40.10:9501" --
↪cmon-user=admin2 --password="userP@55" --rpc-tls
```

Result displays same as the following example:

```
ID STATE   TYPE              OWNER  GROUP  NAME           COMMENT
 2 STARTED postgresql_single admin2 admins pgsql-18       All nodes are␣
↪operational.
 9 STARTED galera            admin2 admins mariadb-galera All nodes are␣
↪operational.
Total: 2
```

For security purposes, and you do not want to expose the password, you can use the following:

```
$ export S9S_USER_CONFIG=/root/.s9s/s9s-controller10.conf;  s9s cluster --
↪list --long --controller="https://192.168.40.10:9501" --rpc-tls
ID STATE   TYPE              OWNER  GROUP  NAME           COMMENT
 2 STARTED postgresql_single admin2 admins pgsql-18       All nodes are␣
↪operational.
 9 STARTED galera            admin2 admins mariadb-galera All nodes are␣
```

(continues on next page)

(continued from previous page)

```
↪operational.
Total: 2
```

Where the file `/root/.s9s/s9s-controller10.conf` contains the following contents:

```
[global]
controller    = https://192.168.40.10:9501
cmon_user     = "admin"
cmon_password = "7201231b-8f0a-4d5e-a755-991e32ce3f66"
```

Note that querying controller `192.168.40.10` with `s9s cluster --list --long` only returns clusters `2` and `9`, the clusters it owns. A cluster can only be viewed, listed, or identified through the controller that currently owns it, whether that controller is specified with `--controller` or through the `controller` parameter in `S9S_USER_CONFIG`.

### Listing the nodes of the cluster

To list the nodes owned by a specific controller, run `s9s node --list --long` against that controller's address.

For example, from the main controller `192.168.40.5`, you can check what nodes each pool member has:

```
$ s9s node --list --long --controller="https://192.168.40.3:9501"

STAT VERSION     CID CLUSTER        HOST            PORT COMMENT
ho-- 1.8.27        1  mariadb-10.5  192.168.121.22 9600 Process 'haproxy'␣
↪is running.
coC- 2.5.0.22113   1  mariadb-10.5  192.168.40.3   9500 Up and running.
Po-- 2.53.5        1  mariadb-10.5  192.168.40.4   9090 Process 'prometheus
↪' is running.
soM- 10.5.29       1  mariadb-10.5  192.168.40.60  3306 Up and running␣
↪(read-write).
koM- 2.1           1  mariadb-10.5  192.168.40.60   112 Process 'keepalived
↪' is running.
soS- 10.5.29       1  mariadb-10.5  192.168.40.61  3306 Up and running␣
↪(read-only).
ho-- 1.8.27        1  mariadb-10.5  192.168.40.61  9600 Process 'haproxy'␣
↪is running.
ko-- 2.1           1  mariadb-10.5  192.168.40.61   112 Process 'keepalived
↪' is running.
Total: 8
```

Or check what nodes and clusters (by CID) controller host `192.168.40.4` has:

```
$ s9s node --list --long --controller="https://192.168.40.4:9501"
STAT VERSION     CID CLUSTER  HOST          PORT COMMENT
coC- 2.5.0.22113   8 pgsql-14 192.168.40.4  9500 Up and running.
poM- 14.23         8 pgsql-14 192.168.40.62 5432 Up and running␣
↪(read-write).
poS- 14.23         8 pgsql-14 192.168.40.66 5432 Up and running (read-only).
Total: 3
```

Or check what nodes and clusters (by CID) controller host `192.168.40.10` has:

```
$ s9s node --list --long --controller="https://192.168.40.10:9501"
STAT VERSION     CID CLUSTER        HOST          PORT COMMENT
coC- 2.5.0.22113   2 pgsql-18       192.168.40.10 9500 Up and running.
Po-- 2.53.5        2 pgsql-18       192.168.40.3  9090 Process 'prometheus'␣
↪is running.
p-S- 18.4          2 pgsql-18       192.168.40.63 5432 Shut down␣
↪(read-only).
ho-- 1.8.27        2 pgsql-18       192.168.40.63 9600 Process 'haproxy' is␣
↪running.
koM- 2.1           2 pgsql-18       192.168.40.63  112 Process 'keepalived'␣
↪is running.
poM- 18.4          2 pgsql-18       192.168.40.64 5432 Up and running␣
↪(read-write).
ho-- 1.8.27        2 pgsql-18       192.168.40.64 9600 Process 'haproxy' is␣
↪running.
ko-- 2.1           2 pgsql-18       192.168.40.64  112 Process 'keepalived'␣
↪is running.
coC- 2.5.0.22113   9 mariadb-galera 192.168.40.10 9500 Up and running.
Po-- 2.53.5        9 mariadb-galera 192.168.40.10 9090 Process 'prometheus'␣
↪is running.
goM- 11.8.8        9 mariadb-galera 192.168.40.33 3306 Up and running␣
↪(read-write).
goM- 11.8.8        9 mariadb-galera 192.168.40.34 3306 Up and running␣
↪(read-write).
goM- 11.8.8        9 mariadb-galera 192.168.40.35 3306 Up and running␣
↪(read-write).
Total: 13
```

Additionally, to see hostname/IP, node type, and database version for every node across all controllers in a single table, run the following from the main controller:

```
$ s9s cluster --list --all-pool --long --print-json | jq -r '
 ␣
↪["cluster","cdt_path","ip","hostname","nodetype","version","version_comment","addresses
↪as $cols
  | ($cols | "| " + join(" | ") + " |"),
    ($cols | map("---") | "| " + join(" | ") + " |"),
    ( [.. | objects | select(.class_name? == "CmonClusterInfo")][]
      | (.cluster_name // .cdt_path) as $cluster
      | .hosts[]?
      | [ $cluster,
          ((.cdt_path // "-") | gsub("^\\s+|\\s+$"; "")),
          (.ip // "-"),
          (.hostname // "-"),
          (.nodetype // .class_name // "-"),
          (.version // "-"),
          (.version_comment // "-"),
          ([.configuration[]?.address] | join("<br>") | if . == "" then "-"␣
↪else . end)
```

(continues on next page)

(continued from previous page)

```
        ]
      | map(tostring | gsub("\\|"; "\\|"))
      | "| " + join(" | ") + " |"
    )'
| cluster | cdt_path | ip | hostname | nodetype | version | version_comment |
| addresses |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.40.60 | 192.168.40.60 | mysql | 10.5.29-MariaDB-log | MariaDB Server | - |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.40.3 | 192.168.40.3 | controller | 2.5.0.22113 | - | - |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.40.4 | 192.168.40.4 | prometheus | 2.53.5 | - | <br><br><br><br> |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.40.61 | 192.168.40.61 | mysql | 10.5.29-MariaDB-log | MariaDB Server | - |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.40.61 | 192.168.40.61 | haproxy | 1.8.27 | - | - |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.40.60 | 192.168.40.60 | keepalived | 2.1 | - | - |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.40.61 | 192.168.40.61 | keepalived | 2.1 | - | - |
|  mariadb-10.5  | / mariadb-10.5 | 192.168.121.22 | 192.168.121.22 | haproxy | 1.8.27 | - | - |
| pgsql-18 | /pgsql-18 | 192.168.40.63 | 192.168.40.63 | postgres | 18.4 | - | - |
| pgsql-18 | /pgsql-18 | 192.168.40.64 | 192.168.40.64 | postgres | 18.4 | - | - |
| pgsql-18 | /pgsql-18 | 192.168.40.10 | 192.168.40.10 | controller | 2.5.0.22113 | - | - |
| pgsql-18 | /pgsql-18 | 192.168.40.3 | 192.168.40.3 | prometheus | 2.53.5 | - | <br><br><br><br> |
| pgsql-18 | /pgsql-18 | 192.168.40.63 | 192.168.40.63 | haproxy | 1.8.27 | - | - |
| pgsql-18 | /pgsql-18 | 192.168.40.64 | 192.168.40.64 | haproxy | 1.8.27 | - | - |
| pgsql-18 | /pgsql-18 | 192.168.40.63 | 192.168.40.63 | keepalived | 2.1 | - | - |
| pgsql-18 | /pgsql-18 | 192.168.40.64 | 192.168.40.64 | keepalived | 2.1 | - | - |
| clickhouse | /clickhouse | 192.168.40.67 | 192.168.40.67 | clickhouse | 25.3 | - | - |
| clickhouse | /clickhouse | 192.168.40.68 | 192.168.40.68 | clickhouse | 25.3 | - | - |
| clickhouse | /clickhouse | 192.168.40.69 | 192.168.40.69 | clickhouse | 25.3 | - | - |
| clickhouse | /clickhouse | 192.168.40.30 | 192.168.40.30 | clickhouse_keeper | 25.3 | - | - |
| clickhouse | /clickhouse | 192.168.40.31 | 192.168.40.31 | clickhouse_keeper | 25.3 | - | - |
```

(continues on next page)

(continued from previous page)

```
| clickhouse | /clickhouse | 192.168.40.32 | 192.168.40.32 |␣
↪clickhouse_keeper | 25.3 | - | - |
| clickhouse | /clickhouse | 192.168.40.5 | 192.168.40.5 | controller | 2.
↪5.0.22113 | - | - |
| clickhouse | /clickhouse | 192.168.40.10 | 192.168.40.10 | prometheus | 2.
↪53.5 | - | <br><br><br><br> |
| pgsql-14 | /pgsql-14. | 192.168.40.66 | 192.168.40.66 | postgres | 14.23␣
↪| - | - |
| pgsql-14 | /pgsql-14. | 192.168.40.62 | 192.168.40.62 | postgres | 14.23␣
↪| - | - |
| pgsql-14 | /pgsql-14. | 192.168.40.4 | 192.168.40.4 | controller | 2.
↪5.0.22113 | - | - |
| mariadb-galera | /mariadb-galera | 192.168.40.33 | 192.168.40.33 | galera␣
↪| 11.8.8-MariaDB-log | MariaDB Server | - |
| mariadb-galera | /mariadb-galera | 192.168.40.10 | 192.168.40.10 |␣
↪controller | 2.5.0.22113 | - | - |
| mariadb-galera | /mariadb-galera | 192.168.40.34 | 192.168.40.34 | galera␣
↪| 11.8.8-MariaDB-log | MariaDB Server | - |
| mariadb-galera | /mariadb-galera | 192.168.40.35 | 192.168.40.35 | galera␣
↪| 11.8.8-MariaDB-log | MariaDB Server | - |
| mariadb-galera | /mariadb-galera | 192.168.40.10 | 192.168.40.10 |␣
↪prometheus | 2.53.5 | - | <br><br><br> |
```

### 3.9.11 Operational guidance

#### Recommended workflow

1. Set up a shared CMON database.
2. Choose a shared configuration backend appropriate to the environment.
3. Start the initial controller with pool mode enabled.
4. Add additional controllers that point to the same CMON database and shared config backend.
5. Use `s9s pool-controllers --list` and `s9s cluster --list --all-pool` to verify ownership distribution.
6. For larger fleets, tune capacity and claim-step settings to avoid cold-start overload.

### 3.9.12 Common gotchas

- Assuming Kubernetes is required: it is not. Kubernetes is one supported model, but VMs, bare metal, and cloud-hosted servers are also valid if shared DB and shared config are in place.
- Missing shared config: pool mode requires all controllers to see the same configuration state.
- Wrong HA expectations: pool mode gives you horizontal scaling of controller capacity, but full HA/failover readiness has not shipped in the same release as pool mode for every version. Check your version's release notes before assuming both are available.
- Privilege errors on pool-wide listing: use a superuser/admin account.
- Runtime capacity changes disappearing after restart: only configuration-file values are persistent.

## 3.10 ClusterControl Replication

ClusterControl Replication provides a high-availability solution for ClusterControl using replication with open-source tools. This deployment uses a primary-secondary architecture where only one ClusterControl instance is active at a time, with the other on cold standby.

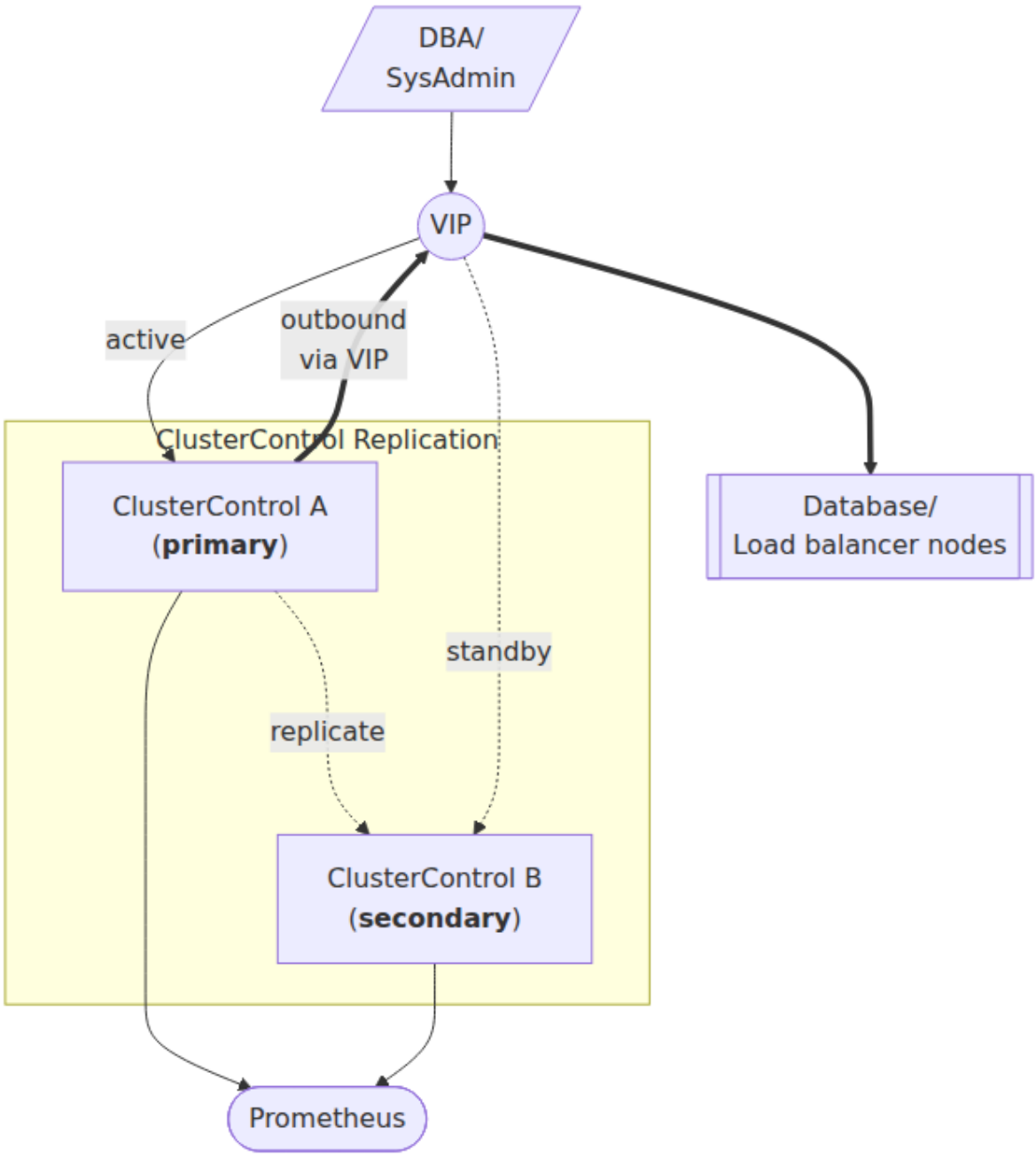
### 3.10.1 Key Components

- **MySQL Replication** - Provides CMON database redundancy using primary-replica topology
- **File Syncing** - Uses rsync and lsyncd for one-way synchronization of ClusterControl files
- **Virtual IP (VIP)** - Managed by Keepalived for seamless client connectivity during failover, and provides a sticky outbound IP address after a failover
- **External Prometheus** - Must be deployed on a separate host (not co-located with ClusterControl)

### 3.10.2 Architecture

The active node holding the VIP is:

- The primary ClusterControl server
- The primary MySQL instance with read-write access
- The source for file syncing operations
- Configured to use VIP for outbound connections




### 3.10.3 Known limitations

Please understand the following limitations and restrictions with this setup:

1. Prometheus is a single point of failure. You can achieve redundancy for this component by implementing remote write to another Prometheus server. This is out of the scope of this article.
2. The managed cluster database backups should NOT be stored on the ClusterControl server. You have other options like store backup on the database node, or upload backups to cloud (optionally choose "Delete after upload"). It is also possible to store the backups on ClusterControl node using shared storage (NFS/CIFS/SMB) mounted on the same path on both cc1 and cc2.
3. This setup relies on virtual IP address, provided by Keepalived which only works on Layer 2 broadcast domain, recommended for LAN. It can be used across WAN only if L2 connectivity is extended (e.g., VPLS or bridging), though it is not recommended for high-latency connections.

### 3.10.4 Requirements

The following requirements must be met to deploy ClusterControl Replication:

- 3 nodes minimum:
  - ClusterControl A (cc1) - Primary node
  - ClusterControl B (cc2) - Secondary node
  - Prometheus (prom) - External monitoring node
- Virtual IP address (e.g., `192.168.73.200`)
- MySQL 8+ with CLONE plugin enabled
- VRRP support between ClusterControl nodes (Layer 2 connectivity)
- Shared SSH private key between nodes

### 3.10.5 Deployment steps

This guide uses the following example configuration:

- **VIP**: 192.168.73.200
- **cc1** (primary): 192.168.73.201
- **cc2** (secondary): 192.168.73.202
- **prom** (Prometheus): 192.168.73.209

**Info**

Adapt all IP addresses, hostnames, and interface names to match your environment.

#### Ubuntu 24.04

1. **Host Preparation (All Nodes)**

   1.1. Configure `/etc/hosts` on all nodes (cc1, cc2, and prom):

```
tee -a /etc/hosts <<EOF
192.168.73.200      ccvip.local ccvip
192.168.73.201      cc1.local cc1
192.168.73.202      cc2.local cc2
192.168.73.209      prom.local prom
EOF
```

   1.2. Generate SSH keys on **cc1**:

```
# As root
ssh-keygen -t rsa    # press Enter on all prompts
```

   1.3. Prepare root SSH access on all nodes (**cc2** and **prom**):

```
mkdir -p /root/.ssh
chmod 700 /root/.ssh
vi /root/.ssh/id_rsa    # copy the content of /root/.ssh/id_rsa on cc1␣
↪here
chmod 600 /root/.ssh/id_rsa
```

1.4. Copy the public key to all nodes:

```
# Get the public key
cat /root/.ssh/id_rsa.pub

# On each node (cc1, cc2, prom), add it to authorized_keys
tee -a /root/.ssh/authorized_keys <<EOF
# paste public key here
EOF
```

1.5. Configure SSH to allow root login with key:

```
sed -i 's/^#PermitRootLogin.*/PermitRootLogin prohibit-password/' /
↪etc/ssh/sshd_config
systemctl restart sshd
```

Verify passwordless SSH from **cc1** to all nodes:

```
ssh root@cc1 hostname
ssh root@cc2 hostname
ssh root@prom hostname
```

2. **Keepalived Installation** (**cc1** and **cc2**)

Keepalived provides a virtual IP service and it will be used as the single-endpoint access for ClusterControl services. The ClusterControl GUI will be accessible via VIP, for example: **https://192.168.73.200/**.

2.1. Install Keepalived and iptables on both ClusterControl nodes:

```
apt update
apt install -y keepalived iptables
```

2.2. Enable non-local IP binding:

```
echo "net.ipv4.ip_nonlocal_bind = 1" | tee -a /etc/sysctl.conf
sysctl -p
```

2.3. Create the SNAT management script `/etc/keepalived/manage_snat.sh` on both **cc1** and **cc2** to force outbound connections to use the VIP:

```
tee /etc/keepalived/manage_snat.sh <<'EOF'
#!/bin/bash

## Change me
VIP='192.168.73.200'
```

(continues on next page)

(continued from previous page)

```
INTERFACE='eth0'
##

TYPE=$1
NAME=$2
STATE=$3

case $STATE in
    "MASTER")
        iptables -t nat -A POSTROUTING -o $INTERFACE -j SNAT --
↪to-source $VIP
        ;;
    "BACKUP"|"FAULT")
        iptables -t nat -D POSTROUTING -o $INTERFACE -j SNAT --
↪to-source $VIP
        ;;
esac
EOF
```

**Warning**

Attention

```
Adjust `INTERFACE='eth0'` and `VIP='192.168.73.200'` to match your␣
↪network interface and virtual IP address. Check with `ip a` or␣
↪`ifconfig`.
```

2.4. Make the script executable:

```
chmod +x /etc/keepalived/manage_snat.sh
```

2.5. Configure Keepalived on **cc1** (priority 105):

```
tee /etc/keepalived/keepalived.conf <<'EOF'
vrrp_script check_cmon {
script "/usr/bin/killall -0 cmon"
interval 2
weight 2
}
vrrp_instance VI_1 {
state BACKUP
nopreempt
interface eth0
virtual_router_id 51
priority 105
advert_int 1
authentication {
    auth_type PASS
    auth_pass 1113
```

(continues on next page)

(continued from previous page)

```
}
virtual_ipaddress {
    192.168.73.200   # the VIP
}
track_script {
    check_cmon
}
notify /etc/keepalived/manage_snat.sh
}
EOF
```

2.6. Configure Keepalived on **cc2** (priority 100):

```
tee /etc/keepalived/keepalived.conf <<'EOF'
vrrp_script check_cmon {
script "/usr/bin/killall -0 cmon"
interval 2
weight 2
}
vrrp_instance VI_1 {
state BACKUP
nopreempt
interface eth0
virtual_router_id 51
priority 100
advert_int 1
authentication {
    auth_type PASS
    auth_pass 1113
}
virtual_ipaddress {
    192.168.73.200   # the VIP
}
track_script {
    check_cmon
}
notify /etc/keepalived/manage_snat.sh
}
EOF
```

**Warning**

Attention

```
Change the `interface` and `virtual_ipaddress` values accordingly.
```

2.7. Ensure Keepalived is stopped on **cc1** and **cc2** and we will start Keepalived later, after we have configured file replication:

```
systemctl stop keepalived
```

3. **Install ClusterControl**

3.1. As root, install ClusterControl on **cc1**:

```
wget https://severalnines.com/downloads/cmon/install-cc
chmod +x install-cc
./install-cc    # follow the installation wizard
```

3.2. Stop all ClusterControl services and MySQL, and disable cmon on boot:

```
systemctl stop cmon cmon-cloud cmon-ssh cmon-events cmon-proxy␣
↪kuber-proxy
systemctl stop mysql
systemctl disable cmon
```

Repeat steps 3.1 and 3.2 on **cc2**. Make sure you stop ClusterControl services and MySQL.

3.3. On **cc1** and **cc2**, set the VIP as the hostname value inside `/etc/cmon.cnf`:

```
hostname=192.168.73.200
```

3.4. On **cc1** and **cc2**, add the following line to define the Prometheus host 192.168.73.209 inside `/etc/cmon.cnf`:

```
prometheus_hostname=192.168.73.209
```

3.5. On **cc1** and **cc2**, configure `/etc/default/cmon` as below to allow ClusterControl RPC interface to listen to all IP addresses (including VIP):

```
# New events client http callback as of v1.4.2!
EVENTS_CLIENT="http://127.0.0.1:9510"
CLOUD_SERVICE="http://127.0.0.1:9518"
RPC_BIND_ADDRESSES="0.0.0.0,127.0.0.1"
```

4. **Configure MySQL replication for CMON DB**

4.1. Stop MySQL service on both **cc1** and **cc2** if you haven't done so:

```
systemctl stop mysql
```

4.2. On both **cc1** and **cc2**, comment the following line inside `/etc/mysql/my.cnf` (because we want to enable binlog for replication):

```
#skip-log-bin
```

4.3. On **cc1**, configure MySQL configuration file, `/etc/mysql/my.cnf` as below under the `[mysqld]` section:

```
# add the following lines if not exist
server-id               = 1
log_bin                 = mysql-bin
binlog_format           = ROW
gtid_mode               = ON
enforce_gtid_consistency= ON
log_slave_updates       = ON
binlog_expire_logs_seconds = 604800
bind-address            = 0.0.0.0
# CLONE plugin
plugin-load-add         = mysql_clone.so
clone                   = FORCE_PLUS_PERMANENT
clone_valid_donor_list  = 'cc1:3306,cc2:3306'
```

4.4. On **cc2**, configure MySQL configuration file, `/etc/mysql/my.cnf` as below under the `[mysqld]` section (the difference is `server-id`):

```
# add the following lines if not exist
server-id               = 2
log_bin                 = mysql-bin
binlog_format           = ROW
gtid_mode               = ON
enforce_gtid_consistency= ON
log_slave_updates       = ON
binlog_expire_logs_seconds = 604800
bind-address            = 0.0.0.0
# CLONE plugin
plugin-load-add         = mysql_clone.so
clone                   = FORCE_PLUS_PERMANENT
clone_valid_donor_list  = 'cc1:3306,cc2:3306'
```

4.5. Start MySQL on **cc1** and **cc2**, make sure MySQL is operational after restart:

```
systemctl start mysql
systemctl status mysql
```

4.6. Configure MySQL on **cc1** as primary:

```
-- mysql on cc1
CREATE USER 'clone'@'%' IDENTIFIED BY 'Secur3_p4ssword';
GRANT BACKUP_ADMIN, CLONE_ADMIN ON *.* TO 'clone'@'%';
-- for repl user, we use mysql_native_password plugin to simplify␣
↪replication configuration
CREATE USER 'repl'@'%' IDENTIFIED WITH 'mysql_native_password' BY
↪'Secur3_p4ssword';
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'repl'@'%';
```

4.7. On **cc2**, initiate the clone to sync MySQL data from **cc1**:

```
-- mysql on cc2
CLONE INSTANCE FROM 'clone'@'cc1':3306 IDENTIFIED BY 'Secur3_p4ssword';
```

4.8. On **cc2**, start the MySQL replication and check the status:

```
-- mysql on cc2
CHANGE REPLICATION SOURCE TO
SOURCE_HOST = 'cc1',
SOURCE_USER = 'repl',
SOURCE_PASSWORD = 'Secur3_p4ssword',
SOURCE_AUTO_POSITION = 1;
```

4.9. Start the replication and check the replication status:

```
-- mysql on cc2
START REPLICA;
SHOW REPLICA STATUS\G
```

Make sure you see the following:

```
    Replica_IO_Running: Yes
    Replica_SQL_Running: Yes
```

The above means **cc2** MySQL is actively replicating from **cc1**.

4.9. On **cc2**, configure MySQL to be a read-only replica to protect against accidental writes:

```
-- mysql on cc2
SET GLOBAL read_only = 1;
SET GLOBAL super_read_only = 1;
```

5. **Configure file syncing**

5.1. Install rsync and lsyncd on **cc1** and **cc2**:

```
apt install -y rsync lsyncd
```

5.2. Create the lsyncd configuration directory:

```
mkdir /etc/lsyncd
```

5.3. Add the following line into `/etc/lsyncd/lsyncd.conf.lua` on **cc1**:

```
settings {
    logfile = "/var/log/lsyncd.log",
    statusFile = "/var/log/lsyncd.status",
    statusInterval = 10
}

-- 1. Sync for directories (Add trailing slashes to source)
local sync_dirs = {
```

(continues on next page)

(continued from previous page)

```
    "/var/lib/cmon/",
    "/etc/cmon.d/",
    "/root/.s9s/",
    "/var/www/html/clustercontrol-mcc/",
    "/var/www/html/cmon-repos/",
    "/usr/share/ccmgr/"
}

for _, path in ipairs(sync_dirs) do
    sync {
        default.rsync,
        source = path, -- The trailing slash here means "sync contents"
        target = "cc2:" .. path,
        rsync = {
            archive = true,
            compress = true
        }
    }
end

-- 2. Sync for the individual file /etc/s9s.conf
sync {
    default.rsync,
    source = "/etc/",
    target = "cc2:/etc/",
    rsync = {
        archive = true,
        _extra = {
            "--include=/cmon.cnf",
            "--include=/cmon-events.cnf",
            "--include=/s9s.cnf",
            "--exclude=*"
        }
    }
}
```

5.4. Add the following line into `/etc/lsyncd/lsyncd.conf.lua` on **cc2**:

```
settings {
    logfile = "/var/log/lsyncd.log",
    statusFile = "/var/log/lsyncd.status",
    statusInterval = 10
}

-- 1. Sync for directories (Add trailing slashes to source)
local sync_dirs = {
    "/var/lib/cmon/",
    "/etc/cmon.d/",
    "/root/.s9s/",
    "/var/www/html/clustercontrol-mcc/",
```

(continues on next page)

(continued from previous page)

```
    "/var/www/html/cmon-repos/",
    "/usr/share/ccmgr/"
}

for _, path in ipairs(sync_dirs) do
    sync {
        default.rsync,
        source = path, -- The trailing slash here means "sync contents"
        target = "cc1:" .. path,
        rsync = {
            archive = true,
            compress = true
        }
    }
end

-- 2. Sync for the individual file /etc/s9s.conf
sync {
    default.rsync,
    source = "/etc/",
    target = "cc1:/etc/",
    rsync = {
        archive = true,
        _extra = {
            "--include=/cmon.cnf",
            "--include=/cmon-events.cnf",
            "--include=/s9s.cnf",
            "--exclude=*"
        }
    }
}
```

5.5. Start Keepalived service on **cc1** only, so VIP will be available on **cc1** before we can start ClusterControl services:

```
systemctl start keepalived
systemctl enable keepalived
```

5.6. Start ClusterControl services on **cc1** only:

```
systemctl start cmon cmon-cloud cmon-ssh cmon-events cmon-proxy␣
↪kuber-proxy
```

5.7. Ping ClusterControl RPC interface for `cid 0` on **cc1** only, make sure you get the "Ok" reply:

```
$ s9s cluster --ping --cluster-id=0
PING Ok   4 ms
```

5.8. Start the lsyncd service on **cc1** only and make sure it is running (DO NOT START it on **cc2**):

```
systemctl start lsyncd
systemctl status lsyncd
```

5.9. We will intentionally set lsyncd to NOT start on boot for both **cc1** and **cc2**:

```
systemctl disable lsyncd
```

5.10. Monitor the file syncing operation. Make sure you see something similar to below:

```
$ tail -f /var/log/lsyncd.log
Thu Jan 22 17:45:02 2026 Normal: recursive startup rsync: /
↪var/www/html/cmon-repos/ -> cc2:/var/www/html/cmon-repos/
Thu Jan 22 17:45:02 2026 Normal: recursive startup rsync: /
↪usr/share/ccmgr/ -> cc2:/usr/share/ccmgr/
Thu Jan 22 17:45:02 2026 Normal: recursive startup rsync: /etc/ ->␣
↪cc2:/etc/
Thu Jan 22 17:45:03 2026 Normal: Startup of /
↪var/www/html/clustercontrol-mcc/ ->␣
↪cc2:/var/www/html/clustercontrol-mcc/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /etc/ -> cc2:/etc/␣
↪finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /var/lib/cmon/ ->␣
↪cc2:/var/lib/cmon/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /usr/share/ccmgr/ ->␣
↪cc2:/usr/share/ccmgr/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /etc/cmon.d/ ->␣
↪cc2:/etc/cmon.d/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /root/.s9s/ ->␣
↪cc2:/root/.s9s/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /var/www/html/cmon-repos/ -
↪> cc2:/var/www/html/cmon-repos/ finished.
```

5.11. On **cc2**, enable keepalived on boot and start it up:

```
systemctl enable keepalived
systemctl stop keepalived
systemctl start keepalived
```

5.12. On **cc2**, you should see it is entering the `BACKUP` state and you don't see the VIP there:

```
$ grep Keepalived /var/log/syslog
2026-01-22T16:42:24.652694+00:00 cc1 Keepalived[720]: Starting VRRP␣
↪child process, pid=730
2026-01-22T16:42:24.652701+00:00 cc1 Keepalived[720]: Startup complete
2026-01-22T16:42:24.652722+00:00 cc1 Keepalived_vrrp[730]: (VI_1)␣
↪Entering BACKUP STATE (init)
```

5.13. Verify that only **cc1** has the VIP:

**cc1:**

```
$ ip a | grep global  # on cc1
inet 192.168.73.201/24 brd 192.168.73.255 scope global eth0
inet 192.168.73.200/32 scope global eth0
```

**cc2:**

```
$ ip a | grep global  # on cc2
inet 192.168.73.202/24 brd 192.168.73.255 scope global eth0
```

5.14. Verify that we have the `POSTROUTING` chain configured, to force VIP as the outbound address on **cc1**:

```
$ iptables -t nat -L POSTROUTING -n --line-numbers
Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    SNAT       all  --  0.0.0.0/0            0.0.0.0/0           ␣
↪to:192.168.73.200
```

At this moment, ClusterControl on **cc1** is the primary and **cc2** is the secondary (replica).

### Rocky Linux 9

1. **Host Preparation (All Nodes)**

1.1. Configure `/etc/hosts` on all nodes (cc1, cc2, and prom):

```
tee -a /etc/hosts <<EOF
192.168.73.200      ccvip.local ccvip
192.168.73.201      cc1.local cc1
192.168.73.202      cc2.local cc2
192.168.73.209      prom.local prom
EOF
```

1.2. Generate SSH keys on **cc1**:

```
# As root
ssh-keygen -t rsa -b 4096   # press Enter on all prompts
```

1.3. Prepare root SSH access on all nodes (**cc2** and **prom**):

```
mkdir -p /root/.ssh
chmod 700 /root/.ssh
vi /root/.ssh/id_rsa    # copy the content of /root/.ssh/id_rsa on cc1␣
↪here
chmod 600 /root/.ssh/id_rsa
```

1.4. Copy the public key to all nodes:

```
# Get the public key
cat /root/.ssh/id_rsa.pub

# On each node (cc1, cc2, prom), add to authorized_keys
```

(continues on next page)

(continued from previous page)

```
tee -a /root/.ssh/authorized_keys <<EOF
# paste public key here
EOF
```

1.5. Configure SSH to allow root login with key:

```
sed -i 's/^#PermitRootLogin.*/PermitRootLogin prohibit-password/' /
↪etc/ssh/sshd_config
systemctl restart sshd
```

1.6. Verify passwordless SSH from **cc1** to all nodes:

```
ssh root@cc1 hostname
ssh root@cc2 hostname
ssh root@prom hostname
```

1.7. Disable firewalld to simplify deployment (or configure appropriate rules):

```
systemctl stop firewalld
systemctl disable firewalld
```

2. **Keepalived Installation** (**cc1** and **cc2**)

Keepalived provides a virtual IP service and it will be used as the single-endpoint access for ClusterControl services. The ClusterControl GUI will be accessible via VIP, for example: `https://192.168.73.200/`.

2.1. Install Keepalived and iptables on both ClusterControl nodes:

```
dnf install -y keepalived iptables
```

2.2. Enable non-local IP binding:

```
echo "net.ipv4.ip_nonlocal_bind = 1" | tee -a /etc/sysctl.conf
sysctl -p
```

2.3. Create the SNAT management script `/etc/keepalived/manage_snat.sh` on both **cc1** and **cc2** to force outbound connections to use the VIP:

```
tee /etc/keepalived/manage_snat.sh <<'EOF'
#!/bin/bash

## Change me
VIP='192.168.73.200'
INTERFACE='eth0'
##

TYPE=$1
NAME=$2
STATE=$3

case $STATE in
```

(continues on next page)

(continued from previous page)

```
    "MASTER")
        # Add the SNAT rule when this node takes the VIP
        iptables -t nat -A POSTROUTING -o $INTERFACE -j SNAT --
↪to-source $VIP
        ;;
    "BACKUP"|"FAULT")
        # Remove the SNAT rule when this node loses the VIP
        iptables -t nat -D POSTROUTING -o $INTERFACE -j SNAT --
↪to-source $VIP
        ;;
    *)
        exit 1
        ;;
esac
EOF
```

**Warning**

Attention

```
Adjust `INTERFACE='eth0'` and `VIP='192.168.73.200'` to match your␣
↪network interface and virtual IP address. Check with `ip a` or␣
↪`ifconfig`.
```

2.4. Make the script executable:

```
chmod +x /etc/keepalived/manage_snat.sh
```

2.5. Configure Keepalived on **cc1** (priority 105):

```
tee /etc/keepalived/keepalived.conf <<'EOF'
vrrp_script check_cmon {
script "/usr/bin/killall -0 cmon"
interval 2
weight 2
}
vrrp_instance VI_1 {
state BACKUP
nopreempt
interface ens18
virtual_router_id 51
priority 105
advert_int 1
authentication {
    auth_type PASS
    auth_pass 1113
}
virtual_ipaddress {
    192.168.73.200    # the VIP
```

(continues on next page)

(continued from previous page)

```
}
track_script {
    check_cmon
}
# Execute the script on any state change
# Instruct outbound to use VIP
notify /etc/keepalived/manage_snat.sh
}
EOF
```

2.6. Configure Keepalived on **cc2** (priority 100):

```
tee /etc/keepalived/keepalived.conf <<'EOF'
vrrp_script check_cmon {
script "/usr/bin/killall -0 cmon"
interval 2
weight 2
}
vrrp_instance VI_1 {
state BACKUP
nopreempt
interface ens18
virtual_router_id 51
priority 100
advert_int 1
authentication {
    auth_type PASS
    auth_pass 1113
}
virtual_ipaddress {
    192.168.73.200    # the VIP
}
track_script {
    check_cmon
}
# Execute the script on any state change
# Instruct outbound to use VIP
notify /etc/keepalived/manage_snat.sh
}
EOF
```

**Warning**

Attention

```
Change the `interface` and `virtual_ipaddress` values accordingly.
```

2.7. Ensure Keepalived is stopped on **cc1** and **cc2** and we will start Keepalived later, after we have configured file replication:

```
systemctl stop keepalived
```

3. **MySQL Server Setup** (**cc1** and **cc2**)

   To simplify the failover/resyncing process, we will use MySQL 8 which provides the CLONE plugin. Otherwise, by default, ClusterControl would install and use MariaDB 10.5 on Rocky 9.

   We will first install MySQL on **cc1** and **cc2**, and we will tell ClusterControl installer script to use this MySQL server instead.

   3.1. Install MySQL Server on **cc1** and **cc2**:

```
dnf install -y mysql-server
```

   3.2. On **cc1** and **cc2**, start MySQL server for initialization and then stop it:

```
systemctl start mysqld
# wait for 60 seconds to let datadir initialized
systemctl stop mysqld
```

   3.3. On **cc1**, update MySQL configuration file at `/etc/my.cnf.d/mysql-server.cnf` by adding the following lines under the `[mysqld]` section:

```
innodb_buffer_pool_size = 2G       # 50% of server's 4GB RAM
server-id               = 1
log_bin                 = mysql-bin
binlog_format           = ROW
gtid_mode               = ON
enforce_gtid_consistency= ON
log_slave_updates       = ON
binlog_expire_logs_seconds = 604800
bind-address            = 0.0.0.0
# CLONE plugin
plugin-load-add         = mysql_clone.so
clone                   = FORCE_PLUS_PERMANENT
clone_valid_donor_list  = 'cc1:3306,cc2:3306'
```

   3.4. On **cc2**, update MySQL configuration file at `/etc/my.cnf.d/mysql-server.cnf` by adding the following lines under the `[mysqld]` section (the difference is `server-id`):

```
innodb_buffer_pool_size = 2G       # 50% of server's 4GB RAM
server-id               = 2
log_bin                 = mysql-bin
binlog_format           = ROW
gtid_mode               = ON
enforce_gtid_consistency= ON
log_slave_updates       = ON
binlog_expire_logs_seconds = 604800
bind-address            = 0.0.0.0
# CLONE plugin
plugin-load-add         = mysql_clone.so
clone                   = FORCE_PLUS_PERMANENT
clone_valid_donor_list  = 'cc1:3306,cc2:3306'
```

3.5. Start MySQL and run secure installation on both nodes:

```
systemctl enable mysqld
systemctl start mysqld
systemctl status mysqld
mysql_secure_installation
```

Set root password when prompted (e.g., `Secur3_p4ssword`).

**Warning**

Attention

```
If you choose `Yes` for `VALIDATE PASSWORD COMPONENT`, you need to␣
↪set very strong passwords for all MySQL-related accounts in this␣
↪documentation.
```

4. **ClusterControl Installation** (**cc1** and **cc2**)

4.1. As root, install ClusterControl on **cc1**:

```
wget https://severalnines.com/downloads/cmon/install-cc
chmod +x install-cc
./install-cc    # follow the installation wizard
```

4.2. Stop all ClusterControl services and disable cmon on boot:

```
systemctl stop cmon cmon-cloud cmon-ssh cmon-events cmon-proxy␣
↪kuber-proxy
systemctl disable cmon
```

4.3. Repeat steps 4.1 and 4.2 on **cc2**. Make sure you stop ClusterControl services.

4.4. On **cc1**, set the VIP as the hostname value inside `/etc/cmon.cnf`:

```
hostname=192.168.73.200
```

4.5. On **cc1**, add the following line to define our Prometheus host inside `/etc/cmon.cnf`:

```
prometheus_hostname=192.168.73.209
```

4.6. On **cc1** and **cc2**, configure `/etc/default/cmon` as below to allow ClusterControl RPC interface to listen to all IP addresses (including VIP):

```
# New events client http callback as of v1.4.2!
EVENTS_CLIENT="http://127.0.0.1:9510"
CLOUD_SERVICE="http://127.0.0.1:9518"
RPC_BIND_ADDRESSES="0.0.0.0,127.0.0.1"
```

5. **MySQL Replication Configuration**

5.1. Configure MySQL on **cc1** as primary:

```
-- mysql on cc1
CREATE USER 'clone'@'%' IDENTIFIED BY 'Secur3_p4ssword';
GRANT BACKUP_ADMIN, CLONE_ADMIN ON *.* TO 'clone'@'%';
-- for repl user, we use mysql_native_password plugin to simplify␣
↪replication configuration
CREATE USER 'repl'@'%' IDENTIFIED WITH 'mysql_native_password' BY
↪'Secur3_p4ssword';
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'repl'@'%';
```

5.2. On **cc2**, initiate the clone to sync MySQL data from **cc1**:

```
-- mysql on cc2
CLONE INSTANCE FROM 'clone'@'cc1':3306 IDENTIFIED BY 'Secur3_p4ssword';
```

5.3. On **cc2**, start the MySQL replication and check the status:

```
-- mysql on cc2
CHANGE REPLICATION SOURCE TO
    SOURCE_HOST = 'cc1',
    SOURCE_USER = 'repl',
    SOURCE_PASSWORD = 'Secur3_p4ssword',
    SOURCE_AUTO_POSITION = 1;
```

5.4. Start the replication and check the replication status:

```
-- mysql on cc2
START REPLICA;
SHOW REPLICA STATUS\G
```

5.5. Make sure you see the following:

```
    Replica_IO_Running: Yes
Replica_SQL_Running: Yes
```

The above means **cc2** MySQL is actively replicating from **cc1**.

5.6. On **cc2**, configure MySQL to be a read-only replica to protect against accidental writes:

```
-- mysql on cc2
SET GLOBAL read_only = 1;
SET GLOBAL super_read_only = 1;
```

6. **File Synchronization Setup**

6.1. Install rsync and lsyncd on **cc1** and **cc2**:

```
dnf install -y rsync lsyncd
```

6.2. Clear up all existing lines inside `/etc/lsyncd.conf` and add the following lines on **cc1**:

```
settings {
    logfile = "/var/log/lsyncd.log",
    statusFile = "/var/log/lsyncd.status",
```

(continues on next page)

(continued from previous page)

```
    statusInterval = 10,
    insist = true
}

-- 1. Sync for directories (Add trailing slashes to source)
local sync_dirs = {
    "/var/lib/cmon/",
    "/etc/cmon.d/",
    "/home/cloud-user/.s9s/",
    "/var/www/html/clustercontrol-mcc/",
    "/var/www/html/cmon-repos/",
    "/usr/share/ccmgr/"
}

for _, path in ipairs(sync_dirs) do
    sync {
        default.rsync,
        source = path, -- The trailing slash here means "sync contents"
        target = "cloud-user@cc2:" .. path,
        rsync = {
            archive = true,
            compress = true,
            verbose = true,
            rsync_path = "sudo rsync",
            _extra = {"--delete"},
        }
    }
end

-- 2. Sync for the individual files
sync {
    default.rsync,
    source = "/etc/",
    target = "cloud-user@cc2:/etc/",
    rsync = {
        archive = true,
        compress = true,
        verbose = true,
        rsync_path = "sudo rsync",
        _extra = {
            "--include=/cmon.cnf",
            "--include=/cmon-events.cnf",
            "--include=/s9s.cnf",
            "--exclude=*",
            "--delete"
        }
    }
}
```

6.3. Clear up all existing lines inside `/etc/lsyncd.conf` and add the following lines on **cc2**:

```
settings {
    logfile = "/var/log/lsyncd.log",
    statusFile = "/var/log/lsyncd.status",
    statusInterval = 10,
    insist = true
}

-- 1. Sync for directories (Add trailing slashes to source)
local sync_dirs = {
    "/var/lib/cmon/",
    "/etc/cmon.d/",
    "/home/cloud-user/.s9s/",
    "/var/www/html/clustercontrol-mcc/",
    "/var/www/html/cmon-repos/",
    "/usr/share/ccmgr/"
}

for _, path in ipairs(sync_dirs) do
    sync {
        default.rsync,
        source = path, -- The trailing slash here means "sync contents"
        target = "cloud-user@cc1:" .. path,
        rsync = {
            archive = true,
            compress = true,
            verbose = true,
            rsync_path = "sudo rsync",
            _extra = {"--delete"}
        }
    }
end

-- 2. Sync for the individual files
sync {
    default.rsync,
    source = "/etc/",
    target = "cloud-user@cc1:/etc/",
    rsync = {
        archive = true,
        compress = true,
        verbose = true,
        rsync_path = "sudo rsync",
        _extra = {
            "--include=/cmon.cnf",
            "--include=/cmon-events.cnf",
            "--include=/s9s.cnf",
            "--exclude=*",
            "--delete"
        }
    }
```

(continues on next page)

(continued from previous page)

```
}
```

6.4. Start Keepalived service on **cc1** only, so VIP will be available on **cc1** before we can start ClusterControl services:

```
systemctl start keepalived
systemctl enable keepalived
```

6.5. Start ClusterControl services on **cc1** only:

```
systemctl start cmon cmon-cloud cmon-ssh cmon-events cmon-proxy␣
↪kuber-proxy
```

6.6. Ping ClusterControl RPC interface for `cid 0` on **cc1** only, make sure you get the "Ok" reply:

```
$ s9s cluster --ping --cluster-id=0
PING Ok   4 ms
```

6.7. Start the lsyncd service on **cc1** only and make sure it is running (DO NOT START it on **cc2**):

```
systemctl start lsyncd
systemctl status lsyncd
```

6.8. We will intentionally set lsyncd to NOT start on boot for both **cc1** and **cc2**:

```
systemctl disable lsyncd
```

6.9. Monitor the file syncing operation. Make sure you see something similar to below:

```
$ tail -f /var/log/lsyncd.log
Thu Jan 22 17:45:02 2026 Normal: recursive startup rsync: /
↪var/www/html/cmon-repos/ -> cc2:/var/www/html/cmon-repos/
Thu Jan 22 17:45:02 2026 Normal: recursive startup rsync: /
↪usr/share/ccmgr/ -> cc2:/usr/share/ccmgr/
Thu Jan 22 17:45:02 2026 Normal: recursive startup rsync: /etc/ ->␣
↪cc2:/etc/
Thu Jan 22 17:45:03 2026 Normal: Startup of /
↪var/www/html/clustercontrol-mcc/ ->␣
↪cc2:/var/www/html/clustercontrol-mcc/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /etc/ -> cc2:/etc/␣
↪finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /var/lib/cmon/ ->␣
↪cc2:/var/lib/cmon/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /usr/share/ccmgr/ ->␣
↪cc2:/usr/share/ccmgr/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /etc/cmon.d/ ->␣
↪cc2:/etc/cmon.d/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /root/.s9s/ ->␣
↪cc2:/root/.s9s/ finished.
Thu Jan 22 17:45:04 2026 Normal: Startup of /var/www/html/cmon-repos/ -
↪> cc2:/var/www/html/cmon-repos/ finished.
```

6.10. On **cc2**, enable keepalived on boot and start it up:

```
systemctl enable keepalived
systemctl stop keepalived
systemctl start keepalived
```

6.11. On **cc2**, you should see it is entering the `BACKUP` state and you don't see the VIP there:

```
$ grep Keepalived /var/log/messages
2026-01-22T16:42:24.652694+00:00 cc1 Keepalived[720]: Starting VRRP␣
↪child process, pid=730
2026-01-22T16:42:24.652701+00:00 cc1 Keepalived[720]: Startup complete
2026-01-22T16:42:24.652722+00:00 cc1 Keepalived_vrrp[730]: (VI_1)␣
↪Entering BACKUP STATE (init)
```

6.12. Verify that only **cc1** has the VIP:

**cc1:**

```
$ ip a | grep global  # on cc1
inet 192.168.73.201/24 brd 192.168.73.255 scope global eth0
inet 192.168.73.200/32 scope global eth0
```

**cc2:**

```
$ ip a | grep global  # on cc2
inet 192.168.73.202/24 brd 192.168.73.255 scope global eth0
```

6.13. Verify that we have the `POSTROUTING` chain configured, to force VIP as the outbound address on **cc1**:

```
$ iptables -t nat -L POSTROUTING -n --line-numbers
Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    SNAT       all  --  0.0.0.0/0            0.0.0.0/0           ␣
↪to:192.168.73.200
```

At this moment, ClusterControl on cc1 is the primary and cc2 is the secondary (replica).

### Deployment summary

To summarize the setup up until this point:

- VIP is on **cc1**. Check with `ip` or `ifconfig` command.
- Only ClusterControl services on **cc1** are running, while all ClusterControl services on **cc2** are stopped.
- Only lsyncd service on **cc1** is running, while lsyncd service on **cc2** is stopped.
- We haven't configured Prometheus for monitoring yet. This will happen after we have deployed/imported a cluster into ClusterControl. See below.

Access ClusterControl GUI at `https://192.168.73.200` and complete the initial setup.

### 3.10.6 Deploy/Import the first cluster

It is necessary to have the first cluster in order to configure an external Prometheus server, as we don't want ClusterControl to use a local Prometheus on **cc1** (and/or **cc2**).

1. Login to ClusterControl GUI using the VIP at `https://192.168.73.200/`. Register the first admin user and continue.
2. If the host has Internet connection, ClusterControl will be automatically configured with a 30-day trial license.
3. Configure SSH key-based authentication using the same private key from **cc1** to all DB nodes.
4. Deploy or import a cluster using the **Deploy a cluster** wizard.
5. After the deployment/import job completes, ClusterControl will configure and install Prometheus on 192.168.73.209 (`prom.local`) automatically (as configured previously). Verify this by looking at the **Nodes** page.

Once a Prometheus is running correctly, we can conclude ClusterControl Replication is fully set up now.

### 3.10.7 Management

#### Failover from cc1 to cc2

In order to activate **cc2**, the following conditions must be true:

- All ClusterControl services are stopped on **cc1**.

```
systemctl stop cmon cmon-cloud cmon-ssh cmon-events cmon-proxy kuber-proxy
```

- MySQL server is stopped on **cc1**.

```
systemctl stop mysql
```

- Lsyncd service is stopped on **cc1**.

```
systemctl stop lsyncd
```

- Or you can just perform an OS shutdown on **cc1**.

Stop MySQL replication on **cc2**, reset binary log and disable `read-only`:

```
--- mysql on cc2
STOP REPLICA;
RESET MASTER;
RESET SLAVE ALL;
SET GLOBAL read_only=0;
SET GLOBAL super_read_only=0;
```

Start ClusterControl services on **cc2**:

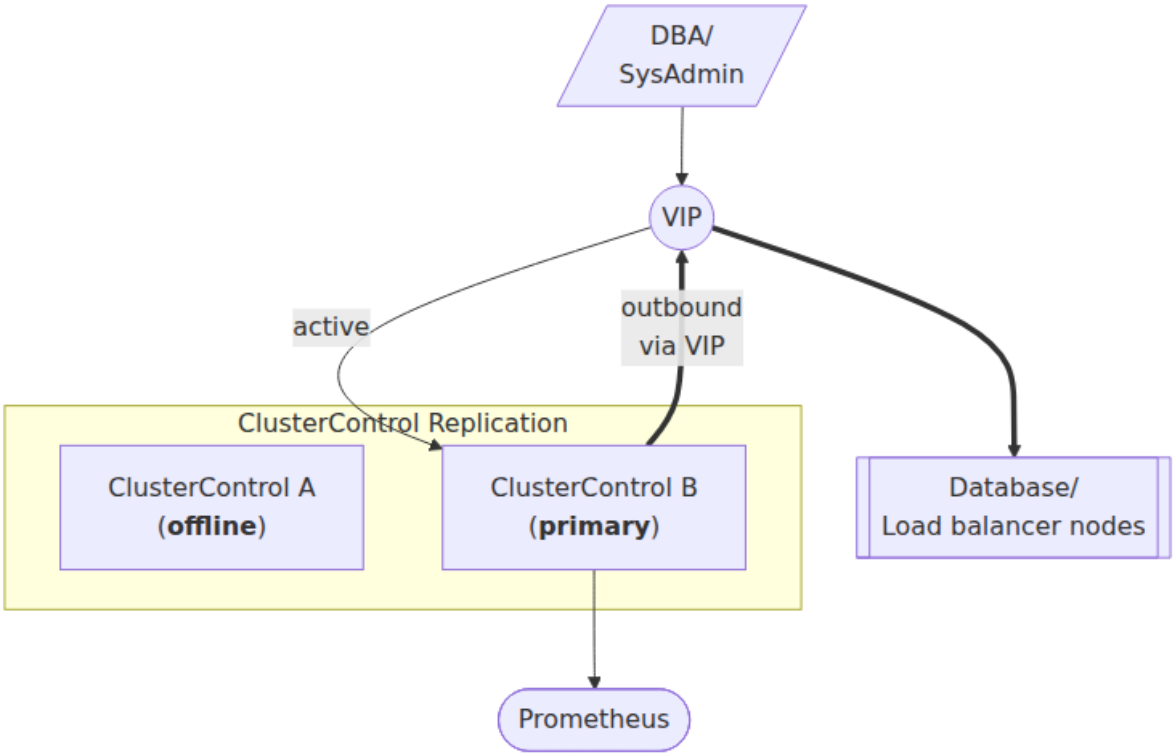
```
systemctl start cmon cmon-cloud cmon-ssh cmon-events cmon-proxy kuber-proxy
```

Start lsyncd on **cc2**:

```
systemctl start lsyncd
```

At this point, **cc2** is the primary and is holding the VIP (because Keepalived checks if cmon is running on this host).

The following diagram visualize the current topology after **cc2** is promoted:




### Preparing old cc1 as secondary

Let's say **cc1** was down, and we have promoted **cc2**, and **cc1** has come back up. We must reconfigure **cc1** to replicate from **cc2** (the current primary).

Make sure the following services are stopped:

```
systemctl stop cmon cmon-cloud cmon-ssh cmon-events cmon-proxy kuber-proxy
systemctl stop lsyncd
```

On **cc1**, run the following clone statement to sync MySQL data from **cc2**:

```
-- mysql on cc1
CLONE INSTANCE FROM 'clone'@'cc2':3306 IDENTIFIED BY 'Secur3_p4ssword';
```

On **cc1**, start the MySQL replication and check the status:

```
-- mysql on cc1
CHANGE REPLICATION SOURCE TO
  SOURCE_HOST = 'cc2',
  SOURCE_USER = 'repl',
  SOURCE_PASSWORD = 'Secur3_p4ssword',
  SOURCE_AUTO_POSITION = 1;
```

Start the replication and check the replication status:

```
-- mysql on cc1
START REPLICA;
SHOW REPLICA STATUS\G
```

Make sure you see the following:

```
    Replica_IO_Running: Yes
    Replica_SQL_Running: Yes
```

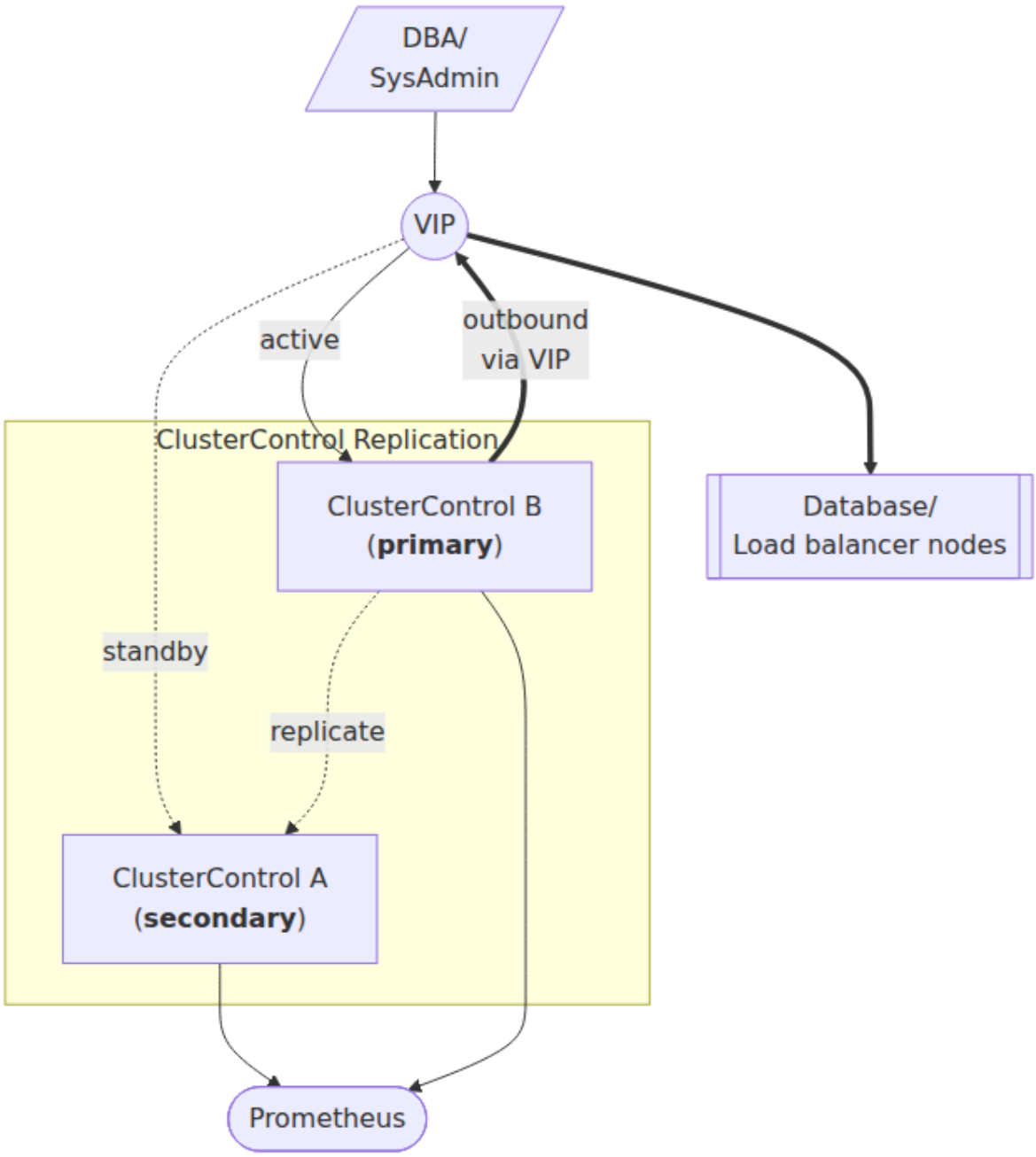
The above means MySQL on **cc1** is actively replicating from **cc2**.

On **cc1**, configure MySQL to be a `read-only` replica to protect against accidental writes:

```
-- mysql on cc1
SET GLOBAL read_only = 1;
SET GLOBAL super_read_only = 1;
```

At this point, **cc2** is the primary ClusterControl, and **cc1** is the secondary.

The following diagram visualize the current topology after **cc2** is promoted, and **cc1** is the secondary and syncing with the primary:




**Tip**

To promote **cc1** as the primary, repeat steps described previously in the failover process. Failover from **cc1** to **cc2** but swap **cc2** with **cc1** and vice versa.

### Outbound connections using VIP

In this setup, when a ClusterControl node is holding the virtual IP address (VIP), it is important to force the kernel to use the VIP as the source IP address of an outbound connection. We can achieve this by using iptables NAT table. This has been incorporated automatically in the Keepalived notify event to run `manage_snat.sh` script as shown in the Keepalived installation section.

For understanding purposes, if you want to add the rule manually, use the `-A` flag:

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 192.168.73.200
```

To remove the rule manually, use the `-D` flag instead:

```
iptables -t nat -D POSTROUTING -o eth0 -j SNAT --to-source 192.168.73.200
```

Or, you can also remove a rule using the line number:

```
iptables -t nat -L POSTROUTING -n --line-numbers     # list out the rules␣
↪with line numbers
iptables -t nat -D POSTROUTING 1                     # delete the iptables␣
↪rule on line 1
```

Otherwise, the kernel would use the primary IP address of `eth0` (`192.168.73.201` for **cc1** and `192.168.73.202` for **cc2**) for outbound connectivity. This will possibly cause issues because these IP addresses are not being explicitly granted during the deploy/import stage.

## 3.11 Reference Architecture

This page brings together production-tested architecture patterns, sizing math, and best practices for deploying ClusterControl and the database clusters it manages. The individual pieces: hardware sizing, HA setup, load balancer placement, network security, offline deployment, DR strategy are documented in depth on their own pages. This page assembles them into complete blueprints and answers "given my situation, what should this actually look like end to end?"

Use it as follows:

1. Start with *server sizing*, *networking*, *storage layout*, *SSH key management*, and whether you need *external CMON DB/Prometheus*. These apply to every deployment regardless of size.
2. Pick a *ClusterControl deployment architecture* based on how critical and how large your fleet is.
3. Layer in *multi-site/multi-datacenter*, *offline/restricted-network*, or *cloud* considerations as they apply to your environment.
4. Compare your plan against the *worked example deployments* and the *database cluster topologies* for the engines you run.

**Info**

This page is a curated overview, not a replacement for the detailed pages it links to. For exact step-by-step configuration, follow the links inline.

### 3.11.1 1. Recommended server specs (capacity planning)

Resource consumption on the ClusterControl host is driven primarily by:

- **Number of managed nodes** (database + load balancer): Each node adds SSH polling, Prometheus scrape targets, and CMON database writes.
- **Number of clusters**: Each cluster gets its own CMON configuration file (`/etc/cmon.d/cmon_<id>.cnf`) and adds its own recurring management/monitoring jobs to the controller's workload.

- **Monitoring interval and retention**: `db_stats_collection_interval` and `host_stats_collection_interval` (see *ClusterControl Configuration*) control how often CMON writes monitoring samples. Prometheus's own scrape interval and retention (default 15 days) control exporter-side storage.
- **Query monitoring**: Agent-based query monitoring (`cmonagent`) adds CPU and network overhead per monitored node, proportional to query throughput.
- **Backup activity**: Streaming backups through the ClusterControl host (rather than direct node-to-node or node-to-cloud) add CPU, network, and disk I/O spikes during backup windows.

### Sizing tiers

| Tier | Managed nodes | Recommended ClusterControl host | HA/scale pattern |
|---|---|---|---|
| Minimum | 1 | 2 CPU, 2 GB RAM, 20 GB disk | Single instance |
| Small | 3 – 15 | 4 CPU, 8 GB RAM, 50 GB disk | Single instance. Add a *Secondary Standby* if the fleet is business-critical |
| Medium | 15 – 50 | 16 CPU, 16 GB RAM, 150 GB disk | Single instance + Secondary Standby, or *CMON Controller Pool* |
| Large | 50 – 150 | 24 CPU, 32 GB RAM, 300 GB disk | CMON Controller Pool or *ClusterControl Replication with VIP* |
| Very large | 150 – 1000+ | Multiple hosts, each sized as "Large" | 5–8+ ClusterControl instances behind *Ops-C* |

**Note**

Note The "Very large" instance count and the HA/scale pattern assigned to each tier are rough guidelines, not prescriptive rules: the right HA pattern depends more on criticality than raw node count, and the 5–8 figure is a documented starting point (*How it Works? → Multiple controllers management*) to adjust from based on observed per-instance load rather than follow rigidly.

```
The host specs above are baseline minimums by managed-node count, taken␣
↪from the [hardware requirements␣
↪table](../getting-started/overview/requirements.md#hardware). Cluster␣
↪count, monitoring interval/retention, agent-based query monitoring, and␣
↪streaming backups through the ClusterControl host (the drivers listed␣
↪above) all add load independent of node count, so a fleet with many␣
↪clusters or heavy query/backup activity should size toward the higher␣
↪end of, or above, its tier. The [Capacity␣
↪planning](../getting-started/overview/requirements.md#capacity-planning)␣
↪section documents a real-world reference point of 8-24 CPU cores, 8-32␣
↪GB RAM, and 300 GB disk for a single controller managing 67 clusters␣
↪(~130 nodes), meaningfully heavier than this table's Small/Medium␣
↪figures for a similar node count. Treat the tiers here as a starting␣
↪point and validate against observed load, not as a hard ceiling.
```

See *Requirements → Hardware* for the underlying reasoning per resource (CPU/RAM/disk/network), and *Tuning for Large Deployment* once a host is running hot.

### CPU and RAM calculation

- **Start from the tier table above**, based on managed node count, then adjust up if you have many small clusters (each with its own CMON configuration and recurring job/monitoring overhead) rather than few large ones, or if agent-based query monitoring is enabled fleet-wide.
- **RAM is dominated by the CMON database's InnoDB buffer pool.** By default, ClusterControl allocates 50% of host RAM to `innodb_buffer_pool_size`. On a host dedicated to ClusterControl, you can push this to 70%, provided at least 2 GB is reserved for OS processes and caches:

```
innodb_buffer_pool_size = total_RAM × (0.5 to 0.7)
reserved_for_OS         ≥ 2 GB
```

  Check the InnoDB buffer pool hit ratio (`SHOW ENGINE INNODB STATUS\G`). A ratio close to `1000/1000` means the working set fits in RAM. A lower ratio means either more RAM or a higher buffer pool percentage is needed. See *Tuning for Large Deployment → Tuning InnoDB buffer pool size* for the safe change procedure.

### Disk calculation

Disk usage has three independent drivers that should be sized and ideally partitioned. See *Section 3* separately:

1. **CMON database growth**: Driven by `db_stats_collection_interval`/`host_stats_collection_interva` (lower interval = more rows written per unit time) and the number of managed nodes/clusters. There is no fixed formula here since it depends on cluster count and history retention. Monitor `/var/lib/mysql` growth over the first weeks of production use and adjust.
2. **Prometheus time-series storage**: This one is approximately calculable using Prometheus's general storage rule of thumb of ~1–2 bytes per compressed sample (a widely-used approximation from Prometheus's own storage model, not a ClusterControl-specific figure. Treat it as a starting estimate rather than an exact prediction):

```
disk_bytes ≈ (active_time_series_count) × (86400 /␣
↪scrape_interval_seconds) × retention_days × ~1.5 bytes
```

   Each managed node contributes multiple time series (node/process exporter + database-specific exporter. See *How it Works? → Monitoring tools* for the full exporter list). Increasing retention beyond the 15-day default, or adding many nodes, increases `/var/lib/prometheus` disk usage roughly linearly. See *Requirements → Disk space*.
3. **Backup repository (if used)**. See *Backup repository architecture* below. Size this independently of the two drivers above, since backup sets can dwarf monitoring data.

**Real-world reference point**

A single ClusterControl server with 24 CPUs, 32 GB RAM, and 150 GB disk (running at ~92% disk usage) has been observed in production managing **67 clusters and around 130 database/load balancer nodes**. See How to Optimize Performance of ClusterControl and Its Components for tuning guidance beyond this point.

### Backup repository architecture

Where backups land is itself a sizing and architecture decision:

- **On the database node**: Fastest, no network hop, but ties the backup's survival to the node's survival.
- **On the ClusterControl node**: Convenient for small deployments, but adds directly to the disk sizing above, and is explicitly **not recommended** if you run *ClusterControl Replication*.
- **Shared storage (NFS/CIFS/SMB)** mounted identically across ClusterControl HA nodes. Required if backups need to be visible to more than one ClusterControl node.
- **Cloud/object storage (S3-compatible, Azure Blob, GCS)**: The recommended default for offsite durability. See *Section 9* and *Admin Guide → Backup & Restore*.

### Monitoring subsystem sizing

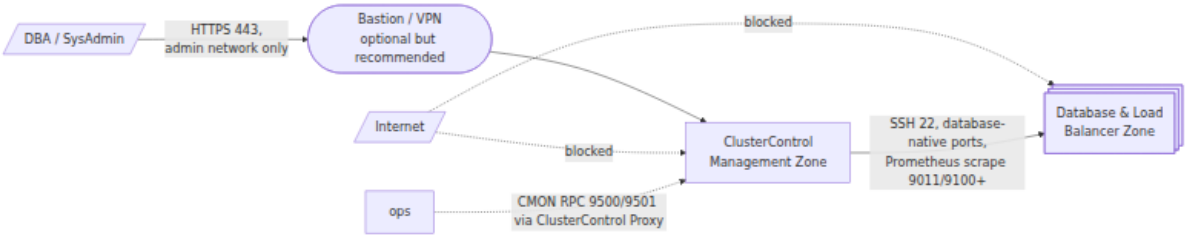
By default, **all clusters share one Prometheus instance** on the ClusterControl host. Split off a **dedicated Prometheus host** for a specific cluster when:

- A single cluster generates a disproportionate share of metrics volume (very high node count, very high query rate).
- The combined metrics volume across all clusters makes the shared instance a bottleneck.
- You need a longer retention period than the 15-day default.

See *Section 5* for how to configure a dedicated or fully external Prometheus.

## 3.11.2 2. Networking setup

ClusterControl should sit in a management zone that can reach the database zone over SSH, database-native, and Prometheus scrape ports, but is not directly reachable from the public Internet.



### Port matrix

| Service | Default port |
|---|---|
| SSH | 22 |
| HTTPS (ClusterControl GUI) | 443 |
| CMON RPC | 9500 |
| CMON RPC (TLS) | 9501 |
| CMON Events (notifications) | 9510 |
| CMON SSH (web-based console) | 9511 |
| CMON Cloud | 9518 |
| Streaming backup transfer (socat/netcat) | 9999, falling back to 9990–9998 (configurable) |
| ClusterControl Proxy (Ops-C / multi-controller) | 19501 |
| Prometheus | 9090 |
| Node/process exporters (all managed nodes) | 9100 / 9011 |

This is a condensed view. The full per-database-engine and per-load-balancer port matrix (MySQL, Galera, MongoDB, PostgreSQL, HAProxy, ProxySQL, MaxScale, Keepalived, garbd, SQL Server, Redis/Valkey, Elasticsearch) is maintained in one place to avoid drift: see *Requirements → Firewall and security groups*. Open the full 9990–9999 range (not just 9999) so concurrent or fallback backup streams aren't blocked; see `netcat_port` in *Reference Manuals → ClusterControl Controller*.

### SSH trust model

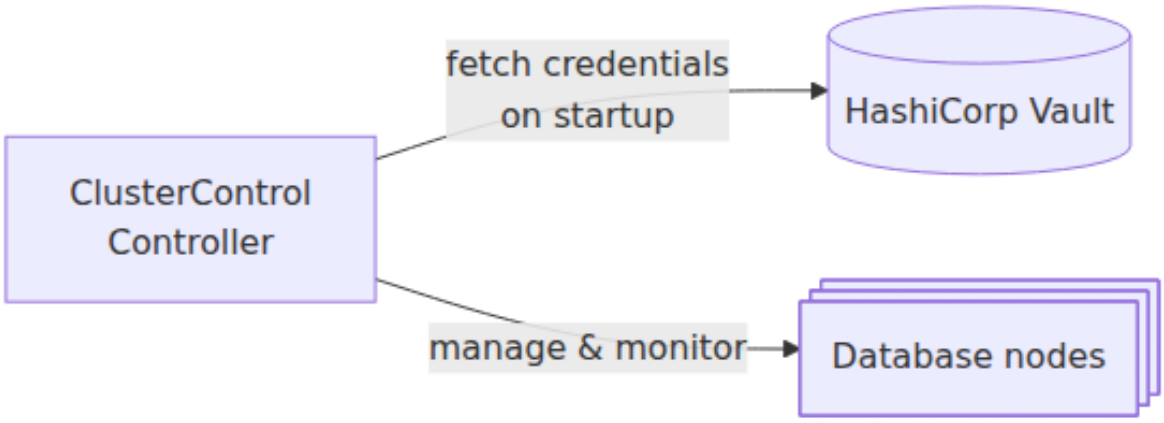
- Management/deployment operations are **agentless**, relying entirely on passwordless, key-based SSH from the ClusterControl host to every managed node. Monitoring, by contrast, is **agent-based** (Prometheus + exporters, plus `cmonagent` for query monitoring). See *How it Works? → Monitoring operation*.
- Trust is one-directional: managed nodes never need to SSH back into the ClusterControl host.
- Lock down SSH so that managed nodes only accept SSH connections from the ClusterControl host, and the ClusterControl host itself is not reachable via SSH from outside the admin network. See *Securing ClusterControl → SSH* and *Requirements → SSH key-based authentication*.

### Encryption architecture

| Traffic | Recommendation |
|---|---|
| Admin/browser → ClusterControl GUI | HTTPS with a properly signed certificate (default is self-signed). See *Securing ClusterControl → HTTPS* |
| ClusterControl clients → CMON controller | Prefer RPC-TLS (port 9501) over plaintext RPC (port 9500) |
| Application → database | Enable database-native TLS and manage certificates centrally. See *Management → Certificates* |
| Database → database (replication) | Enable replication-layer TLS where supported by the database engine |
| ClusterControl → cloud/offsite backup storage | Use the cloud provider's TLS endpoint. Consider backup encryption at rest |

#### Secrets management and centralized authentication

For stricter secrets management than flat CMON configuration files (which must be locked to `700` permissions, root-owned. See *Securing ClusterControl → File permission*), integrate **HashiCorp Vault**: the controller authenticates to Vault on startup and stores/retrieves credentials there instead.




See Integration → Hashicorp Vault for configuration parameters (`vault_token`, `vault_addr`, `vault_path`, `vault_auto_migrate`).

For access control, integrate an existing directory service (Active Directory, FreeIPA, OpenLDAP) so users authenticate with corporate credentials, and map LDAP groups onto ClusterControl user groups/privileges, combined with *Users & Teams* to scope what each team can see and do. See *Integration → LDAP*.

### 3.11.3 3. Storage configuration and partitioning

Rather than a single root partition for everything, separate the workloads that write to disk so a runaway one (typically backups or Prometheus) cannot starve the others:

| Mount point | Contains | Notes |
|---|---|---|
| `/` (root) | OS, ClusterControl packages | Keep this insulated from data growth by moving the items below to their own mounts |
| `/var/lib/mysql` | CMON database (MySQL/MariaDB) | Fast, low-latency storage (SSD). This is on the hot path for every job and monitoring write |
| `/var/lib/prometheus` | Prometheus time-series data | Size per the *disk calculation* above. Grows with retention and node count |
| `/var/log` (or a dedicated log partition) | `cmon.log`, `cmon_<id>.log` | CMON writes almost every second per cluster, so a spindle/cheap disk is fine here since writes are sequential. Point `logfile=` in `cmon_X.cnf` at this partition |
| Backup repository mount | Local backup copies (if stored on the ClusterControl node) | Use a **separate partition**, ideally on networked/clustered storage (NFS/CIFS/SMB) for easy mounting during restores. Never let this share the root partition |

See *Tuning for Large Deployment → File System and Partitioning* and *Requirements → Disk space* for the full guidance this table is based on.

### 3.11.4 4. SSH key management and rotation

#### Initial setup

ClusterControl uses `libssh` with RSA (fully tested) or other supported public-key algorithms (`ssh-dss`, `ssh-ed25519`, `ecdsa-sha2-nistp256/384/521`) for passwordless, key-based SSH from the ClusterControl host to every managed node. See *Requirements → SSH key-based authentication* for generating and distributing the initial key pair, and *Requirements → Operating system user* for the `sudo`/`pbrun`/`doas` privilege-escalation options that pair with it.

Best practices for the key itself:

- Use a **dedicated key pair** for ClusterControl. Never reuse a personal or shared administrator key, so revoking access for one purpose doesn't require rotating everything else that key touches.
- Keep the private key permissioned `600` and owned by root, the standard SSH requirement for a private key file, stricter even than the `700` required of CMON configuration files themselves (see *Securing ClusterControl → File permission*).
- For larger or segmented environments, consider a **key per environment or per network segment** (e.g., one key for production, one for staging) to limit blast radius if a single key is ever compromised.

#### Rotation procedure

`ssh_identity` is a per-cluster CMON configuration option (set inside `/etc/cmon.d/cmon_<id>.cnf`). If it is left undefined for a cluster, CMON falls back to the `os_user`'s home directory and looks for `.ssh/id_rsa`. If every cluster relies on that default rather than setting `ssh_identity` explicitly, they all resolve to the same key path, which is consistent with how *Migrating ClusterControl to Another Server* handles the key: it copies/replaces the key at that one fixed path (e.g. `/root/.ssh/id_rsa`) rather than editing every `cmon_<id>.cnf`. If you *have* set `ssh_identity` explicitly to a custom, per-cluster path for any cluster, you'll need to update those files individually instead. Rotate without downtime by adding the new key **before** removing the old one:

1. Generate a new key pair at a temporary path on the ClusterControl host (`ssh-keygen -t rsa -f ~/.ssh/id_rsa.new`), keeping the old key pair in place for now.
2. Append (not replace) the new public key to `~/.ssh/authorized_keys` on every managed node, via configuration management or with `ssh-copy-id -i ~/.ssh/id_rsa.new.pub <os_user>@<managed-node>`.
3. Verify passwordless SSH with the new identity from the ClusterControl host to every node (`ssh -i ~/.ssh/id_rsa.new <os_user>@<managed-node>`).
4. Move the new private key over the existing path (e.g. `mv ~/.ssh/id_rsa.new /root/.ssh/id_rsa` and `mv ~/.ssh/id_rsa.new.pub /root/.ssh/id_rsa.pub`). Only if `ssh_identity=` was explicitly set to a custom, per-cluster path do you need to place it there and update the corresponding `/etc/cmon.d/cmon_<id>.cnf` files instead.
5. Restart the `cmon` service and confirm management/monitoring jobs continue to run normally.
6. Once confirmed stable, remove the old public key from `authorized_keys` on every managed node, and securely delete the old private key.

- Rotate on a **regular schedule** (aligned to your organization's key-rotation policy) and **immediately** after any personnel offboarding or suspected key exposure.
- Migrating ClusterControl to new hardware is a natural point to rotate keys as well. See *Migrating ClusterControl to Another Server*, which calls out key rotation explicitly as part of security/compliance

re-segmentation.

### 3.11.5 5. Using an external CMON database and external Prometheus

#### External CMON database

By default, the installer provisions a local MySQL/MariaDB server on the ClusterControl host to act as the CMON database (`mysql_hostname=127.0.0.1`). This can instead point at a MySQL/MariaDB server you already operate, for example a centrally-managed, already-HA MySQL service:

- During manual installation, initialize the controller against a remote database explicitly:

```
cmon --init \
     --mysql-hostname="<external-db-host-or-ip>" \
     --mysql-port="3306" \
     --mysql-username="cmon" \
     --mysql-password="xxxx" \
     --mysql-database="cmon" \
     --hostname="<clustercontrol-host-ip>" \
     --rpc-token="<generated-token>" \
     --controller-id="clustercontrol"
```

  See *Manual Installation* for the full procedure, and the *mysql_hostname*/`mysql_port` configuration options for all supported aliases.

- The install script (`install-cc`) also supports reusing an **existing local** MySQL server rather than installing a new one. See *Quickstart*.

**Warning**

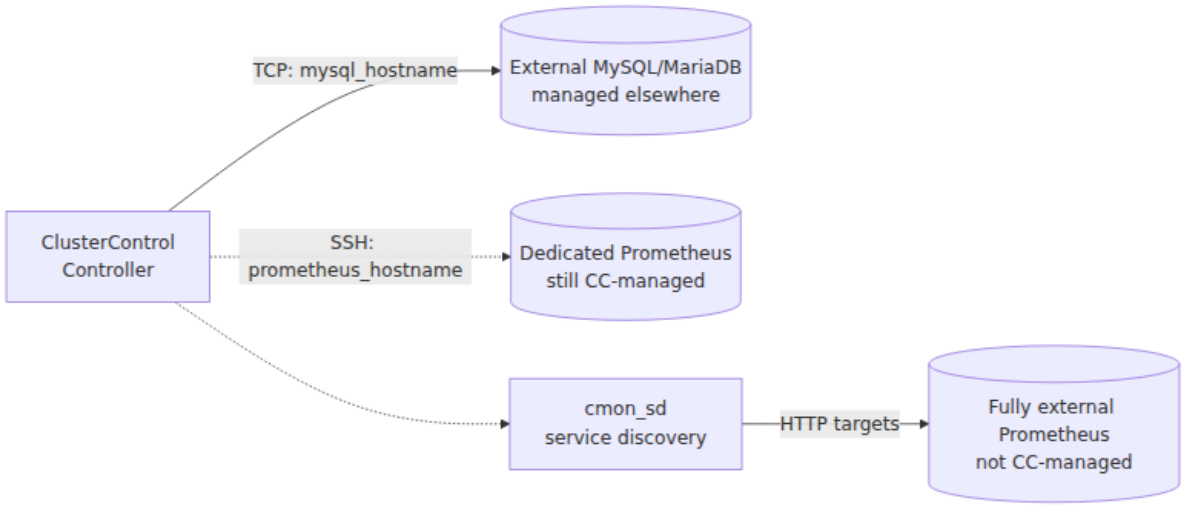
Attention

The CMON database sits on the hot path for nearly every management and monitoring operation, so an external database must be low-latency (TCP, not WAN) and reachable at all times, or ClusterControl will block waiting for a working connection. Plan its own HA and backups independently. ClusterControl does not manage the lifecycle of a database it did not provision.

#### External Prometheus

There are two distinct patterns, depending on whether ClusterControl should manage the Prometheus lifecycle:

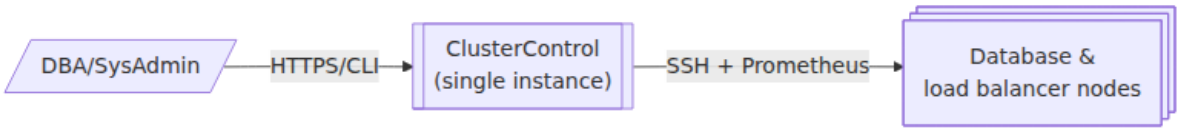
- **Dedicated Prometheus node, still managed by ClusterControl**: reconfigure a cluster's agent-based monitoring to point at a separate host over SSH (`prometheus_hostname`, `prometheus_ssh_user`, and related options). ClusterControl still installs, configures, and daemonizes Prometheus and exporters there. See *Integration → Prometheus*.
- **Fully external Prometheus, not managed by ClusterControl**: use the *cmon_sd* service discovery tool, which exposes an HTTP endpoint listing every node ClusterControl currently manages (as Prometheus `http_sd_configs` targets). Point your own, independently-run Prometheus at this endpoint instead of relying on ClusterControl's built-in instance.




### 3.11.6 6. ClusterControl deployment architectures

#### 6.1 Single ClusterControl controller

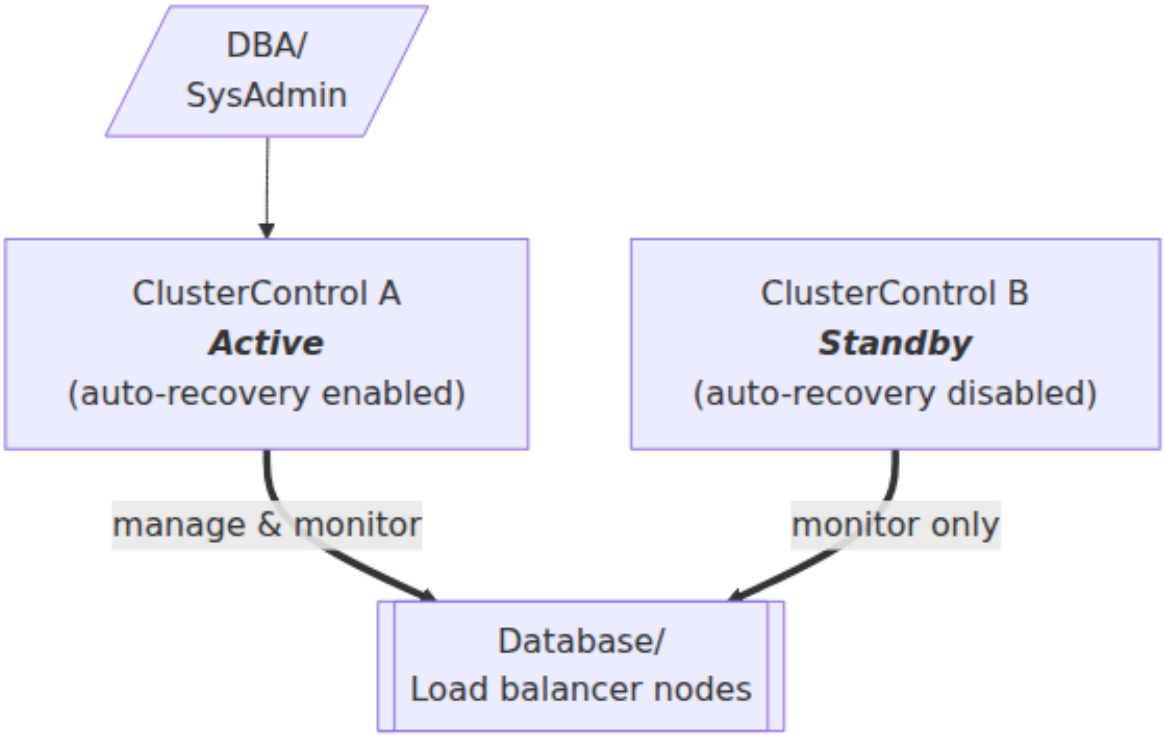
The default and simplest architecture: one `cmon` controller manages one or more clusters. Per *How it Works? → Architecture*, "ClusterControl components must reside on a node separate from the database cluster". For example, a three-node Galera Cluster requires ClusterControl on a fourth, independent node (see *Online Installation → Requirements*). Running it on a managed database node itself isn't a supported topology.




Appropriate for small, non-critical fleets (see the *sizing tiers*). Has no built-in redundancy for the ClusterControl layer itself: if this host is lost, monitoring and automation pause until it (or a restore of it) comes back, though the database clusters themselves keep running unaffected.

#### 6.2 Active/passive secondary standby

A second, independent ClusterControl instance imports and monitors the same clusters as the primary, but with **auto-recovery disabled**, so it never races the primary to perform failover.

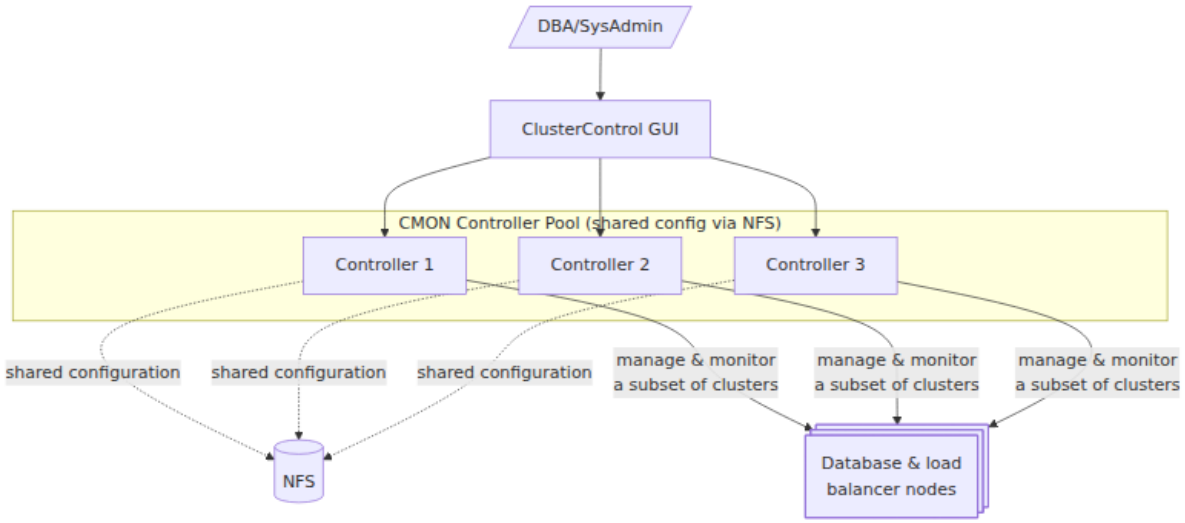



- No shared state between the two instances: each has its own CMON database and configuration.
- Promotion is **manual**: to fail over, enable auto-recovery on the standby and disable it on the old primary.
- Cheapest and simplest HA pattern. Also, the natural monitoring leg of a *multi-site deployment*.
- One license covers both instances: licensing is bound to managed node count, not ClusterControl instance count.

See *Redundancy & High Availability → Secondary Standby* for the full setup procedure.

### 6.3 CMON Controller Pool

A pool of controllers shares the workload of managing clusters, instead of a single controller carrying the entire fleet. GA since ClusterControl 2.4.0.

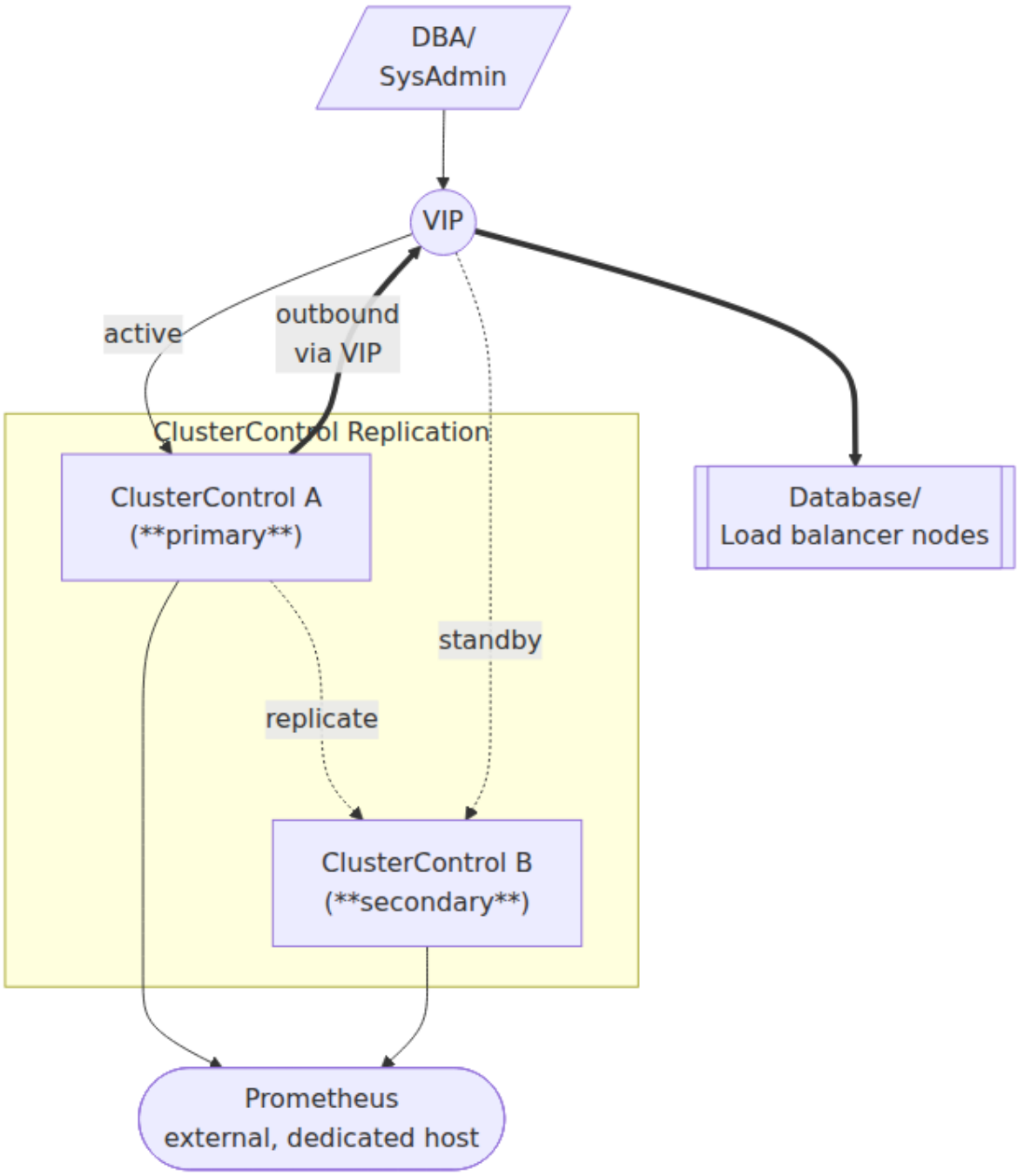


- **Horizontal scaling**: controllers can be added to or removed from the pool as workload grows or shrinks, redistributing clusters across the pool.
- **Shared CMON configuration** kept consistent across the pool via NFS.
- **Gradual adoption**: pool mode can be enabled/disabled on an existing controller without a full redeployment.
- Reach for this pattern when a single controller becomes a **performance** bottleneck, not just an availability one, e.g., medium-to-large fleets where job scheduling and monitoring load need to be spread horizontally.

See *Redundancy & High Availability → CMON Controller Pool* for enabling, adding, and removing controllers.

### 6.4 ClusterControl Replication with VIP

A primary-secondary HA architecture built from open-source building blocks (MySQL replication, `rsync`/`lsyncd`, Keepalived), where only one ClusterControl instance is active at a time and a virtual IP (VIP) provides a single, stable endpoint for both the GUI and outbound management traffic. Introduced in ClusterControl 2.4.0.




Key components and constraints:

- **MySQL replication** gives CMON database redundancy (primary → replica). **File syncing** (`rsync`/`lsyncd`) keeps ClusterControl's own files consistent one-way. **Keepalived** owns the VIP.
- Requires a minimum of **3 nodes**: two ClusterControl nodes plus an **external Prometheus** host: Prometheus must not be co-located with ClusterControl, and is itself a single point of failure unless you add remote-write to a second Prometheus.
- The VIP relies on Layer-2 broadcast (VRRP), so this is a **LAN pattern by default**: usable across a WAN only if L2 connectivity is bridged/extended, and this is a poor fit for most cloud VPCs (see *Section 9*).
- Managed cluster backups should **not** be stored on local disk on either ClusterControl node. The

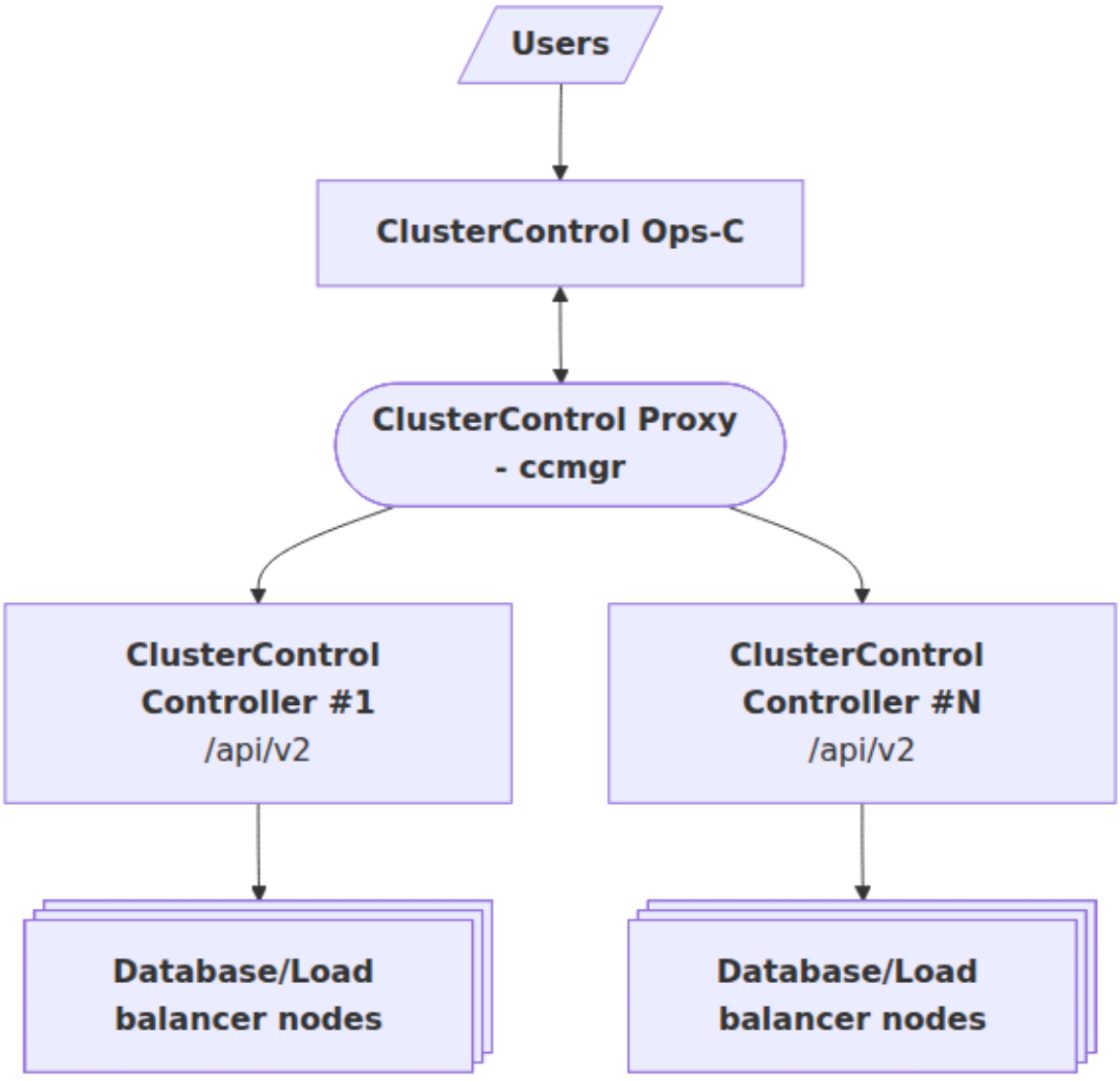
exception is shared storage (NFS/CIFS/SMB) mounted at the same path on both, since it remains reachable regardless of which node is active.

- Requires MySQL 8+ with the CLONE plugin enabled.

See *Redundancy & High Availability → ClusterControl Replication* for the full deployment steps, including OS-specific instructions.

### 6.5 Multi-controller with Ops-C

For very large fleets, running more controllers is not enough on its own. **ClusterControl Ops-C** ("Operations Center") provides a single, centralized view, aggregating status across independently-running controllers through the **ClusterControl Proxy** (`ccmgr`).




- As a rule of thumb, plan for **5 to 8 ClusterControl instances per 1000+ database nodes**, depending on hardware size per instance. Treat this as a starting point and adjust based on observed per-instance load (see the *sizing tiers* above).
- Installed by default since ClusterControl 2.3.2, but disabled until enabled under **Settings → License → Enable Multiple Controllers**, which provisions a dedicated **root** user used only for managing

CMON controllers. This integration user can itself authenticate via an existing LDAP or Active Directory service instead of a local ClusterControl password.

- Because Ops-C only aggregates and visualizes, it does not itself need to be highly available in the same sense as a controller.

See *User Guide → ClusterControl Ops-C* and *Reference Manuals → ClusterControl Proxy*.

### 6.6 Choosing between patterns

| Criteria | Secondary Standby | CMON Controller Pool | ClusterControl Replication (VIP) |
|---|---|---|---|
| Primary goal | Availability (DR-style) | Horizontal scale + availability | Availability with a single, stable endpoint |
| Failover | Manual | N/A. Workload is already distributed | Automatic, VIP-based |
| Shared state between CC instances | None | Shared config via NFS | MySQL replication + file sync |
| Extra infrastructure | +1 CC host | +N CC hosts, shared NFS | +1 CC host, +1 dedicated Prometheus host |
| WAN-friendly | Yes | Depends on latency to shared NFS | No. LAN/L2-only by default |
| Cloud-friendly | Yes | Yes | Needs a cloud-native VIP substitute. See *Section 9* |
| Best for | Small/medium fleets, cross-DC monitoring leg | Medium/large fleets outgrowing one controller | Fleets needing a single, always-on endpoint |

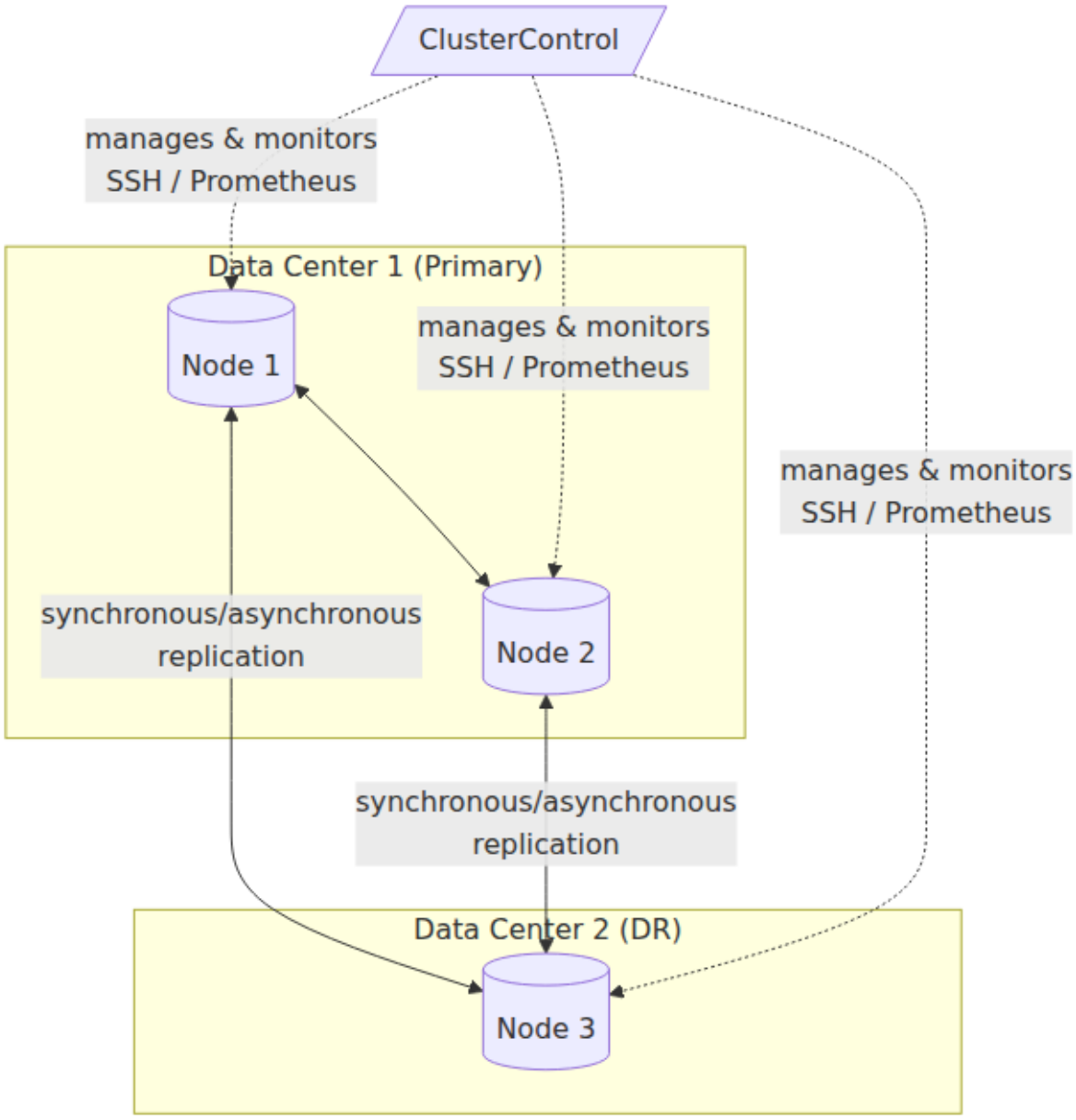
**Note**

Note The WAN-friendly/Cloud-friendly assessment for CMON Controller Pool is a general characterization of shared-NFS behavior, not a specific claim documented in *Redundancy & High Availability*. Validate NFS latency and availability for your own network before relying on it across sites or clouds.

### 3.11.7 7. Multi-site / multi-datacenter deployment

#### Pattern A: stretched cluster across data centers

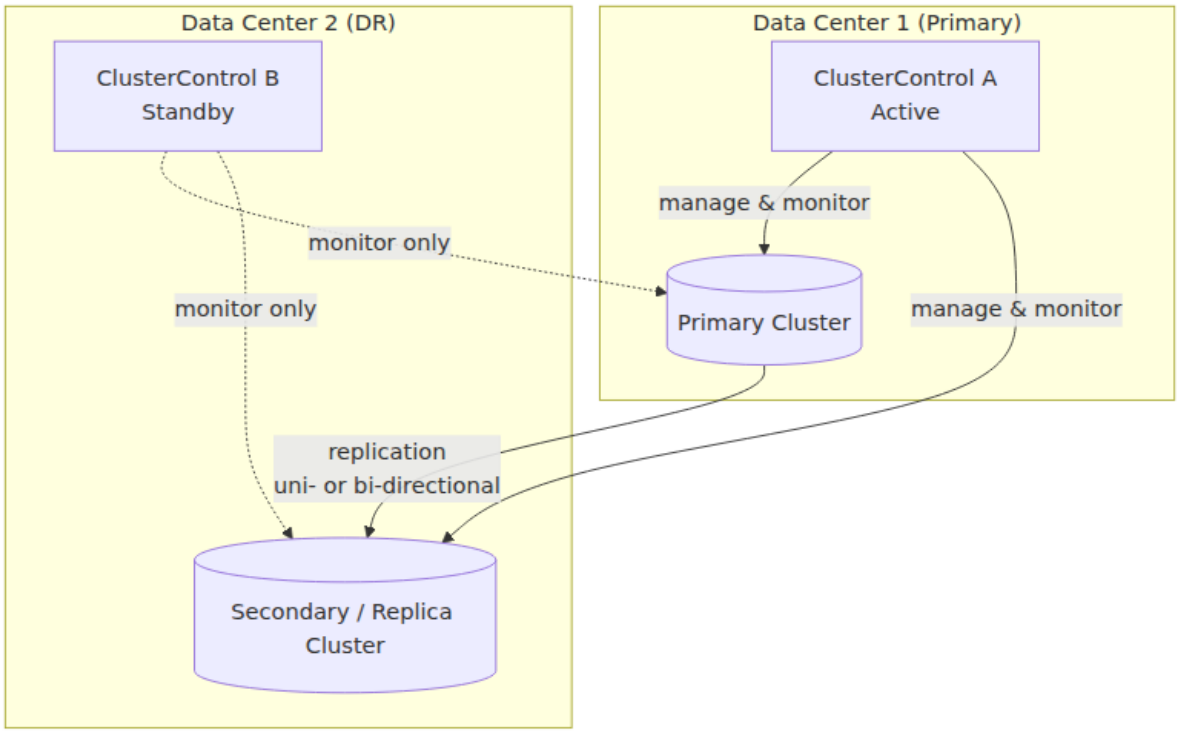
A single ClusterControl instance manages nodes spread across two or more data centers. This requires low, stable WAN latency between sites. There's no fixed, ClusterControl-documented threshold for this, but as a general guideline, keep round-trip latency as low as possible (well under 100 ms is a commonly cited target) for certification-based clustering (Galera/PXC), since every write must be certified across all sites before committing.



Place ClusterControl in, or nearest to, the data center you consider primary. See *Requirements → Network bandwidth*. For Galera-based clusters spanning sites, use **Galera segments** to group nodes by physical site: only one node per segment relays writesets to the other segments, reducing cross-WAN chatter. See the `Galera Segment` note in *Add Database Node → MySQL Galera*.

### Pattern B: ClusterControl HA across data centers

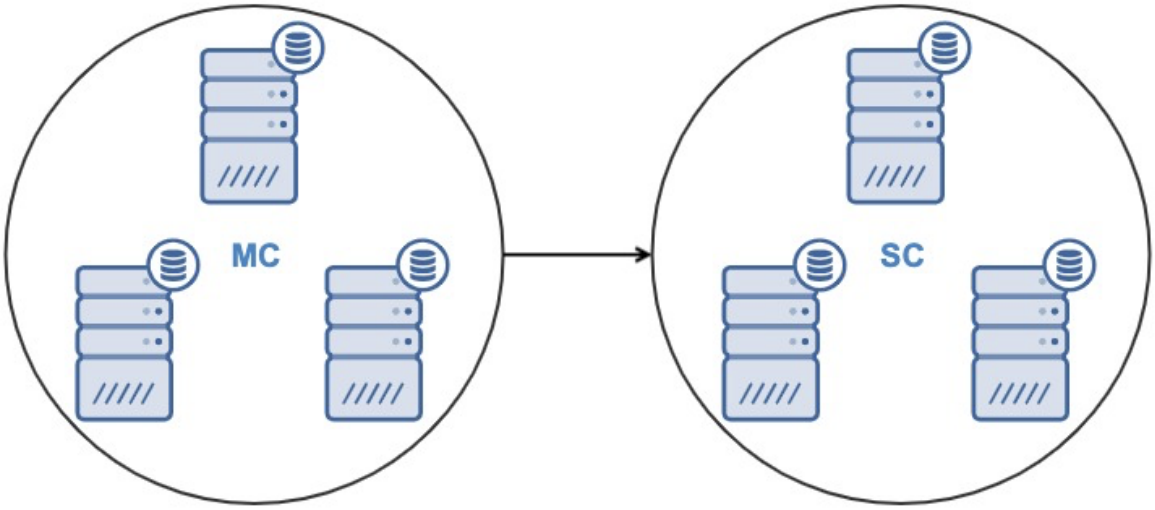
Applies the *Secondary Standby* pattern to a cross-site database topology. A single active ClusterControl instance in DC1 deploys the primary cluster and, via **Create Replica Cluster**, a second, independent cluster in DC2 linked by uni or bi-directional replication. The same controller manages and monitors *both* clusters, the same way it would if they were in one DC (this is the controller you'd run `s9s replication --list` against for either side. See *Create Replica Cluster*). A standby ClusterControl instance in DC2 imports and monitors those same two clusters as well, exactly as Secondary Standby specifies, but with auto-recovery disabled.




If DC1 (and the active controller with it) goes down, promote the standby per Section 6.2's usual procedure (enable auto-recovery on `ClusterControl B`) to take over active management and monitoring of both clusters. Independently of that, failing over the *database* traffic to the secondary cluster means disabling read-only on it and redirecting application traffic there (automatic if a load balancer already sits in front, manual otherwise). See *Deploy and Failover to Secondary Cluster* for that procedure. If replication was bi-directional, failback can happen once the old primary returns and catches up to zero lag. Otherwise, the old primary must be rebuilt from the secondary.

### Pattern C: cluster-to-cluster (C2C) replication

For active-active or DR setups where two separate clusters need to stay in sync, ClusterControl can create a replication layer between a source and a target cluster (uni or bi-directional) via **Create Replica Cluster**, initiated from the source cluster's dashboard. The clusters are independent database topologies linked by replication, not independently-managed ClusterControl deployments. The controller that creates the link manages and monitors both sides:




**Warning**

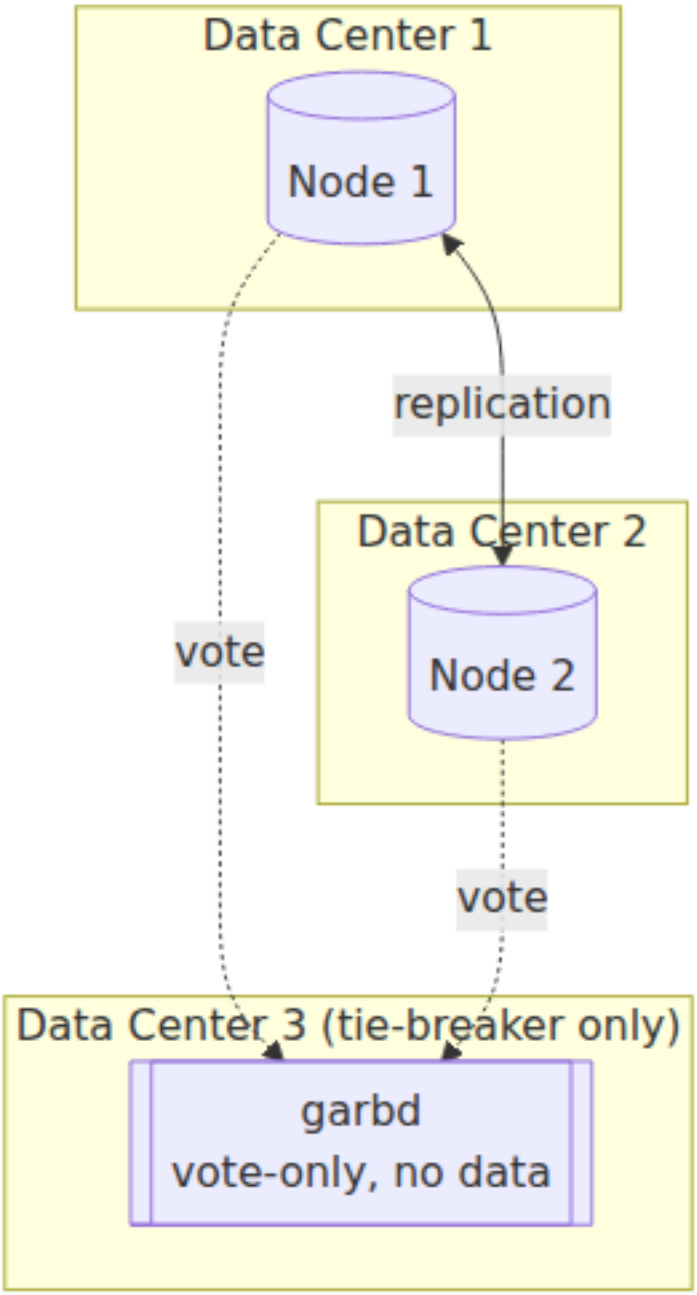
Attention

Active-active (bi-directional) replication is only recommended when applications write to disjoint data sets on either cluster, since the underlying replication engines do not offer conflict detection or resolution.

All nodes in the replica cluster are read-only by default, making this pattern suitable both as **read-scaling** across regions and as **DR**: the same replica cluster can serve both purposes. See *Scaling → Create Replica Cluster*.

### Pattern D: three-data-center quorum for Galera

A refinement of Pattern A for exactly two "real" sites: a 2-node/1-node split can survive loss of the one-node site but not loss of the two-node site. Instead, place one database node in each real site and a lightweight *garbd* arbitrator in a third, independent site to provide a third vote.




This gives true multi-site fault tolerance (losing any one of the three sites still leaves a quorum) at a fraction of the cost of a full third data node, since `garbd` carries no data. Deploy `garbd` on its own host in DC3, not on the ClusterControl node itself. See *Create Load Balancer → Galera Arbitrator* for deployment details.

### Disaster recovery: backup-based fallback

Regardless of which pattern above you run, layer a low-cost, backup-based fallback underneath it: schedule backups, upload them to cloud/offsite storage, and verify restores periodically. This is slower to recover (hours, not minutes) but has no standing infrastructure cost, and catches scenarios (e.g., logical corruption) that replication alone would faithfully propagate rather than protect against.

The one-click *Create Cluster from Backup* GUI wizard is currently limited to Galera Cluster (MariaDB Galera, Percona XtraDB Cluster) backups. For every other engine, restore the backup onto new nodes using that engine's own restore procedure (see *Admin Guide → Backup & Restore*) and import the result as

a new cluster instead.

### 3.11.8 8. Offline or restricted-network deployment

In air-gapped or restricted-network environments, ClusterControl, the underlying OS, and every database engine's packages must come from a repository reachable **inside** the network: there is no direct path to the public Internet at any point.

Two documented approaches, depending on scope:

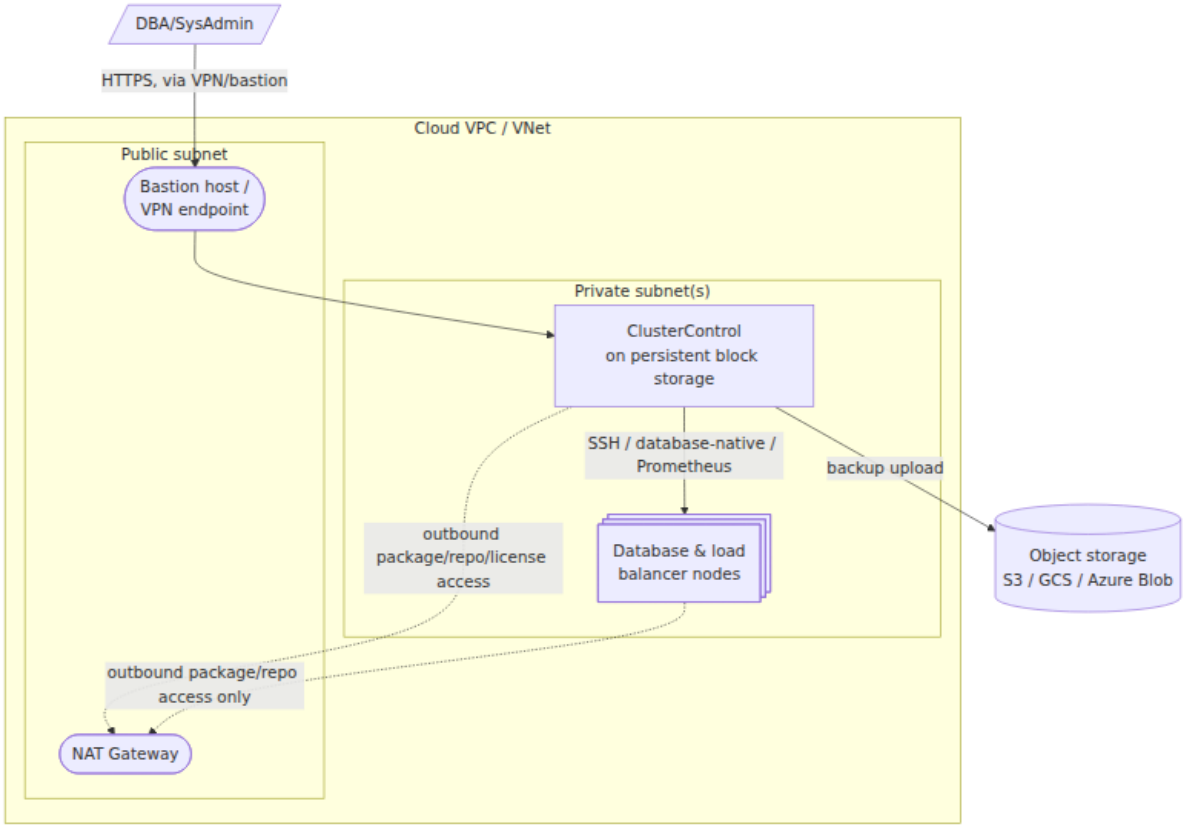
- **Single-purpose air-gap tarball**: run `install-cc --airgap-repo` on a temporary host matching the target OS to build and export a `.deb`/`.rpm` repository tarball, then import it on the air-gapped ClusterControl host. See *Offline Installation*.
- **Full satellite repository server**: for air-gapped environments that also need to deploy databases (MySQL, MariaDB, PostgreSQL, MongoDB, etc.) offline, stand up an internal repository mirror serving OS, database-vendor, and Severalnines packages to every node. See *Deploy Database Cluster in Offline Environment*.

Operational considerations specific to this architecture:

- **What won't work without Internet**, even after installation: uploading backups to cloud storage, Cloud Providers integration, load balancer deployment (needs EPEL/vendor repos), in-place upgrades, and new cluster/database deployment: all of these need their own repository access, satellite-mirrored or otherwise. See the full list in *Offline Installation*.
- **Monitoring exporters** (Prometheus, node/process/database exporters) must be pre-downloaded into `/var/cache/cmon/packages` on the ClusterControl node. The source URLs are listed in `/usr/share/cmon/templates/packages.conf`. See *How it Works? → Monitoring tools*.
- **NTP** must be served from an internal time source reachable by every node, since public NTP servers are unreachable. See *Requirements → Timezone*.
- **Licensing** normally activates automatically over the Internet. For offline environments, request an offline trial license from Severalnines Sales in advance (contact Sales directly regarding production licensing for air-gapped environments). See *Getting Started → Editions and releases*.

### 3.11.9 9. Cloud deployment considerations

Cloud deployments follow the same architecture patterns as on-premise ones, with a handful of cloud-specific adjustments:




- **Launch from a marketplace image** where available to skip manual dependency setup, or install manually into an existing VPC. See *ClusterControl on Marketplaces* for the passwordless-SSH and key-pair setup specific to cloud-launched hosts.
- **VIP-based failover needs a cloud-native substitute.** *ClusterControl Replication's* Keepalived VIP relies on Layer-2 broadcast (VRRP), which most cloud VPCs/VNets do not support across subnets. In the cloud, prefer the *Secondary Standby* or *CMON Controller Pool* patterns instead. If a single stable endpoint is required, a DIY option (not a ClusterControl-provided or documented feature, so build and test it independently) is to front ClusterControl with the cloud provider's own load balancer, or a reassignable Elastic IP / floating private IP driven by a custom failover script.
- **Use persistent, network-attached block storage** (EBS, Persistent Disk, Managed Disk) for `/var/lib/mysql` (CMON DB) and `/var/lib/prometheus`. Never ephemeral/instance-local storage, which is lost on stop or reclaim.
- **Use native object storage for backups** (Amazon S3, Google Cloud Storage, Azure Blob, or any S3-compatible provider) rather than local disk, for durability and to avoid growing the ClusterControl host's own storage. See *Integration → Cloud Providers*.
- **Keep ClusterControl and its managed nodes in the same region on low-latency private networking**, but distribute HA database members across availability zones where supported to avoid a single-zone failure. Use a Secondary Standby or Ops-C for cross-region visibility instead of stretching a single controller across regions.
- **Security groups/NSGs replace on-prem firewalls**: apply the same *port matrix* as security group rules, scoped to the specific subnets/security groups that need each port rather than `0.0.0.0/0`.

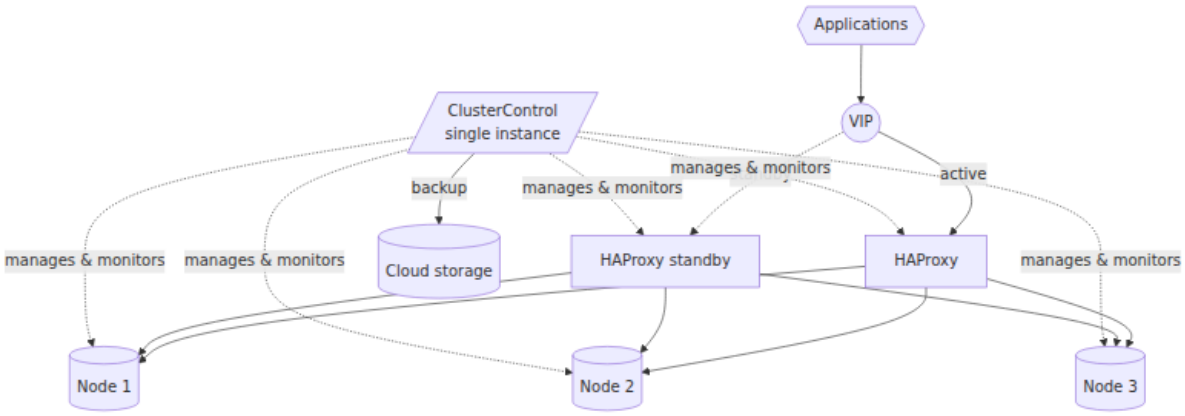
### 3.11.10 10. Example deployments

The following examples stitch the patterns above into complete blueprints for three common situations. Treat the numbers and pattern choices as a starting point, not a hard rule. Validate against your own workload using *Section 1*.

#### 10.1 SME / internal database usage (standard customers)

A typical profile for a small-to-mid-size company running ClusterControl to manage its own production databases for internal applications (ERP, e-commerce backend, internal tools), the most common ClusterControl deployment shape.

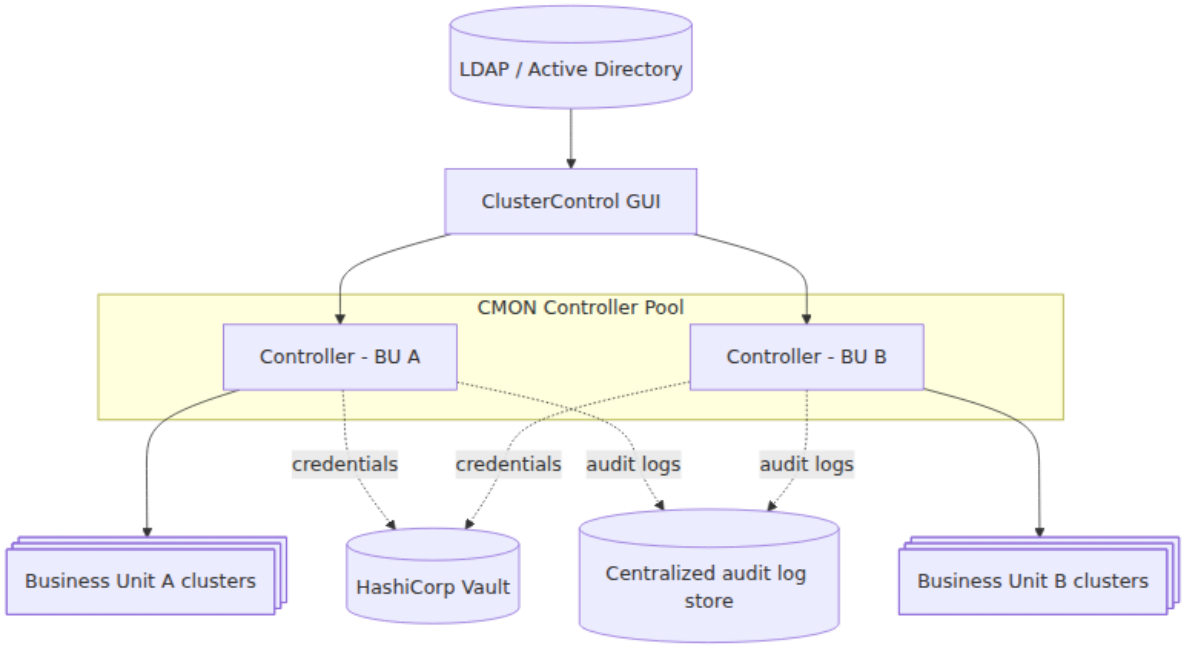
| Layer | Choice |
|---|---|
| ClusterControl | *Single instance*, 4 CPU / 8 GB RAM / 50 GB disk |
| Database topology | One Galera/PXC cluster (3 nodes) + HAProxy/Keepalived (2 nodes) |
| Monitoring | Default shared Prometheus on the ClusterControl host |
| Backups | Daily full + streaming to cloud storage, delete-after-upload enabled |
| DR | *Backup-based fallback* only |
| Security | Self-signed TLS acceptable initially. SSH locked to the ClusterControl host only |




#### 10.2 Private DBaaS provider

A profile for a larger organization: a software vendor running many tenant/product databases internally, or a regulated enterprise (such as a bank) offering database infrastructure as an internal shared service across business units. “Private” here means the consumers are internal teams/products, not external paying customers. The emphasis shifts to multi-tenancy, RBAC, audit, and compliance rather than public self-service.

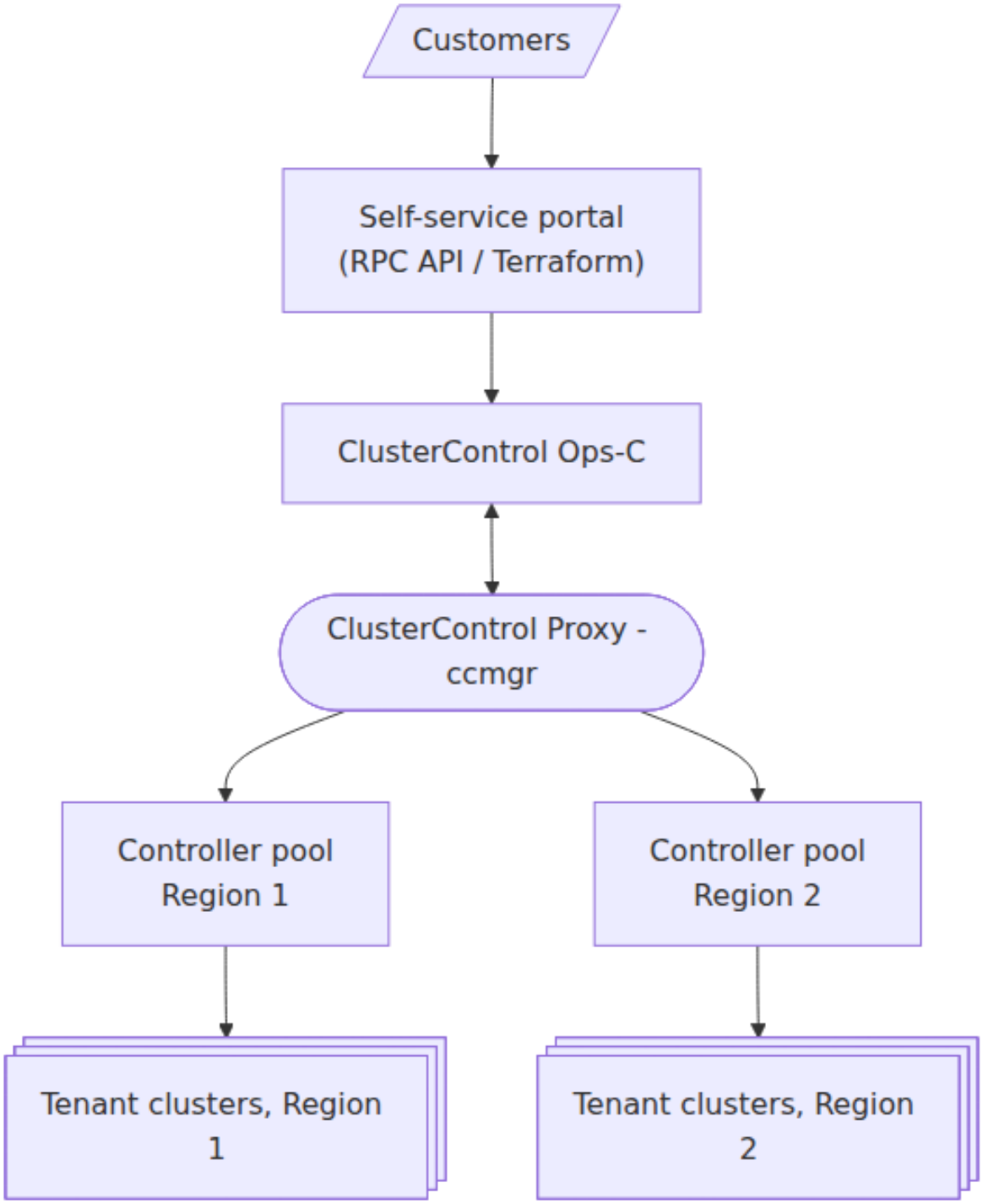
| Layer | Choice |
|---|---|
| ClusterControl | *CMON Controller Pool* or *Secondary Standby* per region/business unit |
| Database topology | Multiple clusters segmented per business unit/product, mixing engines per *Section 11* |
| Monitoring | Dedicated Prometheus per business unit or high-volume cluster |
| Backups | Centralized backup repository (NFS or private object storage), with scheduled verification |
| Access control | LDAP/Active Directory-backed *Users & Teams*, scoped per business unit. *Database Audit Logs* enabled for compliance |
| Security | RPC-TLS, HashiCorp Vault for credential storage, network-segmented per business unit |



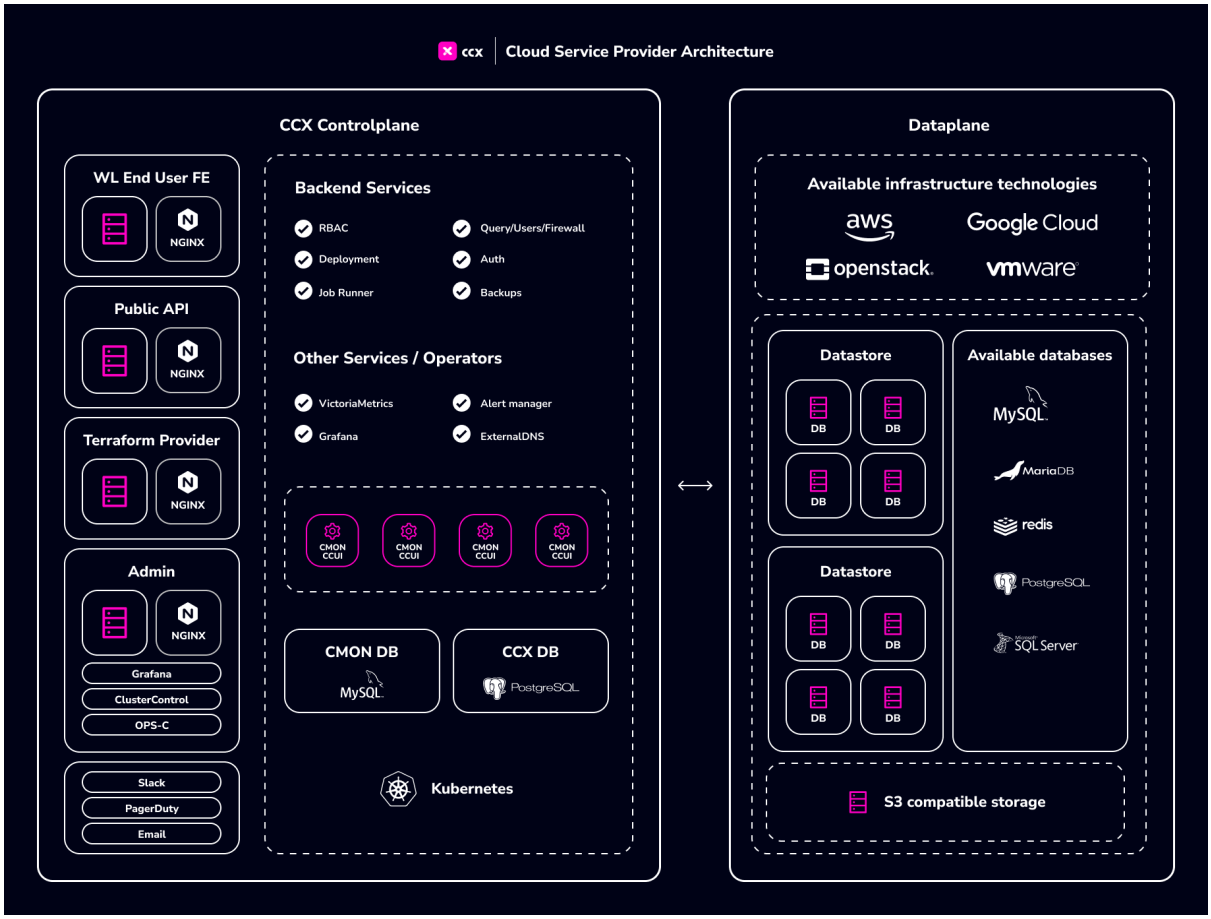

### 10.3 Public DBaaS provider

A profile for a telecom or service provider offering database hosting/management as a product to external, paying customers, requiring strict tenant isolation, self-service automation, and scale across regions or subsidiaries.

| Layer | Choice |
|---|---|
| ClusterControl | Multiple controller pools per region/subsidiary, aggregated via *Ops-C* |
| Automation | Customer self-service built on the *ClusterControl RPC API* and/or *Terraform Provider*, rather than the GUI directly |
| Database topology | One cluster (or cluster set) per tenant, isolated at the network and credential level |
| Monitoring | Dedicated Prometheus per region. Per-tenant dashboards/alerting |
| Backups | Regional object storage repositories, isolated per tenant |
| Security | Full RPC-TLS, Vault-backed credentials, strict per-tenant network segmentation |




At this scale, building and operating this multi-region control/data-plane split yourself is a significant undertaking. **CCX**, Severalnines' own hosted database-as-a-service platform, already implements this pattern as a productized control plane/data plane architecture: a control plane (RBAC, deployment, job runner, auth, backups, metrics via VictoriaMetrics/Grafana, alerting) driving a data plane that provisions MySQL, MariaDB, PostgreSQL, Redis, and SQL Server across AWS, Google Cloud, OpenStack, or VMware, backed by S3-compatible storage:




Providers building a customer-facing DBaaS offering should evaluate **building on CCX** (or a similar architecture modeled on it) against self-hosting a bespoke Ops-C/controller-pool stack. The trade-off is control and customization (self-hosted) versus time-to-market and operational overhead (CCX).

### 3.11.11 11. Database cluster reference topologies

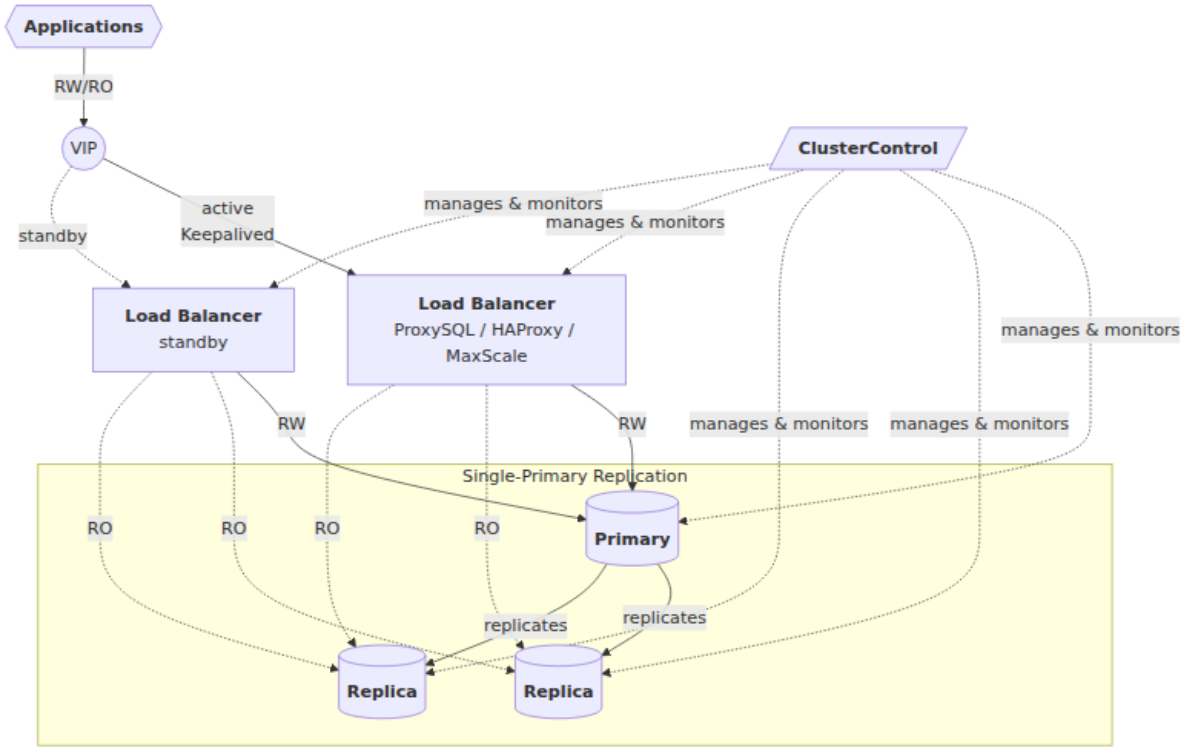
The following patterns show recommended **production** architectures, including load balancer and VIP placement, for each database family managed by ClusterControl. For a minimal, step-by-step deployment (fixed IPs, exact commands), see the linked *Day-1 Operations tutorial* instead. This section focuses on the shape of the architecture and the reasoning behind each choice.

| Family | Pattern | Minimum production nodes | Recommended production nodes | Load balancer |
|---|---|---|---|---|
| MySQL, MariaDB | Single-primary replication | 2 (1 primary + 1 replica) | 3 (1 primary + 2 replicas) | ProxySQL, HAProxy, or MaxScale + Keepalived |
| PostgreSQL, TimescaleDB | Single-primary replication | 2 (1 primary + 1 replica) | 3 (1 primary + 2 replicas) | HAProxy + Keepalived (PgBouncer optional) |
| MySQL Group Replication, Galera/PXC/Mari Cluster | Multi-primary certification-based clustering | 3 (odd count) | 3 (odd count) | HAProxy or ProxySQL + Keepalived |
| MongoDB | Replica set | 3 (1 primary + 2 secondaries, odd count) | 3 (odd count) | Driver-based (mongos for sharded only) |
| MongoDB | Sharded cluster | 11 total (3 config servers + 2 shards × 3 nodes + 2 mongos for HA) | Same as minimum. Add shards as data grows | 2+ mongos routers |
| Redis, Valkey | Sentinel | 3 (1 primary + 2 replicas, each with Sentinel) | 3 (odd count) | Sentinel-aware client |
| Redis, Valkey | Cluster | 6 (3 masters + 3 replicas) | Same as minimum. Add shards (masters) as write throughput needs grow | Cluster-aware client |
| SQL Server | Always On availability group | 2 (1 primary + 1 secondary replica) | 3 (1 primary + 2 secondary replicas) | Direct-connect to replica IPs (no ClusterControl-managed listener. See *11.5*) |
| Elasticsearch | High availability cluster | 5 (3 master-eligible + 2 data nodes) | 5, scaling data nodes (not master-eligible ones) as load grows | Coordinating node / client access |

**Note**

Note Kubernetes-hosted databases follow a different operational model: ClusterControl connects to your Kubernetes environment via the ClusterControl Kubernetes Proxy instead of managing nodes over SSH. See *Integration → Kubernetes*.

### 11.1 Single-primary replication (MySQL, MariaDB, PostgreSQL, TimescaleDB)



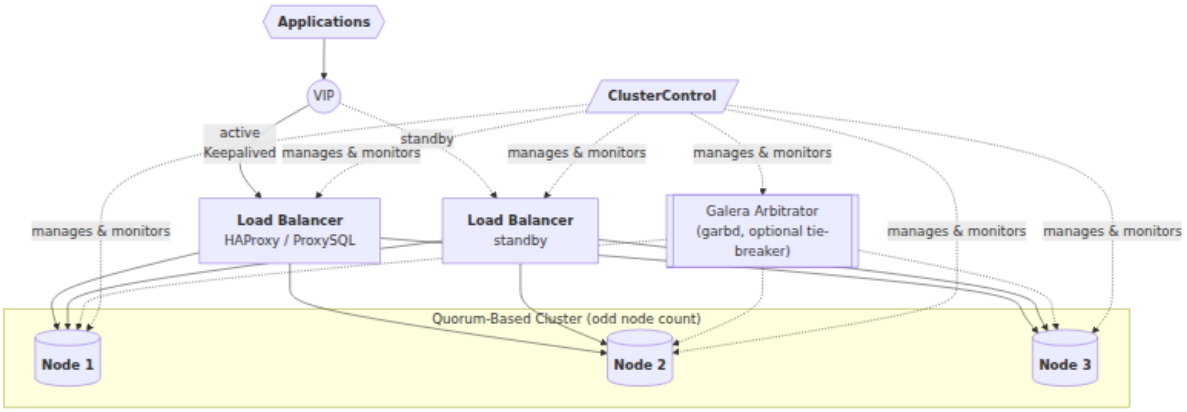

- Start with one replica for read scaling and failover. Add more as read load grows. This pattern scales reads linearly by adding replicas behind the load balancer.
- Route writes and reads through a load balancer rather than the application directly, so failover is transparent. See *Create Load Balancer*.
- Deploy at least two load balancer nodes with Keepalived VIP to avoid the load balancer becoming a single point of failure. The *Percona XtraDB Cluster*, *MySQL Group Replication*, and *PostgreSQL Streaming Replication* Day-1 tutorials all call this out explicitly as the production recommendation, and the *MariaDB Cluster* tutorial deploys it by default. Not every tutorial in this family sets up two load balancers in its minimal example (the logical-replication and TimescaleDB walkthroughs don't), so treat this as the production target regardless of which example you started from.

**Why 2 replicas, not 1, is the production baseline.** The table above lists the bare technical minimum (1 primary + 1 replica = 2 nodes), but size for **3 nodes (1 primary + 2 replicas)** on anything business-critical. When the primary fails, ClusterControl's automatic recovery promotes one replica to primary and reconfigures every *other* replica to follow it. With only 1 replica, that single failover event leaves you with a lone primary and zero replicas until someone manually provisions a new one, no more read scaling and no more failover target. The old primary itself doesn't simply rejoin as a replica either way: for MySQL/MariaDB it comes back up but stays excluded from replication until manually restaged, while for PostgreSQL/TimescaleDB ClusterControl forcefully shuts it down to prevent accidental writes. See the same two Failover and Recovery sections linked above for each mechanism. With 2 replicas, the same

failover event still leaves you with a primary and 1 replica: reads keep working and you still have somewhere to fail over to next time.

- **PostgreSQL/TimescaleDB specifics:** only **HAProxy** is supported as a ClusterControl-managed load balancer here. ProxySQL and MariaDB MaxScale support MySQL, MariaDB, and Galera Cluster only (see *Create Load Balancer → Support Matrix*). Add **PgBouncer** (optional) in front of the primary when connection counts are high. PostgreSQL forks a backend process per connection, so pooling reduces resource consumption more than it would for MySQL/MariaDB's threaded connection model. ClusterControl supports session, transaction, and statement pool modes (see *Create Load Balancer → PgBouncer*). Because ClusterControl surfaces the new primary after a failover but does not automatically redirect existing application connections to it, put HAProxy (or PgBouncer pointed at HAProxy) in front of the cluster rather than connecting the application directly to a node's IP. Otherwise, you have to manually find and re-point to the new primary yourself.
- **MySQL/MariaDB specifics:** ProxySQL and MaxScale add SQL-aware, replication-topology-aware routing (they detect the new writable primary after a failover automatically), while HAProxy does simpler TCP-level health-check-based routing, either works, but ProxySQL/MaxScale need less manual reconfiguration of health-check scripts if you change topology later.
- Consider a **delayed replica** for point-in-time protection against logical mistakes (e.g., accidental `DELETE`/`DROP`).
- Relevant tutorials: *MySQL Replication*, *PostgreSQL Streaming Replication*, *PostgreSQL Logical Replication*, *TimescaleDB Streaming Replication*.

### 11.2 Multi-primary certification-based clustering (Galera, Percona XtraDB Cluster, MariaDB Cluster, MySQL Group Replication)




This diagram shows the shared node/quorum shape; which nodes actually accept writes differs by engine, see **Write routing** below. Galera/PXC/MariaDB Cluster route writes to all three nodes by default. MySQL Group Replication commonly runs single-primary instead, with only one node accepting writes at a time even though all are technically eligible.

- Always use an **odd number of nodes** (3, 5, …) to avoid split-brain when computing quorum.
- When you can only afford two full nodes, deploy `garbd` as a lightweight, vote-only third participant instead of a full database node, on its own host, not the ClusterControl node itself, which is an unsupported placement for `garbd` (see *Create Load Balancer → Garbd*).
- MySQL Group Replication follows the same quorum-based node shape as Galera (odd node count, same arbitration guidance), but differs in write routing, see below.

- ProxySQL and MaxScale add SQL-aware query routing on top of the same node layout, while HAProxy is a simpler Layer-4 router.

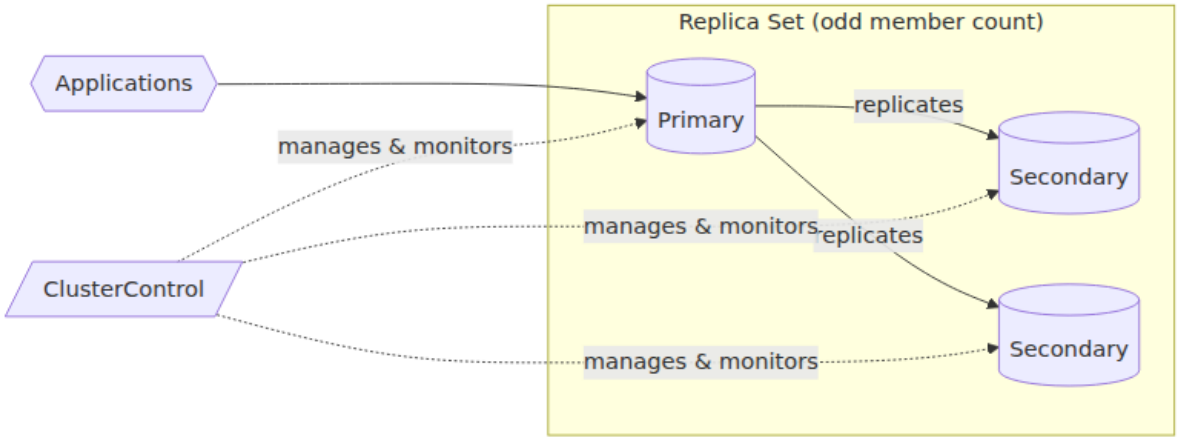
**Write routing: multi-writer vs. single-writer.** Galera/PXC/MariaDB Cluster are certification-based multi-primary systems, every node can accept writes, and ClusterControl's own MariaDB Galera Cluster tutorial deploys HAProxy with read/write splitting turned *off*, load-balancing writes across all three nodes with no host distinction. This is a reasonable default for workloads with little row-level write contention. But conflicts between concurrent writes on different nodes are only caught at commit time (certification), so hot rows or a small, frequently-updated working set see more rollback-and-retry overhead as write concurrency across nodes increases. This same cost is why the *Percona XtraDB Cluster* tutorial recommends restoring a large dataset on a single node first, explicitly because every insert would otherwise be certified by Galera write-set replication, and why the *MariaDB Cluster* tutorial gives the same restore-on-one-node advice, attributed there more generally to the cost of synchronous replication rather than certification by name. If your write pattern has enough contention to matter, pin writes to a single node at a time instead: with HAProxy, set one backend server's **Role** to `active` and the rest to `backup` (see *Create Load Balancer → HAProxy*) so all traffic goes to one node until it fails over to a backup. This is a deliberate configuration choice on top of ClusterControl's defaults (which load-balance across all nodes as shown above), not something it sets up automatically for Galera.

MySQL Group Replication's own tutorial takes the opposite default: it deploys HAProxy with read/write splitting turned *on*, relying on the `mysqlchk_rw_split` health-check script to track whichever single node is currently the read-write primary, consistent with Group Replication's common single-primary operating mode, where only one member accepts writes at a time even though all members are technically eligible.

- Relevant tutorials: *Percona XtraDB Cluster*, *MariaDB Cluster*, *MySQL Group Replication*.

### 11.3 Document store (MongoDB)

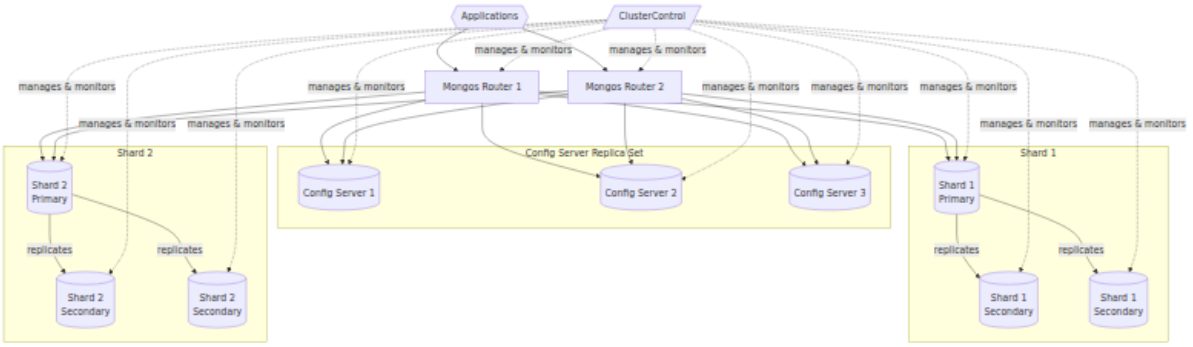
#### Replica set




The MongoDB driver is replica-set aware, so no external load balancer is required. Keep the member count odd, so the set can always compute a majority for primary election (an even count risks a tied vote with no winner). Add a **hidden member**, a full data-bearing member excluded from the driver's default read routing, for reporting/analytics/backup workloads that would otherwise compete with production traffic for resources. Reach for a vote-only **arbiter** only when you can't afford a third full data node, since an arbiter holds no data and can't serve reads, whereas a hidden member does both.

Relevant tutorials: *MongoDB Replica Set*, *MongoDB Enterprise Replica Set*.

### Sharded cluster

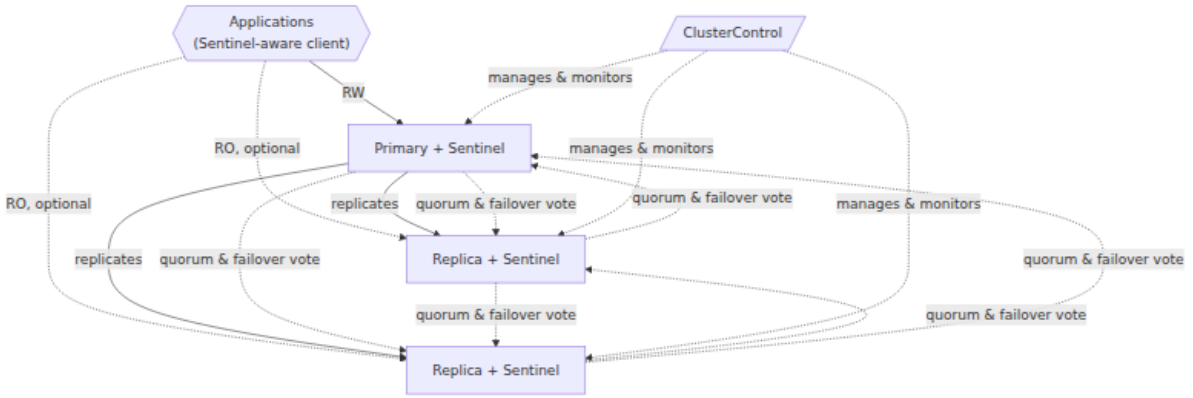



Run the **config server replica set at 3 members**, not 2: an even-numbered replica set can tie during election and fail to produce a primary, and for config servers specifically that means the whole cluster's routing metadata becomes unavailable for writes until it resolves. The same reasoning applies to each shard's own replica set: run **3 nodes per shard**, not 2, so losing a single member still leaves a majority available to elect a primary. Note that the *Day-1 tutorial*'s own minimal example uses only 2 config servers and 1 `mongos` (its prerequisites even allow co-locating that one `mongos` on the ClusterControl node, purely to keep the walkthrough small) rather than the production target below. Treat 3 config servers, 3 nodes per shard, and 2+ `mongos` routers as the production target, not the tutorial's example. Deploy at least two `mongos` routers so applications are not left without a routing path if one fails. As data grows, add shards horizontally. For geographically or tenant-partitioned access patterns, consider zone sharding.

Relevant tutorial: *MongoDB Sharded Cluster*.

## 11.4 In-memory key-value store (Redis, Valkey)

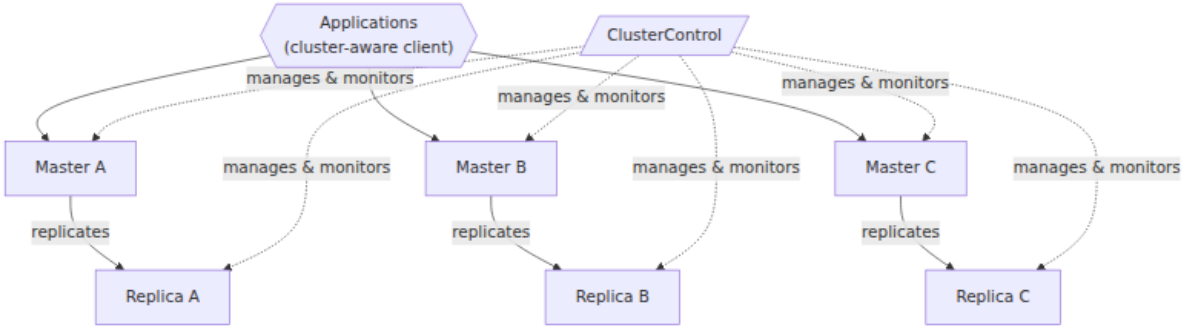
### Sentinel




Use an odd number of Sentinel-monitored nodes, so a quorum can always be reached for automatic failover. Run one Sentinel process **per Redis node, co-located on that same host**. The *Day-1 tutorial*'s topology does this by default (3 nodes, 3 Sentinels) rather than running a separate, centralized pool of Sentinels, so a network partition that isolates one Redis host also isolates its Sentinel's view along with it, keeping quorum voting meaningful. Choose Sentinel when the dataset fits on a single primary.

Relevant tutorials: *Redis Sentinel*, *Valkey Sentinel*.

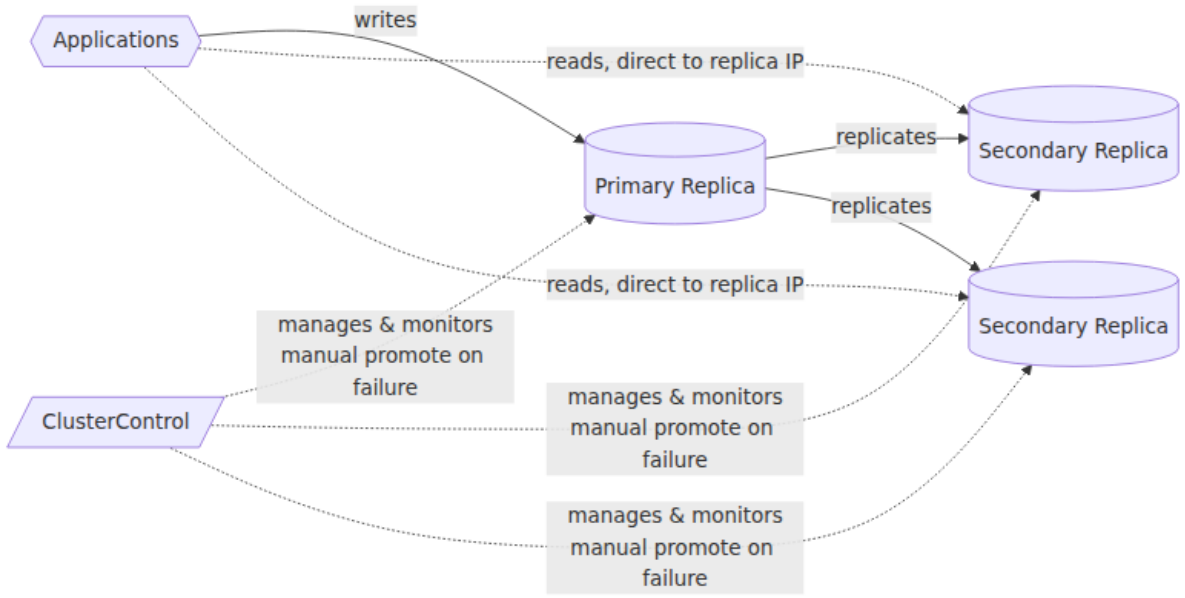
### Cluster




Choose Cluster over Sentinel once the working set outgrows a single primary's memory. Each master should have at least one replica, a masterless shard has no failover target. To scale write throughput, add more shards (masters), not just more replicas to existing shards: ClusterControl's built-in scaling action for this topology only adds a replica to an existing shard, so introducing an additional shard for write scaling is a separate, manual step.

Relevant tutorials: *Redis Cluster*, *Valkey Cluster*.

## 11.5 SQL Server Always On availability group




This topology differs from the others in this section in one important way: ClusterControl's SQL Server support doesn't deploy an AG Listener or automatic quorum-based failover. The *Day-1 tutorial* has applications connect for reads directly to a replica's IP address rather than through a listener, and recovering from a primary failure is a manual **Promote Replica** action in the ClusterControl GUI/CLI, not an automatic failover. If your application needs transparent, listener-based reconnection or SQL Server's own automatic quorum-based failover, set up the AG Listener and the underlying cluster quorum (WSFC on Windows, Pacemaker on Linux) yourself using SQL Server's native tooling, and use ClusterControl for monitoring/management on top of that. This isn't something the deployment wizard configures for you.

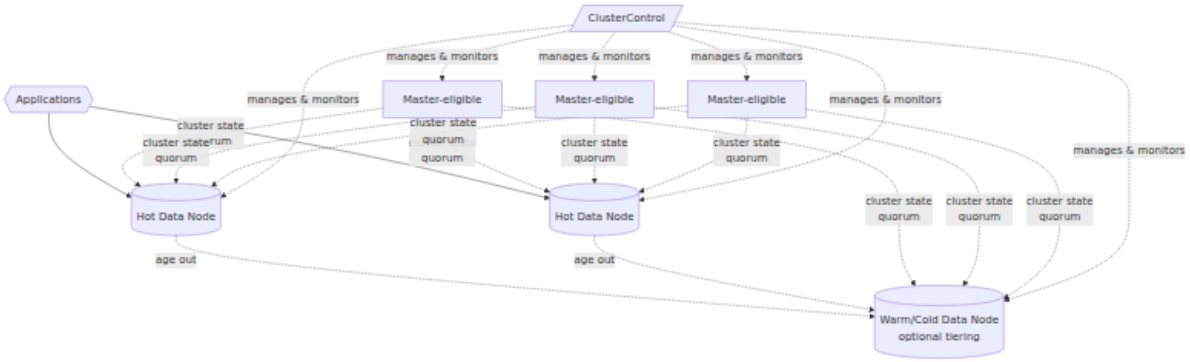
- **Recommended: 3 nodes (1 primary + 2 secondaries), not the 2-node technical minimum**. The same failover-headroom logic as *Section 11.1* applies here too: promoting one secondary should still leave a remaining secondary in place, rather than a lone primary with no replica until one is rebuilt. This also matches the *Day-1 tutorial*'s own example, which deploys 1 primary + 2 secondaries rather

than the bare 1+1 minimum.

- Since failover here is a manual action rather than an automatic one, alert on primary/replica health promptly (see *Alerts & Notifications*) so someone can trigger the promotion quickly rather than relying on it happening on its own.

Relevant tutorials: *Standalone* (not recommended beyond throwaway/dev use, it's a single node with no replica at all), *Always-On Availability Group*.

### 11.6 Search cluster (Elasticsearch)




Separate the three master-eligible nodes from the data nodes so a data-node failure never risks a split-brain election. Keep the master-eligible count odd (3 is the standard baseline) so the cluster can always elect from a majority even if one is unreachable. Scale by adding **data** nodes as ingest/query load grows, not master-eligible ones. The *Single-node Cluster* tutorial itself states plainly that it "isn't recommended for production workloads… use it in production only for low-stakes workloads where downtime and potential data loss are acceptable," and the *High Availability Cluster* tutorial cautions against over-scaling in the other direction too: a larger node count adds its own coordination/processing overhead, so add nodes to match measured load rather than provisioning far ahead of it. Node-level HA is necessary but not sufficient on its own. Also, set each index's `number_of_replicas` to at least 1, so losing a data node doesn't also mean losing a shard's only copy. For larger, time-series-like datasets, consider hot-warm(-cold) tiering. Relevant tutorials: *Single-node Cluster*, *High Availability Cluster*.

**See Also**

- *Getting Started → Requirements*
- *Getting Started → How it Works?*
- *Getting Started → Offline Installation*
- *Getting Started → ClusterControl on Marketplaces*
- *Admin Guide → Redundancy & High Availability*
- *Admin Guide → Tuning for Large Deployment*
- *Admin Guide → Securing ClusterControl*
- *Admin Guide → Migrating ClusterControl to Another Server*
- *User Guide → ClusterControl Ops-C*
- *User Guide → Cloud Providers*
- User Guide → Integration → Hashicorp Vault

- *User Guide → Integration → LDAP*
- *Reference Manuals → Tools → cmon_sd*

## 3.12 Tuning for Large Deployment

Managing hundreds of cluster inside ClusterControl need some special adjustment in the controller node, database, and cmon service.

Below are some configuration you might want to adjust for large scale deployment in ClusterControl.

### 3.12.1 Tuning InnoDB buffer pool size

ClusterControl components rely on a MySQL or MariaDB database as the persistent store for monitoring data collected from the managed nodes and all ClusterControl metadata (e.g., what jobs there are in the queue, backup schedules, backup statuses, etc.). By default, the installer script will install whatever version comes by the standard repository of the OS.

The installer script would do some basic tuning like configuring the datadir location, MySQL port, user, and InnoDB buffer pool size at the beginning of the installation stage. However, the tuning might not be suitable once you have imported or created more clusters/nodes. With an increased number of nodes to be monitored and managed, ClusterControl would use more resources, and the database layer is commonly the first bottleneck that users encounter. Some further tuning might be needed to contain the incoming load.

Buffer pool size is an important component and has to be configured upfront to improve the MySQL performance. It allows MySQL processing to be happening inside memory instead of hitting the disk. A simple rule of thumb is to check the InnoDB hit ratio and look for the following line under BUFFER POOL AND MEMORY section:

```
mysql> SHOW ENGINE INNODB STATUS\G

...
Buffer pool hit rate 986 / 1000
...
```

The hit rate of `986 / 1000` indicates that out of 1000 row reads, it was able to read the row in RAM 986 times. The remaining 14 times, it had to read the row of data from disk. Simply said, `1000 / 1000` is the best value that we are trying to achieve here, which means the frequently-accessed data fits fully in RAM.

Increasing the `innodb_buffer_pool_size` value will help a lot to accomodate more room for MySQL to work on. However, ensure you have sufficient RAM resources beforehand. By default, ClusterControl allocates 50% of the RAM to the buffer pool. If the host is dedicated to ClusterControl, you can even push it to a higher value like 70%, provided you spare at least 2GB of RAM to the OS processes and caches. If you can't allocate that much, increasing the RAM is the only solution.

Changing this value requires a MySQL restart (older than MySQL 5.7.5), thus the correct service restart ordering will be:

1. Modify `innodb_buffer_pool_size` value inside `my.cnf`.
2. Stop the CMON service.
3. Restart MySQL/MariaDB service.

4. Start the CMON service.

Or simply reboot the host if you can afford a longer downtime.

### 3.12.2 Using MySQL socket file

By default, the installer script will automatically configure the following host value inside every ClusterControl database configuration files:

| Configuration File | Value |
|---|---|
| `/etc/cmon.cnf` | `mysql_hostname=127.0.0.1` |
| `/etc/cmon.d/cmon_X.cnf` (X is the cluster ID) | `mysql_hostname=127.0.0.1` |

The above will force the MySQL client to connect via TCP networking, just like connecting to a remote MySQL host although ClusterControl is running on the same server as the MySQL server. We purposely configured it this way to simplify the installation process since almost every OS platform pre-configures MySQL socket file differently, which greatly reduces the installation failure rate.

Changing the value to "localhost" will force the client to use the MySQL Unix socket file instead:

| Configuration File | Value |
|---|---|
| `/etc/cmon.cnf` | `mysql_hostname=localhost` |
| `/etc/cmon.d/cmon_X.cnf` (X is the cluster ID) | `mysql_hostname=localhost` |

On Unix based systems, MySQL programs treat the host name localhost specially, in a way that is likely different from what you expect compared to other network-based programs. For connections to localhost, MySQL programs attempt to connect to the local server by using a Unix socket file. This occurs even if a –port or -P option is given to specify a port number.

Using MySQL UNIX socket file is much more secure and will cut off the network overhead. It is always recommended over TCP. However, you need to make sure the socket file is configured correctly. It must exist on the following directives inside `my.cnf` and every MySQL option files on ClusterControl node, for example:

```
[mysqld]
socket=/var/lib/mysql/mysql.sock

[client]
socket=/var/lib/mysql/mysql.sock

[mysql]
socket=/var/lib/mysql/mysql.sock

[mysqldump]
socket=/var/lib/mysql/mysql.sock
```

Changing the socket file path in MySQL will also require a MySQL and CMON restart.

### 3.12.3 CMON pool size

Tune the pool size in `/etc/cmon.conf`, the default connection pool size for cmon is 64. You might want to increase the pool size in `cmon.conf` configurations depend on how much the cluster you have. You can increase 1,5x - 2x from the cluster size.

```
cmon_pool_size=128
```

Restart ClusterControl controller (cmon) service to load the change:

```
systemctl restart cmon
```

### 3.12.4 File descriptors

In some huge database farm environment, there are some errors related to the file descriptors with message "too many open files" on the Prometheus. Tune the file descriptors in `/etc/sysctl.conf` by adding or increasing the `fs.file-max` parameter:

```
fs.file-max=1048576
```

Tune soft and hard limits in `/etc/security/limits.conf`, and set the soft/hard limits too in limits.conf related to open files:

```
*   soft    nproc       1024000
*   hard    nproc       1024000
*   soft    nofile      1024000
*   hard    nofile      1024000
```

Save the file and re-login the terminal session. Verify with:

```
ulimit -a
```

### 3.12.5 OpenSSH with SSSD

ClusterControl uses SSH protocol as its main communication channel to manage and monitor remote nodes. The default OpenSSH configurations are pretty decent and should work in most cases. However, in some environments where SSH is integrated with other security enhancement tools like SElinux or System Security Services Daemon (SSSD), it could bring significant impact to the SSH performance.

We have seen many cases where an ever increasing amount of SSH connections established to each of the managed nodes and eventually, both the ClusterControl server and all managed nodes max out their system memory with SSH connections. In some cases, only a normal full system reboot nightly on the ClusterControl server could solve the problem.

If you running your infrastructure with System Security Services Daemon (SSSD), it's advised for you to comment the following line inside OpenSSH client configuration at `/etc/ssh/ssh_config` on ClusterControl node:

```
#ProxyCommand /usr/bin/sss_ssh_knownhostsproxy -p %p %h
```

The above will skip using SSSD to manage the host key, which will be handled by OpenSSH client instead.

### 3.12.6 File System and Partitioning

ClusterControl controller writes new entry in its log file almost every second for every cluster. For those who wants to take advantage of this sequential writes on disk and would like to save cost, you can use a spindle disk for this purpose. Modify the following line inside all cmon_X.cnf (where X is the cluster ID):

```
logfile=/new/partition/log/cmon_X.log
```

Restart CMON service to apply the change:

```
systemctl restart cmon
```

If you are using ClusterControl as the backup repository, it's recommended for you to allocate sufficient disk space on a separate partition other than the root partition. It gets better if the partition resides on a networked or clustered file system for easy mounting with the targeted nodes when performing restoration operation. We have seen cases where the created backups ate up all disk space of the main partition, eventually impacting ClusterControl and its components.

## 3.13 Prometheus storage retention

ClusterControl uses Prometheus to store time-series monitoring data. Prometheus is designed to store high-resolution time-series data, which can accumulate quickly. By default, the storage retention is set to 15 days with 10 seconds scrape interval. To put into perspective, with 5 database nodes (each has at least 3 exporters' endpoints), this can accumulate up to 5 GB total disk size (check the disk usage of `/var/lib/prometheus`).

You can adjust the Prometheus storage retention and scrape interval by going to ***ClusterControl GUI → Clusters → choose a cluster → Dashboards → More... → Configuration → Edit***. Click **Save** and ClusterControl will reconfigure the agent-based monitoring (together with Prometheus reconfiguration) for the chosen cluster.

## 3.14 CMON data history

ClusterControl monitoring data will be purged based on the value set at ***ClusterControl GUI → Clusters → choose a cluster → Settings → System Settings → System → save_history_days*** (default is 7 days). Some users might find this value to be too low for auditing purposes. You can increase the value accordingly however, the longer collected data exist in the CMON database, the bigger space it needs. It is recommended to lower the disk space threshold under ***ClusterControl GUI → Clusters → choose a cluster → Settings → System Settings → Thresholds → diskspace*** so you will get an early warning in case the CMON database grows significantly.

Alternatively, you can modify the cmon configuration file (`/etc/cmon.d/cmon_X.cnf`, where X is the cluster id) and set `save_history_days` and pass how many days of the history you would like to keep. Please keep in mind you have to restart the cmon service to apply the change.

If you have configured a very short interval recurring jobs like backup jobs running every hour, it would produce lots of job activities. We suggest lowering the controller job history retention period, to like 1 or 2 days. It can be done by setting `save_history_days=1` which mean the controller only keeps the last two days of job history.

## 3.15 Log rotation

The `cmon` process has internal log rotation scheduling where it will log up to 5 MB in size before archiving `/var/log/cmon.log` and `/var/log/cmon_{cluster ID}.log`. The archived log will be named as cmon.log.1 (or cmon_{cluster ID}.log.1) sequentially, with up to 9 archived log files (total of 10 log file rotation).

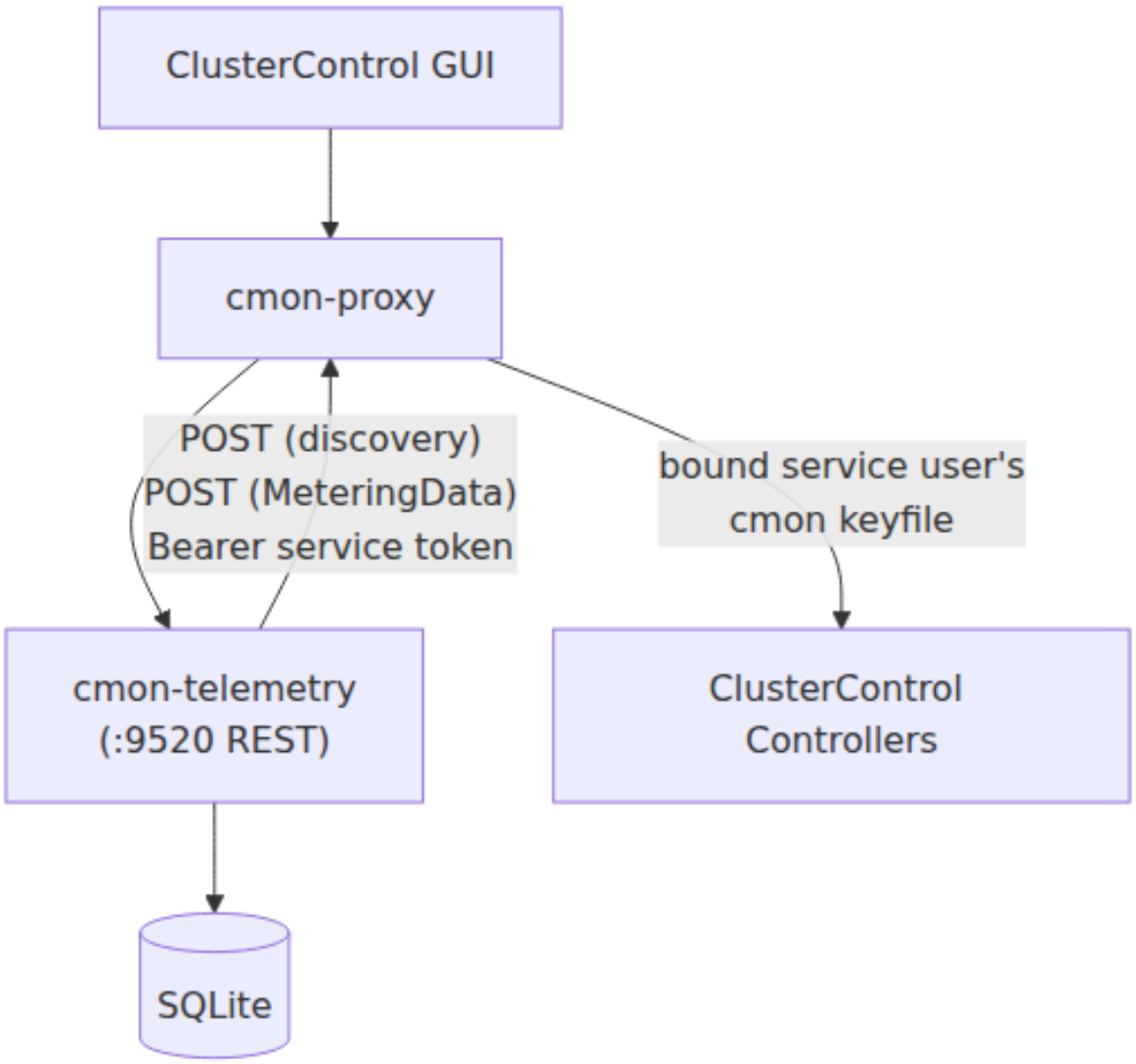
ClusterControl Usage Metering provides billing-grade accounting of the database-server estate under management. For every collection tick, it records one snapshot per eligible node (hardware class, RAM, vCPU, disk, tags). At report time, it computes per-period high-watermarks, a filtered or estate-wide summary, and a cryptographic seal that makes the report tamper-evident. These sealed reports can back invoices, tenant chargebacks, and audit trails. A typical use case is a Managed Service Provider (MSP) reporting its billable estate.

Metering runs in a standalone service, `cmon-telemetry`. It polls ClusterControl (via `cmon-proxy`) on a configurable interval, stores per-node snapshots in a local SQLite database, and seals each generated report with two signatures: HMAC-SHA256 (internal integrity) and Ed25519 (third-party verifiable operator signature). A collection tick is just two HTTP calls and a database write; there is no streaming pipeline to operate.

**Note**

Note Usage metering requires **ClusterControl 2.5.0 or newer**, with `cmon-proxy`, `cmon-telemetry`, and the ClusterControl controller all upgraded together.

## 3.16 Architecture at a glance




- **`cmon-telemetry`** (the service) polls `cmon-proxy` each tick for snapshots, stores them in SQLite, produces sealed reports on demand, and exposes a REST API on port `9520`.
- **`cmon-proxy`** (the credential gate) holds the controller-side RSA keyfiles in `service_users[]` and issues a Bearer service token for `cmon-telemetry` to call back through. Calls to the controllers happen as the bound service user, not as any operator session.
- **Billing UI** lives at `/billing` in the ClusterControl Ops-C web application (with multi-controller enabled). See *ClusterControl Ops-C*. It is the operator interface to generate, view, verify, download, and delete reports.

## 3.17 Prerequisites

| Component | Requirement |
|---|---|
| ClusterControl | 2.5.0 or newer (`cmon-proxy` + `cmon-telemetry` + controller upgraded together) |
| `clustercon` package | Installed; typically on the same host as `cmon-proxy` (a collocated install is the documented path) |
| `cmon-proxy` configuration | `billing_enabled: true` in `ccmgr.yaml`; service user + token provisioned via `ccmgradm serviceauth-bootstrap` (see below); `cmon_telemetry_url` + `cmon_telemetry_token` set in `ccmgr.yaml` so the Billing UI can reach `cmon-telemetry` (see *Wire the Billing UI passthrough*) |
| Ports | `9520` inbound on the `cmon-telemetry` host (REST). Outbound from `cmon-telemetry` to `cmon-proxy` (`/proxy/*` and `/v2/*`) on whichever port `cmon-proxy` listens on: `19051` is `cmon-proxy`'s own default, while a standard ClusterControl installation fronts it on `443`. |
| Operating system | Debian/Ubuntu or RHEL-based distributions (DEB/RPM packages) |
| Go runtime | None; `cmon-telemetry` is a single static binary |

## 3.18 Installing cmon-telemetry

Pick **one** host. It can be the same host as `cmon-proxy` for collocated installs, or a separate host for split deployments.

### Debian/Ubuntu

```
sudo apt-get update
sudo apt-get install -y clustercontrol-telemetry
```

### RHEL/CentOS/Rocky Linux/AlmaLinux

```
sudo yum install -y clustercontrol-telemetry
```

The package lays down:

- `/usr/bin/cmon-telemetry`: the binary
- `/etc/cmon-telemetry/config.yaml`: configuration template (preserved on upgrade)
- `/var/lib/cmon-telemetry/`: SQLite data directory and the default home for the Ed25519 key file
- `/lib/systemd/system/cmon-telemetry.service`: systemd unit

### 3.18.1 Generate a signing key and Bearer token

```
# Signing key for sealed reports (SHA-256 + HMAC)
SIGNING_KEY=$(openssl rand -hex 32)
echo "Signing key: $SIGNING_KEY"
```

(continues on next page)

(continued from previous page)

```
# Bearer token for REST auth
API_TOKEN=$(openssl rand -hex 24)
echo "API token:   $API_TOKEN"
```

Keep both values safe; you will paste them into the configuration file below. The signing key only needs to survive the retention window (12 months by default); the Bearer token can be rotated at any time.

!!! important "The API token is used in two places"

```
`API_TOKEN` goes into **both** of these, and the two values must match:

- `api_token` in `cmon-telemetry`'s `/etc/cmon-telemetry/config.yaml` (next␣
↪section), and
- `cmon_telemetry_token` in `cmon-proxy`'s `ccmgr.yaml` (see [Wire the␣
↪Billing UI passthrough](#wire-the-billing-ui-passthrough)).

The first one makes `cmon-telemetry` demand the token; the second one makes␣
↪`cmon-proxy` present it. The Billing page does not work without the␣
↪second one.
```

**Note**

Note You do **not** generate the Ed25519 keypair by hand. `cmon-telemetry` writes a fresh Ed25519 keypair to `ed25519_key_file` (default `/var/lib/cmon-telemetry/signing.ed25519`) on first start, with permissions 0600. The public-key fingerprint then shows up in the startup log and on the `/billing/status` endpoint.

### 3.18.2 Write the configuration file

Edit `/etc/cmon-telemetry/config.yaml`:

```
# REST listen socket; Billing UI + curl /billing/* hit here
api_listen: "127.0.0.1:9520"

# Storage
db_path: /var/lib/cmon-telemetry/metering.db
retention_months: 12
log_level: "info"
# logfile: ""             # Empty = stdout/stderr (journald). Set to a path␣
↪for on-disk logging.

# --- Internal integrity seal (HMAC-SHA256, symmetric) ----------------
signing_key: "<SIGNING_KEY generated above>"
key_id:      "key-2026-Q2"
verification_keys:
  key-2026-Q2: "<same SIGNING_KEY>"
  # Add previous keys here during rotation; the list is used by verifyReport
```

(continues on next page)

(continued from previous page)

```
# --- Third-party verifiable export (Ed25519, asymmetric) -------------
# Auto-generated on first start if the file is missing. Never share the
# file; only the public-key fingerprint (visible in /billing/status + UI).
ed25519_key_file: /var/lib/cmon-telemetry/signing.ed25519

# --- REST auth -------------------------------------------------------
# Required for /billing/{reports,snapshots,controllers/status}.
# /billing/status stays open as a health probe.
api_token: "<API_TOKEN generated above>"
# api_tls_cert: /etc/cmon-telemetry/api-server.crt
# api_tls_key:  /etc/cmon-telemetry/api-server.key

# --- Billing collection ----------------------------------------------
# cmon-telemetry holds NO controller credentials. All traffic flows
# through cmon-proxy. Provision cmon-proxy's side as described in the
# next section; the bootstrap drops the token at the default token_file
# path so a collocated install needs zero pasting.
billing:
  enabled: true
  cmon_proxy:
    url: "https://localhost"                # add an explicit :port if␣
↪cmon-proxy is not on 443
    insecure: false                         # set true ONLY for testing␣
↪with self-signed certs
    # token: "<paste-if-not-using-token_file>"
    token_file: /usr/share/ccmgr/service-tokens/cmon-telemetry
  interval:               60m               # production default
  parallelism:            4
  per_controller_timeout: 30s
  period_months:          1                 # default report period when␣
↪generateReport is called without dates
  min_active_hours:       24                # nodes must clear this to␣
↪count as billable
```

### 3.18.3 Start the service

```
sudo systemctl enable --now cmon-telemetry
sudo systemctl status cmon-telemetry
```

Expected output ends with `active (running)`.

Check the journal for the keypair line on first start:

```
sudo journalctl -u cmon-telemetry | grep -i ed25519 | head -2
# generated Ed25519 keypair: fingerprint=sha256:27dccd92548741af,
#   file=/var/lib/cmon-telemetry/signing.ed25519,␣
↪public_key=962d9fb98a3ed1c2...
```

On subsequent restarts, the line reads `loaded Ed25519 keypair: fingerprint=sha256:....` The fingerprint is what you share with the party that verifies your exported reports (see *Report seals and*

*verification*).

Verify the service is listening:

```
sudo ss -tlnp | grep -E ':9520\s'
# LISTEN … 127.0.0.1:9520 …
```

## 3.19 Provisioning the cmon-proxy service token

`cmon-telemetry` needs a Bearer token to call `cmon-proxy`. The token resolves on `cmon-proxy`'s side to a `service_user` (a controller-side RSA keypair) that authenticates against each controller.

There are two provisioning paths (choose either one):

- *Path A* is the standard
- *Path B* is for manual or scripted installs.

### 3.19.1 Path A: `ccmgradm serviceauth-bootstrap` (recommended)

This provisions everything end to end, where it generates an RSA keypair for the controller-side service user, registers the public key on each controller listed in `instances:` via the `createUser` RPC, writes the `service_users[]` + `service_tokens[]` entries to `ccmgr.yaml`, and drops the token at the default location `cmon-telemetry` reads from.

Run on the `cmon-proxy` host, as root (so the generated files end up owned by the daemon user). The following is an example if the ClusterControl admin user is called "ccadmin" with password 'SuperStrongP455':

```
CCMGR_ADMIN_PASSWORD='SuperStrongP455' \
  sudo -E ccmgradm serviceauth-bootstrap --name cmon-telemetry --admin-user␣
↪ccadmin
```

The environment-variable form keeps the password off the shell-history line. The admin credentials are required at bootstrap time only and are never persisted to `ccmgr.yaml`.

Per-controller outcomes print at the end (✓ `created` / · `user already existed (pubkey re-registered)` / ✗ `error`). The token is printed once with a "save this" note; `cmon-telemetry`'s default `token_file` (`/usr/share/ccmgr/service-tokens/cmon-telemetry`) reads it automatically, so no pasting is needed on a collocated install.

```
sudo systemctl restart cmon-proxy    # reload the new ccmgr.yaml
```

**Warning**

Multi-controller pools

Bootstrap iterates the primary URLs in `instances:`. The controller replicates the user record across pool members, but per-user public keys do not propagate. On a multi-controller pool, only the primary's metering fetch succeeds initially. Workaround: re-run bootstrap after adding or replacing pool members:

```
ccmgradm serviceauth-bootstrap --name cmon-telemetry --admin-user admin
# then on each controller pool member, if the user got suspended during␣
↪the gap:
s9s user --unsuspend cmon-telemetry
```

### 3.19.2 Path B: manual via `s9s`

For deployments without `ccmgradm`, or when you want explicit control over the keypair and `ccmgr.yaml` edits:

```
# 1. On each controller in the pool: create a dedicated service user and a␣
↪fresh keypair.
sudo -u cmon s9s user --create --generate-key --group=admins cmon-telemetry
# This writes the private key to /home/cmon/.s9s/cmon-telemetry.key and␣
↪registers
# the public half against the controller-side user. Repeat on every pool␣
↪member,
# since per-user public keys do not propagate (see the multi-controller␣
↪caveat above).

# 2. Copy the private key to a location cmon-proxy can read.
sudo install -m 640 -o s9s_cc -g severalnines \
  /home/cmon/.s9s/cmon-telemetry.key /
↪usr/share/ccmgr/service-users/cmon-telemetry.key

# 3. Generate a Bearer token and write the hint file.
TOKEN=$(openssl rand -hex 32)
sudo install -m 640 -o s9s_cc -g severalnines \
  /dev/stdin /usr/share/ccmgr/service-tokens/cmon-telemetry <<< "$TOKEN"
```

Then add two blocks to `/usr/share/ccmgr/ccmgr.yaml`:

```
service_users:
  - name: cmon-telemetry
    cmon_username: cmon-telemetry
    cmon_keyfile: /usr/share/ccmgr/service-users/cmon-telemetry.key

service_tokens:
  - name: cmon-telemetry
    token: <the $TOKEN from step 3>
    service_user: cmon-telemetry
    endpoints:
      - path: /proxy/controllers/status
        methods: [GET, POST]
      - path: /v2/clusters
        methods: [POST]
```

Followed by restarting the `cmon-proxy` service:

```
sudo systemctl restart cmon-proxy
```

### 3.19.3 Verify provisioning landed

```
sudo journalctl -u cmon-telemetry -n 100 | grep '\[collect\]'
# expect: tick start controllers=N captured=...
#         (per-controller lines, no auth errors)

curl -sf -H "Authorization: Bearer $API_TOKEN" \
     http://localhost:9520/billing/controllers/status | jq '.outcome,␣
↪(.results[] | {name, status, error})'
# expect: "outcome": "ok", every result status="ok" or "empty"
```

If `controllers/status` shows `error` rows with `"AccessDenied: User cmon-telemetry is suspended..."`, see the multi-controller caveat above and the troubleshooting table below.

## 3.20 Wire the Billing UI passthrough

This step is required for the Billing page to work. The Billing UI reaches `cmon-telemetry` through `cmon-proxy` (`/proxy/telemetry/billing/*`), the reverse direction of the service token provisioned above: `cmon-proxy` must present `cmon-telemetry`'s API token as a Bearer header on those calls. This is the same `API_TOKEN` you generated in *Generate a signing key and Bearer token* and set as `api_token` in `/etc/cmon-telemetry/config.yaml`.

Neither `ccmgradm serviceauth-bootstrap` nor the package sets this up; add it by hand to `/usr/share/ccmgr/ccmgr.yaml`:

```
billing_enabled: true
cmon_telemetry_url: http://localhost:9520
cmon_telemetry_token: "<API_TOKEN generated earlier>"
```

The three keys do the following:

| Key | Purpose |
|---|---|
| `billing_e` | Gates the **Billing** menu entry and the `/billing` route in the UI. When `false`, a direct link to `/billing` redirects to the overview page. |
| `cmon_tele` | Where `cmon-proxy` forwards `/proxy/telemetry/billing/*` calls. Defaults to `http://localhost:9520` when unset, which is correct for a collocated install. Point it at the `cmon-telemetry` host for a split deployment. |
| `cmon_tele` | The Bearer token `cmon-proxy` presents to `cmon-telemetry`. Must match `api_token` in `/etc/cmon-telemetry/config.yaml`. No default; leave it unset only if `api_token` is also unset. |

Restart cmon-proxy service:

```
sudo systemctl restart cmon-proxy
```

If you no longer have the token in your shell, you can retrieve the `API_TOKEN` value from `cmon-telemetry` configuration file:

```
sudo grep api_token /etc/cmon-telemetry/config.yaml
```

**Warning**

Attention

If `cmon_telemetry_token` is missing or doesn't match `api_token`, every billing call returns 401. The current UI reacts to this by logging the operator out the moment **Billing** is clicked, with this error in the `cmon-proxy` log: `Failed to get JWT token: auth check request failed with status: 401`.

**Warning**

Environment variables override these YAML keys

`cmon-proxy` reads a set of environment variables (typically from `/etc/default/cmon-proxy.env`) that **take precedence over** the `ccmgr.yaml` values above. For historical reasons their names use a `CC_TELEMETRY_` prefix even though the YAML keys are `cmon_telemetry_*`:

| Environment variable | Overrides |
|---|---|
| `CC_TELEMETRY_URL` | `cmon_telemetry_url` |
| `CC_TELEMETRY_TOKEN` | `cmon_telemetry_token` |

If a change to `ccmgr.yaml` appears to have no effect, check `/etc/default/cmon-proxy.env` for a stale value. New deployments should standardise on the YAML keys and leave these unset.

## 3.21 Verifying the pipeline end to end

Wait one `billing.interval` tick (default 60 minutes) after starting `cmon-telemetry` and provisioning the token, then:

```
# Health probe, no auth
curl -sf http://localhost:9520/billing/status | jq .
```

Expected (trimmed):

```
{
  "collector_running": true,
  "collection_healthy": true,
  "health_status": "ok",
  "total_snapshots": 22,
  "billing_period_months": 1,
  "min_active_hours": 24,
  "retention_months": 12,
  "ed25519_public_key":
↪"962d9fb98a3ed1c24406bcd7a94637d2f15bb3dd802d5ad87c666e6a7077bb5f",
  "ed25519_fingerprint": "sha256:27dccd92548741af"
```

(continues on next page)

(continued from previous page)

```
}
```

Record the fingerprint (`sha256:…`) somewhere you can reach later; the party verifying your exported reports will cross-check it against any bundle you send them.

Deeper checks (both require authentication):

```
# Live per-controller fetch outcome (in-memory state of the last collector␣
↪tick)
curl -sf -H "Authorization: Bearer $API_TOKEN" \
     http://localhost:9520/billing/controllers/status | jq .

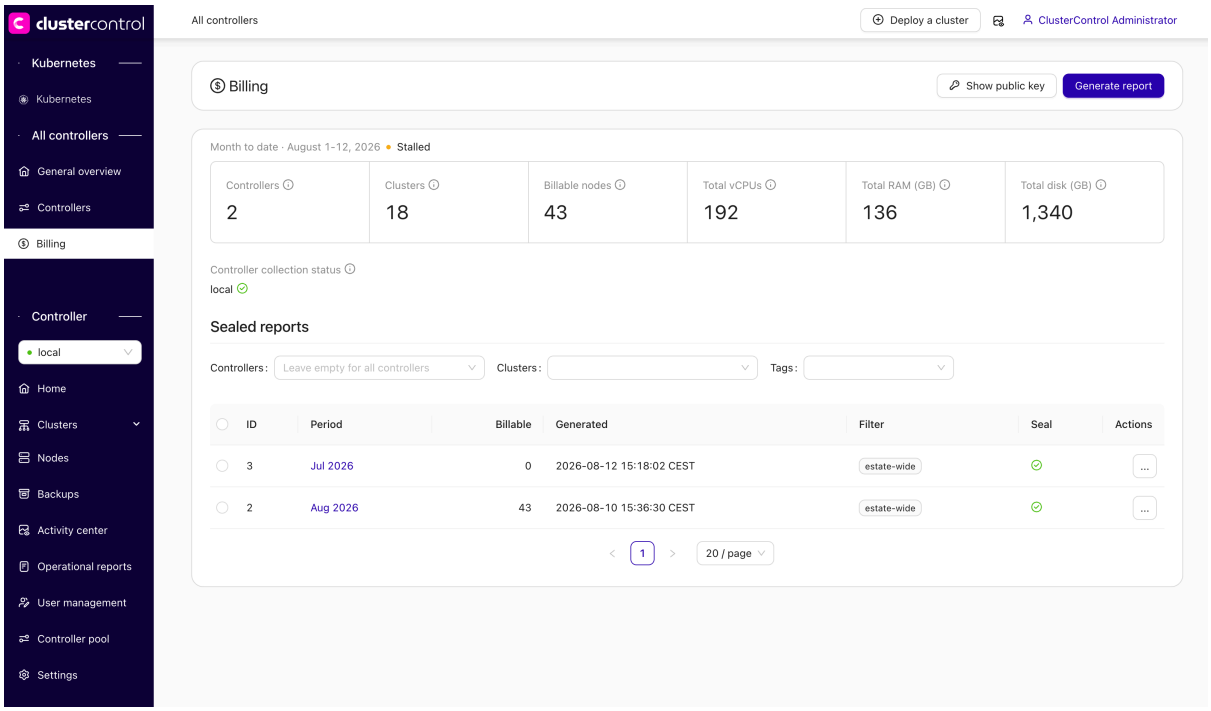
# Paginated raw snapshot read (the event store cmon-telemetry writes each␣
↪tick)
curl -sf -H "Authorization: Bearer $API_TOKEN" -H "Content-Type:␣
↪application/json" \
     -X POST http://localhost:9520/billing/snapshots \
     -d '{"limit": 5}' | jq '.total, .snapshots[0]'
```

### 3.21.1 Troubleshooting if data is not flowing

| Symptom | Check |
|---|---|
| `total_snapshots: 0` after more than one tick | `curl /billing/controllers/status`: `results[]` shows the per-controller outcome with the verbatim controller error string. Audit the token allow-list and service-user provisioning. |
| `outcome: "discovery_error"` on `/billing/controllers/status` | `cmon-telemetry` cannot reach `cmon-proxy`. Check `billing.cmon_proxy.url`, the token in `token_file`, and the service token's `endpoints:` allow-list in `ccmgr.yaml`. |
| Some controllers report `status: "empty"` while others produce snapshots | Per-controller. `empty` means the fetch succeeded but the controller owned no clusters this tick, common in multi-controller pools where ownership migrates. Not an error. |
| Some controllers report `status: "error"` with "User cmon-telemetry is suspended…" | Pool-member key propagation gap (see the multi-controller caveat). Re-run `ccmgradm serviceauth-bootstrap`, then `s9s user --unsuspend cmon-telemetry` on each controller that locked out. |
| `collection_healthy: false` | `journalctl -u cmon-telemetry -f`: collection errors, disk full, corrupted database. |
| `401 Unauthorized` from `/billing/reports` | Bearer token mismatch with `api_token` in `/etc/cmon-telemetry/config.yaml`. `/billing/status` stays open (health probe); other endpoints require the token. |
| `ed25519_fingerprint` missing from `/billing/status` | You are running a pre-2.5.0 binary. Reinstall `clustercontrol-telemetry`. |

## 3.22 The Billing page

Open the ClusterControl Ops-C GUI and click **Billing** in the left navigation (visible only when `billing_enabled: true` on `cmon-proxy`).




The page has four logical blocks, top to bottom:

- **Current period card.** Live high-watermarks for the in-progress billing period (billable nodes, total vCPUs, total RAM, total disk). Totals update as new snapshots arrive. The card header holds the **Show public key** and **Generate report** buttons.
- **Sealed reports heading and filter bar.** The **Clusters** and **Tags** selects narrow the reports table. Matching is strict: rows without filter descriptors are hidden when a selection is active.
- **Sealed reports table.** One row per sealed report, sorted newest-sealed first:
  - **Period**: the billing period the report covers (for example, `Apr 2026`).
  - **Billable**: nodes that met `min_active_hours` during the period.
  - **Generated**: when the report was sealed, in your browser timezone.
  - **Filter**: `estate-wide` or descriptor chips (for example, `tags: customer-acme` or `cluster_ids: 3,2`).
  - **Seal**: a check mark means the report carries a valid cryptographic seal.
  - **Actions**: a menu with **View details** and **Delete**.
- **Detail drawer.** Slides in from the right when a row is clicked. The seal status (`Sealed` or `Seal invalid`) reflects the `hash_valid` / `signature_valid` flags returned by the service. For a deeper check, export the CSV bundle and run `cmon-report-verify` (see below).

### 3.22.1 The Show public key button

Click the key-icon **Show public key** button to open a modal with:

- **Fingerprint** (for example, `sha256:27dccd92548741af`) with a copy-to-clipboard button. This is the short string you share with whoever verifies your reports.
- **Public key (hex)**: the full 64-character hex, also copyable. Useful for automated verification workflows.

Both values come straight from `/billing/status`. The private key lives only in `ed25519_key_file` on the `cmon-telemetry` host and never surfaces in the UI, API, or logs.

## 3.23 Generating a report

Click **Generate report** on the current-period card.

1. **Pick the period.** Monthly billing periods are the default; pick the first and last day of the month. Shorter periods work, but billable counts will be 0 if nodes do not accumulate `min_active_hours`.
2. **Scope (optional).** Leave **Clusters** and **Tags** empty for an estate-wide report. Pick specific clusters or tags to scope to a per-customer or per-tenant report. Tags are free-form (for example, `customer-acme`).
3. **Force regenerate.** Off by default. The service returns `409 Conflict` when an estate-wide report already exists for the period, with the existing `report_id` echoed in the body. Flip it on to overwrite (a new report version is sealed; the previous version stays in the table).
4. **Generate.** On success, the page switches back to the table with the fresh row prepended.

## 3.24 Viewing a report

Click any row in the Sealed reports table, or choose **View details** from the row's actions menu. Things worth knowing:

- **Timestamps** render as `YYYY-MM-DD HH:MM:SS TZ` in the operator's local timezone.
- **Total vCPUs / RAM / disk** are period high-watermarks (peak concurrent usage across billable nodes), not simple sums.
- **Seal tag.** Green `Sealed` means the stored `hash_valid` and `signature_valid` flags are both true. Red `Seal invalid` means either the report's JSON body was altered after sealing or the current `verification_keys` list does not cover the report's `signing_key_id`. For a deeper, third-party-grade check, use the CSV export with `cmon-report-verify`.

**Example report**

Report #2

| | | | |
|---|---|---|---|
| Period | 2026-08-01 02:00:00 CEST → 2026-08-11 01:59:59 CEST | Generated | 2026-08-10 15:36:30 CEST |
| Seal | Sealed | Billable nodes | 43 |
| Total vCPUs | 192 | Total RAM (GB) | 136 |
| Total disk (GB) | 1340 | | |

By type and vendor

| Cluster type | DB vendor | Max concurrent | Max vCPU | Max RAM (GB) | Max volume (GB) |
|---|---|---|---|---|---|
| GALERA | mariadb | 9 | 8 | 4 | 30 |
| GALERA | percona | 5 | 4 | 4 | 30 |
| GROUP_REPL | oracle | 3 | 4 | 2 | 30 |
| MONGODB | mongodb | 3 | 4 | 4 | 30 |
| MSSQL_AO_ASYNC | microsoft | 2 | 4 | 4 | 80 |
| POSTGRESQL_SINGLE | postgresql | 3 | 4 | 4 | 30 |
| REDIS | redis | 4 | 4 | 4 | 30 |
| REPLICATION | mariadb | 3 | 4 | 4 | 30 |
| REPLICATION | oracle | 5 | 4 | 4 | 30 |
| REPLICATION | percona | 3 | 8 | 4 | 30 |
| VALKEY_SENTINEL | valkey | 1 | 2 | 2 | 30 |

Node details

| Node | Cluster | Vendor | Role | Hours | Max vCPU | Max RAM (MB) | Max vol (GB) |
|---|---|---|---|---|---|---|---|
| d3t4k9t4q1ardrm415hg:192.168.73.101 | Valkey8 | valkey | database | 58 | 2 | 2048 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.11 | PXC 8.4 | percona | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.12 | PXC 8.4 | percona | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.13 | PXC 8.4 | percona | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.14 | PXC 8.4 | percona | proxysql | 58 | 4 | 2048 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.15 | PXC 8.4 | percona | proxysql | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.34 | MySQL Ora Rep 80 | oracle | database | 58 | 4 | 2048 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.35 | MySQL Ora Rep 80 | oracle | database | 58 | 4 | 2048 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.36 | Percona 8.0 Rep - ProxySQL | percona | database | 58 | 8 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.46 | MariaDB105-elastx-cust | mariadb | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.47 | Redis Standalone | redis | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.48 | MSSQL2022(16.x) | microsoft | database | 58 | 4 | 4096 | 80 |
| d3t4k9t4q1ardrm415hg:192.168.73.49 | MSSQL2022(16.x) | microsoft | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.66 | MongoDB ReplicaSet 7.0 | mongodb | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.67 | MongoDB ReplicaSet 7.0 | mongodb | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.68 | MongoDB ReplicaSet 7.0 | mongodb | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.71 | MySQL Enterprise Replication 8.4 | oracle | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.72 | MySQL Enterprise Replication 8.4 | oracle | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.73 | MySQL Enterprise Replication 8.4 | oracle | database | 58 | 4 | 4096 | 30 |
| d3t4k9t4q1ardrm415hg:192.168.73.74 | mdb118-elastx-test | mariadb | database | 58 | 4 | 2048 | 30 |

< 1 2 3 > 20 / page

Download JSON Download CSV Close

## 3.25 Downloading JSON / CSV

- **Download JSON**: the raw report structure (summary + by-type-and-vendor + node details + seal metadata). Useful for downstream billing pipelines that read JSON directly.
- **Download CSV**: a ZIP with **four** files. This is the format to send for third-party verification.

The CSV ZIP contents:

| File | Purpose | Signed? |
|---|---|---|
| `report json` | The canonical JSON bytes the service signed, byte-for-byte verbatim. Do not re-format. | Yes: this is what the Ed25519 signature covers. |
| `SIGNAT json` | Manifest: `report_sha256`, `ed25519_signature`, `ed25519_public_key`, `ed25519_fingerprint`, `hmac_signature`, `hmac_key_id`, `signed_at`, `report_id`, `report_version`, `period_start`, `period_end`. | Contains the signature, not signed itself. |
| `summar csv` | Human-readable per-cluster-type + vendor aggregate. Useful for invoices. | Derived view; not cryptographically covered. |
| `node_d csv` | Human-readable per-node breakdown. | Derived view; not cryptographically covered. |

Third parties verify using `report.json` + `SIGNATURE.json`; the CSVs are for humans to read.

## 3.26 Deleting a report

From the row's actions menu, choose **Delete**. Sealed reports are immutable; deleting removes the record permanently. The delete operation is idempotent: calling it for a report ID that is already gone succeeds silently, so retries are safe. Because seals are tamper-evident, delete is the only way to "undo" a report; there is no edit.

## 3.27 Scoping reports to a customer or tenant

Set a tag on each cluster that belongs to the customer (in the ClusterControl GUI: the cluster's **Settings → Tags**). A customer code is the recommended convention, for example `customer-acme`.

Then scope a report by tag:

```
# Via the UI: Clusters or Tags picker in the Generate form.
# Via curl:
curl -sf -H "Authorization: Bearer $API_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "operation":    "generateReport",
    "period_start": "2026-04-01T00:00:00Z",
```

(continues on next page)

(continued from previous page)

```
    "period_end":   "2026-04-30T23:59:59Z",
    "tags":         ["customer-acme"]
  }' http://localhost:9520/billing/reports | jq '.report.summary'
```

The response's `summary` block is scoped to the customer's nodes only. The sealed report is stored as a separate row (a separate `report_id`) so it is independently verifiable. Filtered requests bypass the duplicate-gate that estate-wide reports use: multiple per-customer slices of the same period are expected to coexist.

## 3.28 REST API reference

All routes live under `/billing/*`. `/billing/status` is unauthenticated (health probe); everything else requires `Authorization: Bearer $API_TOKEN`.

```
BASE=http://localhost:9520
AUTH="Authorization: Bearer $API_TOKEN"
CT="Content-Type: application/json"

# Health (no auth)
curl -sf $BASE/billing/status | jq .

# --- /billing/reports (operation discriminator) -------------------------
curl -sf -H "$AUTH" -H "$CT" -d '{"operation":"listReports"}' $BASE/
↪billing/reports | jq '.reports[:3]'

# generateReport: creation-only. 409 with existing_report_id on duplicate.
curl -sf -H "$AUTH" -H "$CT" -d '{
  "operation":    "generateReport",
  "period_start": "2026-04-01T00:00:00Z",
  "period_end":   "2026-04-30T23:59:59Z"
}' $BASE/billing/reports | jq '.report_id, .report.summary'

# Overwrite (re-seal an existing period)
curl -sf -H "$AUTH" -H "$CT" -d '{
  "operation":         "generateReport",
  "period_start":      "2026-04-01T00:00:00Z",
  "period_end":        "2026-04-30T23:59:59Z",
  "force_regenerate":  true
}' $BASE/billing/reports

# Per-customer slice (filter bypasses the duplicate-gate)
curl -sf -H "$AUTH" -H "$CT" -d '{
  "operation":    "generateReport",
  "period_start": "2026-04-01T00:00:00Z",
  "period_end":   "2026-04-30T23:59:59Z",
  "tags":         ["customer-acme"]
}' $BASE/billing/reports | jq '.report_id, .report.filter'

# Use getReport to fetch a specific row by id (preserves filter descriptor)
```

(continues on next page)

(continued from previous page)

```
curl -sf -H "$AUTH" -H "$CT" -d '{"operation":"getReport","report_id":82}' \
  $BASE/billing/reports | jq '.report.summary'

# verifyReport: re-runs HMAC + Ed25519 against the local keys.
curl -sf -H "$AUTH" -H "$CT" -d '{"operation":"verifyReport","report_id":82}
↪' \
  $BASE/billing/reports | jq '{hash_valid, signature_valid, verified_at}'

# exportReport (JSON)
curl -sf -H "$AUTH" -H "$CT" -d '{
↪"operation":"exportReport","report_id":82,"format":"json"}' \
  $BASE/billing/reports -o report-82.json

# exportReport (CSV zip, the third-party verifiable bundle)
curl -sf -H "$AUTH" -H "$CT" -d '{
↪"operation":"exportReport","report_id":82,"format":"csv"}' \
  $BASE/billing/reports -o report-82.zip
unzip -l report-82.zip
# report.json + SIGNATURE.json + summary.csv + node_details.csv

# deleteReport (idempotent)
curl -sf -H "$AUTH" -H "$CT" -d '{"operation":"deleteReport","report_id":82}
↪' \
  $BASE/billing/reports | jq '{report_id, deleted}'

# --- /billing/snapshots: paginated raw event-store read ---------------
# "What's in storage, per controller, since when?" (the data view).
curl -sf -H "$AUTH" -H "$CT" -d '{"limit":5}' $BASE/billing/snapshots \
  | jq '.total, .snapshots[0]'

# Filter by controller, last 7 days
curl -sf -H "$AUTH" -H "$CT" -d '{
  "controller_id": "<controller-xid>",
  "since": "2026-05-11T00:00:00Z",
  "limit": 200
}' $BASE/billing/snapshots | jq '.total'

# --- /billing/controllers/status: live last-tick outcome ---------------
# "Is each controller being collected from right now, and why not if not?"
curl -sf -H "$AUTH" $BASE/billing/controllers/status | jq '
  .outcome,
  (.results[] | {name, controller_id, role, status, error})
'
```

## 3.29 Report seals and verification

Every sealed report carries **two signatures**, each serving a distinct trust model.

### 3.29.1 HMAC-SHA256: internal integrity

Protects `cmon-telemetry`'s own database against tampering by an attacker who reads rows but does not hold the signing key. HMAC is *symmetric* (the same key signs and verifies), so it is **never** shared with third parties.

- A **SHA-256 hash** is computed over the canonical JSON of the report body.
- An **HMAC-SHA256 signature** is computed over the hash, using the `signing_key` active at seal time.
- `signing_key_id` on the report row identifies which key signed it.
- `verifyReport` rebuilds the hash and re-runs the HMAC using `verification_keys[signing_key_id]`. A red `Seal invalid` tag in the UI means either:
  - the report's JSON body was altered after sealing (`hash_valid: false`), or
  - the current `verification_keys` map does not contain the key that signed it (`signature_valid: false`), a key-rotation operational issue, not actual tampering.

During HMAC key rotation:

```
# /etc/cmon-telemetry/config.yaml
signing_key: "<new key>"       # newly sealed reports use this
key_id:      "key-2026-Q3"
verification_keys:
  key-2026-Q3: "<new key>"
  key-2026-Q2: "<previous key>"  # keep old keys while reports signed with␣
↪them still exist
  key-2026-Q1: "<older key>"
```

Old reports remain HMAC-verifiable as long as their `signing_key_id` still appears in `verification_keys`.

### 3.29.2 Ed25519: asymmetric, third-party verifiable

Ed25519 lets anyone with the *public key* verify a report, without ever having access to the signing key. This is what MSPs use to prove to a billing counterparty that the numbers they are sending have not been modified after generation.

- On first start, `cmon-telemetry` writes a fresh Ed25519 keypair to `ed25519_key_file` (default `/var/lib/cmon-telemetry/signing.ed25519`, permissions 0600). The file is reused forever after; do not delete it.
- The public key (64 hex characters) and its fingerprint (`sha256:` + the first 16 hex characters of `SHA256(pubkey)`) appear in `GET /billing/status` and the Billing UI's **Show public key** modal.
- At seal time, an Ed25519 signature is produced over the canonical JSON bytes and stored with the report.
- Export ZIPs include `report.json` + `SIGNATURE.json`, which together are sufficient for offline verification.

### 3.29.3 What the two signatures prove together

| Scenario | HMAC | Ed25519 | Bundle trustworthy? |
|---|---|---|---|
| Report untouched | Valid | Valid | Yes |
| `report.json` edited after export | Not checked on exports (the database copy still matches) | SHA-256 mismatch | No |
| `SIGNATURE.json` signature swapped | Unchanged | Ed25519 verification fails | No |
| Whole bundle re-signed with a different keypair | Unchanged | Self-consistent, but the public key does not match the one on file | No, provided the recipient anchors to a known fingerprint with `--expected-pubkey` |
| Report regenerated from doctored inputs at the source | Valid (the service re-signed it) | Valid | Not detectable cryptographically; this is a trust-boundary issue, not a seal issue |

The last row is the important one: the seal proves a report was not altered *after* generation. It says nothing about whether the snapshots feeding it were accurate.

### 3.29.4 Verifying an exported bundle with cmon-report-verify

The recipient of a billing ZIP (for example, a billing counterparty or auditor) can confirm it has not been tampered with between generation and arrival.

**Step 1: Record the sender's public key once.** Ask the operator for their Ed25519 **fingerprint** (for example, `sha256:27dccd92548741af`) and full **public key hex**. They can copy both from the Billing UI's **Show public key** modal or from `/billing/status`. Save the public-key hex to a file, for example `~/pubkeys/acme-msp.pubkey`.

> **Warning**
>
> This is the trust anchor
>
> Everything else flows from this first handshake. If the operator rotates their key (rare, since the keypair file is persistent across restarts), ask them to re-send the new fingerprint and update your records.

**Step 2: Install the verifier.** The `cmon-report-verify` binary ships in the same `clustercontrol-telemetry` package, at `/usr/bin/cmon-report-verify`. Confirm it runs:

```
cmon-report-verify help
```

**Step 3: Sanity-check the bundle.**

```
unzip -l metering-report-105.zip
# report.json · SIGNATURE.json · summary.csv · node_details.csv
```

If those four files are not all present, the bundle is not a verifiable export. Ask the sender to re-generate via the Billing UI's **Download CSV** button, not JSON (JSON-format exports have no signature manifest).

**Step 4: Verify against the trusted public key.**

```
cmon-report-verify verify metering-report-105.zip \
  --pubkey-file ~/pubkeys/acme-msp.pubkey
```

Expected output when the bundle is trustworthy (exit code `0`):

```
[PASS] sha256:␣
↪5071c7bc7352deb8ec15cb42cc9c5be148fc32dd24b2cd4a9750855932a68c4b matches␣
↪manifest
[PASS] ed25519 signature valid
[PASS] fingerprint: sha256:27dccd92548741af matches manifest
[PASS] public key matches expected

Report ID:            105
Report Version:       3
Period:               2026-04-01 → 2026-04-20
Generated at:         2026-04-20T10:57:09Z
Total billable nodes: 23
Fingerprint:          sha256:27dccd92548741af
```

**Step 5: Interpret a failed check.** If any check fails, the CLI exits non-zero (exit code `1`) and prints which stage failed:

| Failure | What it means | What to do |
|---|---|---|
| `[FAIL] sha256: computed … manifest claims …` | `report.json` was modified after export. | Reject the bundle. Ask the sender to re-download from their Billing UI and forward the untouched ZIP. |
| `[FAIL] ed25519 signature invalid` | The signature was forged or `report.json` was modified. | Same as above: reject, ask for a fresh export. |
| `[FAIL] fingerprint … does not match manifest` | Internal inconsistency within the bundle (rare, points to a broken or malicious repack). | Reject the bundle. |
| `[FAIL] public key mismatch: bundle has X, expected Y` | The bundle was signed by a *different* installation than the one on file. | Could be a key rotation, could be a spoofed bundle. Confirm the fingerprint with the sender out-of-band before accepting anything. |

The exit code is always `2` on I/O or usage errors (missing `SIGNATURE.json`, unreadable file, and so on).

**Step 6: Archive the verified bundle.** Keep the ZIP alongside the invoice in your billing records. Future audits can re-verify it with the same CLI; the signature stays good for the lifetime of the keypair, which does not expire.

Verifier command reference:

```
# Self-consistency check only (bundle matches its own manifest)
cmon-report-verify verify metering-report-105.zip

# Trust-anchored verification (recommended, anchors to a known fingerprint)
cmon-report-verify verify metering-report-105.zip \
```

(continues on next page)

(continued from previous page)

```
  --expected-pubkey␣
↪962d9fb98a3ed1c24406bcd7a94637d2f15bb3dd802d5ad87c666e6a7077bb5f

# Same, with the pubkey in a file
cmon-report-verify verify metering-report-105.zip \
  --pubkey-file ~/pubkeys/acme-msp.pubkey

# Inspection only, no cryptographic check
cmon-report-verify show metering-report-105.zip
```

### 3.29.5 What verification proves and what it does not

| Does prove | Does not prove |
|---|---|
| The `report.json` bytes have not been altered between generation and receipt. | That the numbers in the report are factually correct; the service signs whatever snapshots it has in its database. Input-tampering inside the source environment is a separate trust issue. |
| The signature was produced by the holder of the private key matching the bundle's public key. | That the key holder is who you think they are; that is what `--expected-pubkey` enforces by anchoring to the fingerprint recorded during onboarding. |
| The bundle was sealed at the timestamp in `SIGNATURE.json`. | That the timestamp is "recent"; cross-check `period_start`/`period_end` against the billing cycle. |

## 3.30 Retention and data lifecycle

- `retention_months` (default `12`): a daily job inside `cmon-telemetry` deletes node snapshots older than the retention window. Sealed report rows are not deleted automatically.
- `/billing/status` surfaces `last_retention_cleanup` + `last_cleanup_deleted_rows` so you can confirm cleanup is running.
- **Dedup guarantee:** `(captured_at, node_id)` is unique in the snapshot store, enforced as a database constraint. Wall-clock-aligned tick boundaries keep restart-storms idempotent: a process that crashes mid-tick and restarts in the same interval bucket produces snapshots with the same `captured_at`, which the unique constraint silently drops.
- **Legacy database migration:** databases created by earlier builds predate that unique constraint. They are migrated automatically on first start after the upgrade, collapsing any duplicate rows to the lowest-id row per `(captured_at, node_id)` pair. The migration runs once, inside a transaction, and needs no operator action.
- Rough sizing: 22 eligible nodes × 1 snapshot per 60-minute tick × 12 months ≈ 230k rows, well under 50 MB of SQLite storage in practice. Tighter intervals scale linearly.

## 3.31 Frequently asked questions

**Q: My first report shows `total_billable_nodes: 0` even though I see snapshots in `/billing/status`.**
A: Nodes must accumulate `min_active_hours` (default 24 hours) of active-or-stopped state to count as billable. Wait a day after the first collection tick, or lower `min_active_hours` temporarily for a test. The

active-hours formula is gap-based: it reads timestamps directly, so changing the collection interval mid-run does not retroactively inflate or deflate counts.

**Q: Some hosts have `vcpu: 0` in the first report.**
A: vCPU is resolved in two stages: the host's stat collector first, then a fallback via `cmon-proxy` for hosts that have not been sampled yet. Hosts that are new and have not had a single stat tick will stay at 0 until then.

**Q: I scaled out and added a second cmon-proxy. Will I double-count?**
A: Not if `cmon-telemetry` is configured with a single `billing.cmon_proxy.url`. There is one poller; controller discovery goes through that one `cmon-proxy`, and the `(captured_at, node_id)` constraint dedups regardless. If you front multiple `cmon-proxy` instances with a load balancer, each tick lands on whichever proxy the balancer picked; it is still one logical poller. Multiple `cmon-telemetry` processes against the same database is not supported.

**Q: I selected a cluster in the filter bar and the table is empty.**
A: Filter matching is strict: only rows with a matching `filter` descriptor pass. Estate-wide reports are excluded when a selection is active. Clear the filter bar to see them again.

**Q: The Billing tab is missing from the navigation.**
A: `billing_enabled: false` on the `cmon-proxy` serving the UI. The `/billing` menu entry and route are both gated on that flag; a direct link to `/billing` when disabled redirects to the overview page. Flip `billing_enabled: true` in `ccmgr.yaml` and restart `cmon-proxy`.

**Q: I tried to generate a report for the same period twice and got `409 Conflict`.**
A: Expected; report generation is creation-only. The response body includes the existing `report_id` so the UI can offer to open it. To re-seal, pass `force_regenerate: true`.

**Q: `/billing/controllers/status` shows `status: "empty"` for some controllers but `"ok"` on others.**
A: `empty` is not an error: it means the fetch succeeded and the controller owned zero clusters in this tick. This is common in multi-controller pools where cluster ownership migrates between members.

**Q: `/billing/controllers/status` shows "User cmon-telemetry is suspended for previous authentication failures" on a pool member.**
A: Pool-member key propagation gap (see the multi-controller caveat under provisioning). The controller replicates the user record across pool members, but per-user public keys do not propagate, so a fresh bootstrap leaves the other members with stale keys, authentication fails repeatedly, and the brute-force lockout suspends the user. Re-run `ccmgradm serviceauth-bootstrap` and `s9s user --unsuspend cmon-telemetry` on each suspended member.

**Q: A report's seal is shown as invalid in the UI.**
A: Verify the current `verification_keys` map covers the report's `signing_key_id`. If it does and the seal still fails, the stored JSON was modified outside of `cmon-telemetry`. Consider re-generating the report and investigating the database for tampering.

**Q: How do I rotate the HMAC signing key without invalidating old reports?**
A: Add the new key to `signing_key` and `verification_keys`, update `key_id`, but keep the old key in `verification_keys` until the last report signed with it ages out.

**Q: Should I share my `signing_key` with a billing counterparty?**
A: **No, never.** The HMAC signing key is symmetric: anyone holding it can sign fake reports. Share only the Ed25519 **fingerprint /** public key. The whole point of Ed25519 is that you prove authenticity without disclosing the key.

**Q: What happens if I delete `ed25519_key_file`?**
A: On the next start, `cmon-telemetry` generates a fresh keypair with a new fingerprint. Existing reports

remain signed by the *old* key, so verifiers anchored to the old fingerprint will reject new exports until you also update them with the new fingerprint. Do not delete the file in production without a plan.

**Q: My exported ZIP does not have `SIGNATURE.json` or `report.json`.**
A: You are on a pre-2.5.0 `cmon-telemetry`, or you downloaded the JSON export instead of the CSV ZIP. Only the CSV ZIP carries the third-party-verifiable bundle.

## 3.32 Migrating ClusterControl to Another Server

This guide details the migration process for your ClusterControl instance from one server to another.

### 3.32.1 Example use cases

Here are real-world reasons to move ClusterControl to a new server:

- Operating system, hardware refresh or cloud move - Aging host machines, operating systems reaching end-of-life (e.g., Ubuntu 16.04), or moving to a new cloud provider, region, or VPC. Migrating ClusterControl ensures it runs on a supported, faster, or standardized platform.
- Capacity and performance scaling - If the current ClusterControl server is insufficient for the number of managed clusters, nodes, or metrics, a move to a larger instance with more CPU, RAM, or SSD can significantly improve the speed of backups, reporting, and dashboards.
- Security and compliance re-segmentation - You need the management plane in a hardened subnet, behind a new firewall model, or separating it from production networks to meet compliance standards (e.g., PCI/GDPR/SOX). Migrating ClusterControl lets you change IP addresses, rotate keys, and enforce new controls.

### 3.32.2 Prerequisites

Make sure the following conditions are met:

- You must be running the latest ClusterControl available at the present time, with Prometheus agent-based monitoring enabled. Upgrade the existing ClusterControl host to the latest version available. See *Upgrading ClusterControl*.
- The new ClusterControl server must have equal or bigger capacity in terms of resources like CPU, RAM or disk space. Otherwise, there is a risk of CMON database can't be started up due to InnoDB buffer pool initialization failure.
- It is highly recommended to use the same operating system distribution family as the old ClusterControl. For example, if the ClusterControl host is running on CentOS 7, use Rocky Linux 9 as the operating system on the new host.
- Prepare a list of database nodes that require changing of the grant and privileges after ClusterControl IP address has changed. You can get all the nodes under ***ClusterControl → Nodes*** section.

### 3.32.3 Migration steps

The following recommended steps will install the latest version of ClusterControl and migrate your ClusterControl settings and historical data on the new server. The steps include:

1. Install ClusterControl and Prometheus on the new server.
2. Export the `cmon` database, ClusterControl and Prometheus configuration and data files.
3. Replace any occurrences of the old IP address to the new IP address.

4. Restore the `cmon` database.
5. Update the `cmon` user with the new IP address on all the database nodes.
6. Migrate the SSH key.
7. Start services.

**Info**

Your database clusters are expected to remain unaffected; however, the migration steps will require downtime for ClusterControl.

### Step 1: Install ClusterControl

Install ClusterControl on the new server using the installer script. This will automatically install and configure all ClusterControl's packages and dependencies on the new server.

On the ClusterControl server, run the following commands:

```
wget https://severalnines.com/downloads/cmon/install-cc
chmod +x install-cc
sudo ./install-cc     # omit sudo if you run as root
```

If the new server is running in an air-gapped environment, you will need to install ClusterControl using offline installation. See more details here Offline Installation.

???+ Note

```
In the installation wizard, ensure that the MySQL root and cmon passwords␣
↪match those of the old server. Verify inside `/etc/cmon.cnf`.
```

Once the installation is done, stop ClusterControl services on the old server and the new server to get consistent data, also stop Prometheus on the old server:

```
# both new and old servers
systemctl stop cmon cmon-cloud cmon-events cmon-ssh
systemctl disable cmon
# old server
systemctl stop prometheus
```

### Step 2: Install Prometheus on the new server

ClusterControl automatically installs Prometheus to store time-series monitoring data when the first cluster is added. Since a new ClusterControl host doesn't have an existing cluster, Prometheus won't be installed or running. Therefore, a manual Prometheus installation is required before importing monitoring data from the old server. This step can be skipped if you prefer to start with fresh monitoring data after the ClusterControl migration.

On the new server, download the latest version of Prometheus at https://prometheus.io/download/, install Prometheus binaries and create the operating system user:

```
# new server
# note: the version might change in the future
```

(continues on next page)

(continued from previous page)

```
wget␣
↪https://github.com/prometheus/prometheus/releases/download/v3.7.2/prometheus-3.7.2.linu
tar -xzf prometheus-*.tar.gz
cd prometheus-3.7.2.linux-amd64
cp prometheus /usr/local/bin/
cp promtool /usr/local/bin/
useradd --no-create-home --shell /bin/false prometheus
```

### Step 3: Export ClusterControl database

On the old server, take a MySQL dump of the `cmon` database from the server using `--replace` and `--no-create-info` flags:

```
# old server
mysqldump -uroot -p --databases cmon --replace --no-create-info > /
↪root/cmondb_backup.sql
```

Once the backup is done, copy the MySQL dump file to the new ClusterControl server:

```
# old server
scp /root/cmondb_backup.sql root@{new_server}:/root/
```

### Step 4: Copy ClusterControl and Prometheus data files

On the old server, copy the ClusterControl controller configuration files, data file, Prometheus configuration and data files to the new server. It is recommended to back up the existing ClusterControl data files and configuration files on the new server beforehand:

```
# backup CMON data on new server
mkdir -p /root/cc-backup
cp /etc/cmon.cnf /root/cc-backup/
cp -pfR /etc/cmon.d /root/cc-backup/
cp -pfR /var/lib/cmon /root/cc-backup/
cp /etc/s9s.conf /root/cc-backup/

# old server
# cmon.cnf - ClusterControl main config
scp /etc/cmon.cnf root@{new_server}:/etc/
# /etc/cmon.d/cmon*.cnf - Cluster config
scp -r /etc/cmon.d/* root@{new_server}:/etc/cmon.d/
# /var/lib/cmon - ClusterControl data files
scp -r /var/lib/cmon/* root@{new_server}:/var/lib/cmon/
# /etc/s9s.conf - ClusterControl CLI config
scp /etc/s9s.conf root@{new_server}:/etc/
# depending on the local user you configure to access CLI
scp ~/.s9s {user}@{new_server}:~/
# Prometheus-related data
scp -r /etc/prometheus root@{new_server}:/etc/
scp /etc/systemd/system/prometheus.service root@{new_server}
```

(continues on next page)

(continued from previous page)

```
↪:/etc/systemd/system/
scp -r /var/lib/prometheus root@{new_server}:/var/lib/
```

### Step 5: Replace any occurrences of the old IP address

On the new server, replace all occurrences of the old IP address in the ClusterControl controller configuration files, database dump file and Prometheus configuration file. Run these commands:

```
# new server
sed -i 's|old_ip|new_ip|g' /etc/cmon.cnf
sed -i 's|old_ip|new_ip|g' /etc/cmon.d/cmon*.cnf
sed -i 's|old_ip|new_ip|g' /root/cmondb_backup.sql
sed -i 's|old_ip|new_ip|g' /etc/prometheus/prometheus.yml
```

**Example**

```
sed -i 's|192.168.99.11|192.168.99.201|g' /etc/cmon.cnf
sed -i 's|192.168.99.11|192.168.99.201|g' /etc/cmon.d/cmon*.cnf
sed -i 's|192.168.99.11|192.168.99.201|g' /root/cmondb_backup.sql
sed -i 's|192.168.99.11|192.168.99.201|g' /etc/prometheus/prometheus.yml
```

### Step 6: Restore ClusterControl database

On the new server, restore the database backup onto the database server:

```
mysql -uroot -p < /root/cmondb_backup.sql
```

### Step 7: Update the Clustercontrol user on all the database nodes

To allow the new ClusterControl server to manage the existing database nodes, update the cmon user with the new IP address on all database nodes managed by the old ClusterControl host:

#### MySQL/MariaDB/Percona

For MySQL, only execute the following command on:

- Primary node for primary-replica replication and group replication cluster.
- One of the database nodes for a Galera cluster.

```
-- execute on the primary database node only
RENAME USER 'cmon'@'{old_ip}' TO 'cmon'@'{new_ip}';
```

**Example**

```
SELECT user,host FROM mysql.user WHERE user = 'cmon'; -- before
RENAME USER 'cmon'@'192.168.99.11' TO 'cmon'@'192.168.99.201';
SELECT user,host FROM mysql.user WHERE user = 'cmon'; -- after
```

### PostgreSQL/TimescaleDB

Example for PostgreSQL 17, where the `pg_hba.conf` is located at `/etc/postgresql/17/main/pg_hba.conf`:

```
# update the IP address
sed -i 's|old_ip|new_ip|g' /etc/postgresql/17/main/pg_hba.conf
# reload the pg_hba
psql> SELECT pg_reload_conf();
```

**Example**

```
$ sed -i 's|192.168.99.11|192.168.99.201|g' /
↪etc/postgresql/17/main/pg_hba.conf
$ su - postgres
$ psql
psql> SELECT pg_reload_conf();
```

### Step 8: Migrate SSH key

Copy the SSH private key used by the old server to the new server. If you use root user, it is commonly located at `/root/.ssh/id_rsa`. For sudo user, it is usually located at `/home/{user}/.ssh/id_rsa`.

```
# old server
scp ~/.ssh/id_rsa {user}@{new_server}:~/.ssh/id_rsa
```

**Example**

```
# if you use root user to manage the cluster
$ scp ~/.ssh/id_rsa root@192.168.99.201:~/.ssh/
# if you use sudo user to manage the cluster
$ scp ~/.ssh/id_rsa myadmin@192.168.99.201:~/.ssh/
```

If you choose not to use the same SSH key, you may generate a new key but use the same key path as the old server. For example, generate a new SSH private key and swap it with the existing key file located at `/root/.ssh/id_rsa`. Then set up SSH key-based authentication from ClusterControl server to all database nodes as shown in *Requirements → SSH key-based authentication*.

### Step 9: Start services

Once you have done all the steps, set the correct ownership of the Prometheus directories, start and enable Prometheus and ClusterControl services:

```
chown -Rf prometheus:prometheus /etc/prometheus
chown -Rf prometheus:prometheus /var/lib/prometheus
systemctl enable prometheus
systemctl enable cmon
systemctl start prometheus
systemctl start cmon cmon-cloud cmon-events cmon-ssh
```

Then open a web browser and go to `https://<new_ClusterControl_host>/` and login using your exisiting admin username and password on the welcome page.

**Warning**

Attention

If you store your backups on the ClusterControl node, you have to manually transfer them to the exact location on the new ClusterControl server.

### 3.32.4 Post-migration checklist

Make sure after the migration, the following aspects are working as expected:

| Chec | Aspect | Verification action |
|---|---|---|
| ☐ | Database cluster state. | Go to ***ClusterControl GUI → Home*** and make sure all clusters in the correct state. |
| ☐ | If ClusterControl host is the backup destination, backup files are transferred to the new server. | Go to ***ClusterControl GUI → Backups*** and make sure the database backups are correctly listed. |
| ☐ | ClusterControl CLI is able to communicate with the ClusterControl controller. | ClusterControl CLI: `s9s cluster --list --long` and make sure you see a list of cluster, similar to the GUI. |
| ☐ | Monitoring dashboards are working. | Go to ***ClusterControl GUI → Clusters → choose a cluster → Dashboards*** and make sure the recent monitoring data are populated. Otherwise, fix it with ***ClusterControl GUI → Clusters → choose a cluster → Dashboards → More → Re-enable agent based monitoring***. |

## 3.33 Uninstall or Reinstall ClusterControl

### 3.33.1 Reinstall

To easiest way to re-install ClusterControl is:

#### Red Hat/CentOS/Rocky Linux/AlmaLinux

If you are running on MySQL, substitute “mariadb” with “mysql”:

```
$ systemctl stop mariadb cmon cmon-ssh cmon-events cmon-cloud cmon-proxy
$ dnf remove clustercontrol* s9s-tools mariadb* # or mysql*
$ rm -Rf /var/lib/mysql
$ rm -Rf /var/lib/cmon
$ rm -Rf /etc/s9s.conf
$ rm -Rf /root/.s9s
$ rm -Rf /etc/cmon*
```

#### Debian/Ubuntu

If you are running on MySQL, substitute "mariadb" with "mysql":

```
$ systemctl stop mariadb cmon cmon-ssh cmon-events cmon-cloud cmon-proxy
$ apt purge clustercontrol* s9s-tools mariadb*   # or mysql*
$ rm -Rf /var/lib/mysql
$ rm -Rf /var/lib/cmon
$ rm -Rf /etc/s9s.conf
$ rm -Rf /root/.s9s
$ rm -Rf /etc/cmon*
```

Then, perform the installation again using the installer script:

```
$ ./install-cc
```

### 3.33.2 Uninstall

If ClusterControl is installed on a dedicated host (i.e., not co-located with your application), uninstalling ClusterControl is pretty straightforward. It is enough to bring down the ClusterControl node and revoke the ClusterControl-related user privileges from the managed database nodes:

#### MySQL/MariaDB

```
--- On the managed database nodes
mysql> DELETE FROM mysql.user WHERE user LIKE 'cmon%';
mysql> FLUSH PRIVILEGES;
```

#### PostgreSQL/TimescaleDB

```
--- On the managed database nodes
postgres=# DROP USER 'cmon*';
```

Then, remove all lines related to ClusterControl database users (`cmon*`) inside `pg_hba.conf`.

#### MongoDB

```
// Iterate over all users in the current database
db.getUsers().forEach(function(user) {
    // If the username starts with "cmon"
    if (user.user.match(/^cmon/)) {
        print("Dropping user: " + user.user);
        // Drop the user
        db.dropUser(user.user);
    }
});
```

### Redis/Valkey

```
# On the managed database nodes
for user in $(redis-cli ACL LIST | grep '^user cmon' | cut -d' ' -f2)
do
    echo "Deleting user: $user"
    redis-cli ACL DELUSER "$user"
done
```

Before removing the package, stop all cmon related services:

### On systemd

```
$ systemctl stop cmon cmon-ssh cmon-events cmon-cloud cmon-proxy
```

### On SysVinit

```
$ service cmon stop
$ service cmon-ssh stop
$ service cmon-events stop
$ service cmon-cloud stop
```

If ClusterControl is installed via the Severalnines repository, use the following command to uninstall via the respective package manager:

### Red Hat/CentOS/Rocky Linux/AlmaLinux

```
$ yum remove -y clustercontrol* s9s-tools
```

### Debian/Ubuntu

```
$ sudo apt-get remove -y clustercontrol* s9s-tools
```

Remove ClusterControl-related databases and privileges:

```
mysql> DROP SCHEMA cmon;
mysql> DROP SCHEMA dcps;
mysql> DELETE FROM mysql.user WHERE user LIKE 'cmon%';
mysql> FLUSH PRIVILEGES;
```

CHAPTER

FOUR

# REFERENCE MANUALS

This reference manual provides detailed information on every feature, setting, and command available within ClusterControl, ensuring you have the knowledge needed to maximize the platform's capabilities.

## 4.1 What You'll Find in This Manual

This manual serves as an essential resource for Database Administrators (DBAs), DevOps Engineers, IT Managers, and Developers looking to optimize their database clusters and streamline their operational workflows. Here's an overview of what each section covers:

### 4.1.1 ClusterControl Components

A detailed documentation on all ClusterControl components on configuring ClusterControl, including requirements, supported environments, and integration options. From initial setup to configuring essential components, this section helps you build a strong foundation.

### 4.1.2 Feature Reference

An in-depth look at ClusterControl's features, from cluster deployment and scaling to monitoring, backup, and failover. Each feature is broken down to show how it works, its benefits, and example use cases to help you understand when and why to use each function.

### 4.1.3 Command and API Reference

Detailed documentation on every available command and API endpoint within ClusterControl. This is especially useful for users looking to integrate ClusterControl with other tools or automate tasks, as well as those utilizing the CLI for greater control over their environment.

### 4.1.4 Troubleshooting and Support

Practical advice and solutions for common issues, error codes, and known limitations. This section also provides guidance on accessing support resources and engaging with the ClusterControl community for additional help.

## 4.2 Components

This section introduces the core components that make up ClusterControl. Learn how each component works together to provide deployment, management, monitoring, and automation capabilities for your database clusters.

**See also**

- *ClusterControl Controller →*
- *ClusterControl GUI →*
- *ClusterControl CLI →*
- *ClusterControl Web SSH →*
- *ClusterControl Notifications →*
- *ClusterControl Cloud →*
- *ClusterControl Cloud File Manager →*
- *ClusterControl Proxy →*
- *ClusterControl Agent →*

## 4.3 ClusterControl Controller

ClusterControl Controller (cmon) is the core backend process that performs all automation and management procedures. It is installed as `/usr/sbin/cmon` and comes with a collection of helper scripts in `/usr/bin` the directory (prefixed with `s9s_`) to complete specific tasks. However, some of the scripts have been deprecated due to the corresponding tasks are now being handled by the CMON core process.

ClusterControl Controller packages are available at Severalnines Repository page. The packages are also available at the Severalnines download site. RedHat-based systems should download and install the RPM package while Debian-based systems should download and install the DEB package. The package name is formatted as:

- RPM package (RedHat-based systems): `clustercontrol-controller-{version}-{build number}-{architecture}.rpm`
- DEB package (Debian-based systems): `clustercontrol-controller-{version}-{build number}-{architecture}.deb`

A configuration file `/etc/cmon.cnf` is required to initially run a `cmon` process. It is possible to have several configuration files for multiple clusters as described in the *Configuration Files*.

### 4.3.1 Command Line Arguments

By default, if you just run `cmon` (without any arguments), `cmon` defaults to run in the background. The binary supports several command-line options as shown below:

| Shorthand, Option | Description |
|---|---|
| `-h, --help` | Print help. |
| `--help-config` | Print the manual for configuration parameters. See *Configuration Options*. |
| `--help-init` | Shows the special options for `--init`. |
| `-v, --version` | Prints out the version number and build info. |
| `--debug` | Print more messages into the log. |
| `--logfile=[filepath]` | The path of the log file to be used. |

continues on next page

Table 4.1 – continued from previous page

| Shorthand, Option | Description |
|---|---|
| `--log-format=FORMAT` | Format string for log files. |
| `--log-format-stderr=FORMAT` | Format string for printing log messages. |
| `--no-color` | Do not use colors when printing log to stderr. |
| `--no-log-db` | Do not store cmon logs in database. |
| `--upload-js-scripts` | To upload JS scripts from archive files to CMON DB. |
| `--disable-dsl-advisors` | Disables jobs for all scheduled DSL Advisors. |
| `--mysql-connect-retries=N` | The connect retries to cmonDB before quitting. |
| `-s, --syslog` | Log also to syslog. |
| `-b, --bind-addr='ip1,ip2..'` | Bind Remote Procedure Call (RPC) to IP addresses (default is 127.0.0.1,::1). By default `cmon` binds to 127.0.0.1 and ::1. If another bind-address is needed, then it is possible to define the bind addresses in the file `/etc/default/cmon`. See *Startup File*. |
| `-c, --cloud-service=URL` | A custom clustercontrol-cloud service URL. |
| `-d, --nodaemon` | Run in the foreground. Ctrl + C to exit. |
| `-e, --events-client=URL` | Additional RPC URL where backend sends events. |
| `-g, --grant` | Create grants. |
| `-i, --init` | Creates configuration file and database. |
| `--log-rpc` | Log every RPC call (very verbose). |
| `--no-safety-checks` | Do not check if another cmon is connected. |
| `-p, --rpc-port=[integer]` | Listen on RPC port. The default is 9500. |
| `-r, --directory=[directory]` | Running directory. |
| `-u, --upgrade-schema` | Try to upgrade the CMON schema (Supported from CMON version 1.2.12 and later). |
| `-n, --pool` | Specify if the controller runs in pool. |
| `–secret-storage-engine=<file | k8s |
| `--k8s` | Run cmon in Kubernetes pod (deployment) mode. Implies `--secret-storage-engine=k8s` unless an engine is given explicitly. |
| `--vault` | Deprecated alias of `--secret-storage-engine=vault`. |
| `--migrate-secrets` | Migrate config files and certificates to Kubernetes secrets, then exit. Requires `--k8s`. |
| `--k8s-secret-label-filter=<selector>` | Kubernetes labelSelector used when listing k8s secrets so `cmon` fetches only its own (labelled) secrets. Default is `com.severalnines.cmon=true`; pass an empty value to disable filtering (list all). |

### 4.3.2 Startup File

To customize the `cmon` startup process, you can define the command line arguments in a custom file, instead of hacking up the init script directly. The `cmon` init script (or systemd) will append all configuration options defined inside `/etc/default/cmon` when starting up the cmon process and translates those options into command line arguments. For example:

```
$ cat /etc/default/cmon
RPC_PORT=9500
RPC_BIND_ADDRESSES="10.10.10.13,192.168.33.1,127.0.0.1"
EVENTS_CLIENT=http://127.0.0.1:9510
CLOUD_SERVICE=http://127.0.0.1:9518
```

In the example above, `cmon` will bind into those IP addresses and listen to port 9500 once started. If you filter out the `ps` output from the server, you should get the following:

```
/usr/sbin/cmon --rpc-port=9500 \
    --bind-addr='10.10.10.13,192.168.33.1,127.0.0.1' \
    --events-client='http://127.0.0.1:9510' \
    --cloud-service='http://127.0.0.1:9518
```

### 4.3.3 Configuration Options

Values that consist of special characters must be enclosed with a single-quote. Any value change to the configuration options that shall be applied to the `cmon` configuration file or indirectly via GUI or through CLI command, requires a service restart before they are applied.

Following is the list of configuration options inside the CMON configuration file. You can also see them by using `--help-config` the parameter in the terminal:

```
$ cmon --help-config
```

### General Configurations

| Option | Description |
|---|---|
| clus-ter_id=<in | Cluster identifier. This will be used by CMON to indicate which cluster to provision. It must be unique, two clusters can not share the same ID. Example: `cluster_id=1` |
| name=<stri | Cluster name. The cluster name configured under ***ClusterControl → Clusters → choose a clutser → Actions → Edit details → Name*** precedes this. Example: `name='Galera Cluster'` Another alias: `cluster_name` |
| type=<stri | Cluster type. Supported values are "galera", "mysql_single", "mysqlcluster", "mongodb", "postgresql_single", "replication", "group_replication", "elasticsearch", "sqlserver", "redis-cluster". Example: `type=galera` Another alias: `cluster_type` |
| tags=<stri | A set of strings the user can specify to tag a cluster. - Example: `tag="production;galera;MySQL;app1"` |
| cre-ated_by_jo | The ID of the job created this cluster. This is usually automatically generated by ClusterControl. Example: `created_by_job=13` |
| ven-dor=<strin | Database vendor name. ClusterControl needs to know this in order to distinguish the vendor's relevant naming convention especially for package name, daemon name, deployment steps, recovery procedures, and lots more. Supported value at the moment is "oracle", "percona", "codership", "mariadb", "10gen". Example: `vendor=percona` |
| use_intern integer> | Setting which disabled the 3rd party repository to be set up. Default is 0 (false). |
| en-able_html_ integer> | Enables sending of HTML e-mails. Default is 1 (true). |
| fron-tend_url=< | The ClusterControl URL to be embedded inside e-mail notifications. The URL sent in the emails to direct the recipient to the ClusterControl UI. Example: `fron-tend_url='https://monitor.domain.com/clustercontrol'` |
| er-ror_report | The storage location of error reports on the controller node. |
| ops_report | How many days to keep operational reports. Operational Reports matching retention period are removed. Default is 31. |
| ccx_mode=< | If this cluster is configured from CCX. |
| ccxad-min_passwo | The password used for the ccxadmin user. |
| cloud_only | When enabled, backups configured to be stored in cloud storage (e.g. S3) will be directly uploaded to cloud storage without intermediate node storage. This only applies to streaming backups such as `mariabackup`/`xtrabackup`/`pg_basebackup`. Default: false. |
| database_g | Setting for database growth monitoring. Default: true. |
| database_g | Specifies how many of the largest databases will be included in the database growth analysis. Default: 20. |
| db_growth_ | Determines how long the controller should retain data related to database growth monitoring. Default: 32. |
| email_subj | Custom e-mail subject prefix. |
| force_http | Forces HTTP repository URLs to be replaced with HTTPS. Default: false. |
| enter-prise_toke | Customer's Repo/Download Token for Enterprise versions of Databases. |
| snmp_commu | The SNMP community string to be used on sent SNMP traps. |
| snmp_proto | The SNMP protocol to be used. |
| snmp_trap_ | The host to which SNMP traps will be sent when an alarm occurs. |
| snmp_trap_ | The target port for SNMP traps when an alarm occurs. |
| use_preins | Use pre-installed SSL certs instead of generating new ones. Certificates must exist under `/etc/cmon.d/ccx/` named (`ca.pem`, `server.pem`, `server_key.pem`). Default: false |
| paral-lel_node_s | Maximum number of nodes deployed concurrently during cluster creation. Default: 4, minimum: 1. Parallel deployment is currently supported for Galera Cluster, MySQL Replication, and PostgreSQL (streaming replication). Other cluster types are deployed |

### CMON

| Option | Description |
|---|---|
| hostname=<string> | Hostname or IP address of the controller host. Example: hostname=192.168.0.10 |
| controller_id=<integer> | An arbitrary identifier string of this controller instance. Example: controller_id=1 |
| mode=<string> | CMON role. Supported values are "controller", "dual", "hostonly". Example: mode=controller |
| agentless=<boolean integer> | CMON controller mode (deprecated). Agents are no longer supported. 0 for agent-based or 1 for agentless (default). Example: agentless=1 |
| logfile=<path> | CMON log file location. This is where CMON logs its activity. The file will be automatically generated if it doesn't exist. CMON will write to Syslog by default. Example: logfile=/var/log/cmon.log |
| pidfile=<path> | CMON process identifier file directory. Keep the default value is recommended. Example: pidfile=/var/run |
| mysql_hostname=<string> | The MySQL hostname or IP address where the CMON database resides. Using an IP address is recommended. Default is 127.0.0.1. Example: mysql_hostname=192.168.0.5 Other aliases: cmon_mysql_hostname, cmondb_hostname, local_mysql_hostname, cmon_local_mysql_hostname. |
| mysql_password=<string> | The MySQL password for user cmon to connect to CMON database. Example: mysql_password='cM%^nP4ss' Other aliases: cmon_mysql_password, cmondb_password |
| mysql_port=<integer> | The MySQL port used by CMON to connect to the CMON database. Example: mysql_port=3306 Other aliases: cmon_mysql_port, cmondb_port |
| mysql_socket=<boolean integer> | The local UNIX socket file to be used for MySQL connection (if set, TCP will not be used). Example: mysql_socket=/var/lib/mysql/mysqld.sock |
| mysql_basedir=<path> | The MySQL base directory used by CMON to find MySQL client-related binaries. Example: mysql_basedir=/usr Another alias: basedir |
| mysql_bindir=<path> | The MySQL binary directory used by CMON to find MySQL client-related binaries. Another alias: bindir Example: mysql_bindir=/usr/bin |
| mysql_scriptdir=<path> | The scripts dir of MySQL installation. |
| config_file_path=<path> | The config file path (read-only) for CMON instance. |

continues on next page

Table 4.2 – continued from previous page

| Option | Description |
|---|---|
| cmon_db=<string> | CMON database name. Default to "cmon". Example: cmon_db=cmon Another alias: cmondb_database |
| cmon_user=<string> | The username for the CMON database. Default to "cmon". Example: cmon_user=cmon Another alias: cmondb_user |
| cmondb_ssl_key=<path> | Path to SSL key, for SSL encryption between the CMON process and the CMON database. Example: cmondb_ssl_key=/etc/ssl/mysql/client-key.pem |
| cmondb_ssl_cert=<path> | Path to SSL certificate, for SSL encryption between the CMON process and the CMON database. Example: cmondb_ssl_cert=/etc/ssl/mysql/client-cert.pem |
| cmondb_ssl_ca=<path> | Path to SSL CA, for SSL encryption between the CMON process and the CMON database. Example: cmondb_ssl_ca=/etc/ssl/mysql/ca-cert.pem |
| cluster_ssl_key=<path> | Path to SSL key, for SSL encryption between the CMON process and managed MySQL Servers. Example: cluster_ssl_key=/etc/ssl/mysql/client-key.pem |
| cluster_ssl_cert=<path> | Path to SSL certificate, for SSL encryption between the CMON process and managed MySQL Servers. Example: cluster_ssl_cert=/etc/ssl/mysql/client-cert.pem |
| cluster_ssl_ca=<path> | Path to SSL CA, for SSL encryption between CMON and managed MySQL Servers. Example: cluster_ssl_ca=/etc/ssl/mysql/ca-cert.pem |
| cluster_certs_store=<path> | Directory path to store SSL-related files. This is required when you want to add a new node in an encrypted Galera cluster. Example: cluster_certs_store=/etc/ssl/galera/cluster_1 |
| cluster_ssl_enforce=<boolean integer> | If set to 1 (true), then SSL is required. Default is 0 (false). CMON is going to raise a critical alarm if an SSL connection can not be made to the node. Example: cluster_ssl_enforce=1 |

continues on next page

Table 4.2 – continued from previous page

| Option | Description |
|---|---|
| rpc_key=<string> | Unique secret token for authentication. To interact with individual clusters via CMON RPC interface (port 9500), one must use this key, or else you would get 'HTTP/1.1 403 Access denied'. ClusterControl UI needs this key stored as RPC API Token to communicate with the CMON RPC interface. Each cluster should be configured with a differentrpc_key value. This value is automatically generated when a new cluster/server is created or added into ClusterControl. Example: rpc_key=VJZKhr5CvEGI32dP Another alias: rpc_api_key |
| rpc_version=<string> | The RPC version created the cluster. |
| net_read_timeout=<integer> | Network read timeout value in seconds (DB connections). Default is 10. |
| net_write_timeout=<integer> | Network write timeout value in seconds (DB connections). Default is 10. |
| connect_timeout=<integer> | Network connect timeout value in seconds (DB connections). Default is 10. |
| owner=<integer> | The CMON user ID of the owner. |
| group_owner=<integer> | The CMON group ID of the group owner. |
| cdt_path=<string> | The location in the CMON Directory Tree (CDT). See *s9s-tree*. |
| acl=<string> | The Access Control List is a string controlling the access to the cluster object. See *ACL Text Forms*. |
| staging_dir=<path> | Staging path for temporary files. Example: staging_dir=/home/ubuntu/s9s_tmp Another alias: stagingdir |
| plugin_dir=<path> | Directory path for CMON plugins. Default to /var/cmon/plugins Another alias: plugin_path |
| cmon_use_mail=<boolean integer> | Setting to use the mail command for sending out emails. Default is 0 (false). Example: cmon_use_mail=1 |
| cmon_mail_sender=<email> | The sender email address when sending out emails. Example: cmon_mail_sender=info@example.com |
| cmon_pool_size=<integer> | The connection pool size to the cmon database. Default is 64. Example: cmon_pool_size=32 |
| cmon_ha_heartbeat_interval_millis=<in | CMON HA heartbeat interval measured in milliseconds. Default is 1000. |
| cmon_ha_heartbeat_millis_rand_max=<in | CMON HA timeout random component max value. Default is 2500. |
| cmon_ha_heartbeat_network_timeout=<in | CMON HA heartbeat network connection timeout measured in seconds. Default is 5. |
| cmon_ha_heartbeat_timeout_millis=<int | CMON HA timeout for heartbeats in milliseconds. Default is 5000. |

continues on next page

Table 4.2 – continued from previous page

| Option | Description |
|---|---|
| cmon_ha_max_send_retry=<integer> | CMON HA log send maximum retry count. Default is 15. |
| cmon_ha_send_retry_delay=<integer> | CMON HA log send retry delay in seconds. Default is 3. |
| vault_addr=<URI> | URI address of the Vault server. See Hashicorp Vault Integration. Example: vault_addr=https://localhost:8200 |
| vault_auto_migrate=<boolean integer> | Enable this to auto-move all the secrets to the configured Vault instance. Default is 0 (false). See Hashicorp Vault Integration. Example: vault_auto_migrate=1 |
| vault_path=<path> | The secrets path prefix for ClusterControl credentials. Default value is clustercontrol. The controller is going to store the data within this path, e.g clustercontrol/controller_UUID/cluster_ID/mysql_password. Each ClusterControl instance and controller, and cluster can have its unique namespace. See Hashicorp Vault Integration. Example: vault_path="staging/clustercontrol" |
| vault_token=<string> | A token used by the controller to authorize itself against the Vault server. The Vault token is created by the Vault server. When the vault_token is present credentials for new clusters will be stored in Vault and not in the cmon configuration files. See Hashicorp Vault Integration. |
| audit_ignore_users=<string> | Skip certain users from audit logging. Comma-separated list. Default: system;ccrpc. |
| audit_logfile=<path> | Path to CMON's audit log file. Default: /var/log/cmon_audit.log. |
| connect_heartbeat_when_faulty=<intege | MySQL connections blocking period (in seconds) when a faulty situation (errNo=3159) is detected to prevent resource overuse. Default: 30. |
| mail_queue_disabled=<boolean> | Disable CMON mail feature. Default: false. |
| prometheus_hostname=<string> | A Prometheus node hostname for which CMON shall use custom SSH credentials. |
| prometheus_ssh_user=<string> | SSH username used for accessing the Prometheus node. |
| prometheus_ssh_port=<integer> | Custom SSH port for Prometheus node. Default: 22. |
| prometheus_ssh_password=<string> | SSH password for connecting to Prometheus node. |
| prometheus_ssh_keypath=<path> | SSH key file for connecting to Prometheus node. |
| prometheus_ssh_keydata=<string> | Base64-encoded SSH key for Prometheus connection. |

### Operating System

| Option | Description |
|---|---|
| os=<strin | The operating system distribution runs across the cluster, including ClusterControl host. 'redhat' for RedHat-based distributions (CentOS/Red Hat Enterprise Linux/Oracle Linux) or 'debian' for Debian-based distributions (Debian/Ubuntu). Example: `os=redhat` |
| os-user=<str | Operating system user that will be used by CMON to perform automation tasks like cluster recovery, backups and upgrades. This user must be able to perform super-user activities. Using root is recommended. Example: `os_user=root` Other aliases: `os_user`, `ssh_user` |
| sudo=<com | The command used to obtain superuser permissions. If sudo user requires a password, specify the sudo command with sudo password here. The sudo command must be trimmed by redirecting stderr to somewhere else. Therefore, it is compulsory to have `-S 2>/dev/null` appended in the `sudo` command. Example: `sudo="echo 'My5ud0'` |
| os-user_home | The home directory of the user used on nodes. Example: `osuser_home=/home/ubuntu/` Other aliases: `os_user_home`, `ssh_user_home` |
| software_pack | This is the storage location of software packages, i.e, all necessary files to successfully install a node, if there is no yum/apt repository available, must be placed here. Applies mainly to MySQL Cluster or older Codership/Galera installations. |
| local_repo_ | The used local repository names for cluster deployment. Example: `local_repo_name=epel` |
| init_name | The OS service name used for starting/stopping the database servers. Another alias: `init_service_name` Example: `init_name=postgresql-9.6` |
| disable_numa integer> | Not to use NUMA support dependent features. Default is 1 (enabled). |

### SSH

| Option | Description |
|---|---|
| ssh_identi | The SSH key or key pair file will be used by CMON to connect managed nodes (including ClusterControl node) passwordless. If undefined, CMON will use the home directory of `os_user` and look for `.ssh/id_rsa` file. Example: `ssh_identity=/root/.ssh/id_rsa` Other aliases: `ssh_keypath`, `identity_file` |
| ssh_port=< | The SSH port used by CMON to connect to managed nodes. If undefined, default to 22. Example: `ssh_port=22` |
| ssh_acquir integer> | Setting for `libssh` - should it acquire a remote tty. Default is 1 (true). Example: `ssh_acquire_tty=1` |
| ssh_passwo | The SSH password used for connection to nodes. This is not recommended, hence ClusterControl enforces the passwordless SSH instead. |
| ssh_timeou | Network timeout value in seconds for SSH connections. Default is 30. Example: `ssh_timeout=30` Another alias: `libssh_timeout=<integer>` |
| libssh_loglev | Setting for `libssh` logging verbosity to stdout. Accepted values are 0 (NONE), 1 (WARN), 2 (INFO), 3 (DEBUG), 4 (TRACE). Example: `libssh_loglevel=2` |
| slow_ssh_w | A warning alarm will be raised if it takes longer than the specified time to set up an SSH connection (secs). Default is 6 seconds. Example: `slow_ssh_warning=6` |
| slow_ssh_c | A critical alarm will be raised if it takes longer than the specified time to set up an SSH connection (secs). Default is 12 seconds. - Example: `slow_ssh_critical=12` |

### Automatic Recovery

| Option | Description |
|---|---|
| en-able_cluster_a integer> | If undefined, CMON defaults to 0 (false) and will NOT perform automatic recovery if it detects cluster failure. The supported value is 1 (cluster recovery is enabled) or 0 (cluster recovery is disabled). |
| en-able_node_auto integer> | If undefined, CMON defaults to 0 (false) and will NOT perform automatic recovery if it detects node failure. The supported value is 1 (node recovery is enabled) or 0 (node recovery is disabled). |
| en-able_autorecov integer> | If undefined, CMON defaults to 0 (false) and will NOT perform automatic recovery if it detects node or cluster failure. The supported value is 0 (cluster and node recovery are disabled) or 1 (cluster and node recovery are enabled). This setting will internally set enable_node_autorecovery and en-able_cluster_autorecovery to the specified value. |
| node_recovery_ | Specify a lock file and if present on a node, the node will not recover. The administrator is responsible to create/remove the file. Example: node_recovery_lock_file=/root/do_not_recover |
| node_recovery_ | A timeout value for waiting the new node to reach the synchronized state. Default : 28800. |
| add_cluster_ti | Timeout value for cluster start during deployment (currently NDB only). Default : 28800. |
| send_clear_clu integer> | Send notification email if a cluster failure event is cleared (meaning the cluster is recovered). Default is true. Another alias: send_clear_alarm |
| replica-tion_stage_fai | Automatically stops a slave node if any cluster's stage job fails. Default: false. |
| replica-tion_failover_ | Number of extra db_stats_collection_interval rounds to wait before proceeding with failover. Default: 0. |

### Monitoring and Thresholds

| Option | Description |
|---|---|
| monitored_mountpoints=<paths> | The MySQL/MongoDB/PostgreSQL data directory used by database nodes for disk performance in a comma-separated list. Example: monitored_mountpoints=/var/lib/mysql,/mnt/data/mysql. Another alias: monitored_mount_points |
| monitored_nics=<string> | List of network interface names to be monitored for network performance in a comma-separated list. Example: monitored_nics=eth1,eth2 |
| db_stats_collection_interval=<integer | Database metrics sampling interval in seconds. The lowest value is 1. Default is 30 seconds. Example: db_stats_collection_interval=30 |
| host_stats_collection_interval=<integ | Host metrics sampling interval in seconds. The lowest value is 1. Default is 30 seconds. Example: host_stats_collection_interval=30 |

continues on next page

Table 4.3 – continued from previous page

| Option | Description |
|---|---|
| host_stats_window_size=<integer> | Setting the window size (in seconds) to examine (CPU, memory..) stats to raise/clear host stats alarms. The default is 180 seconds. Example: host_stats_window_size=300 |
| lb_stats_collection_interval=<integer> | Load balancer stats collection interval. Default is 15. Example: lb_stats_collection_interval=30 |
| db_schema_stats_collection_interval=< | How often database growth and table checks are performed in seconds. This translates to information_schema queries. Default is 10800 seconds (3 hours). 0 means disabled. Example: db_schema_stats_collection_interval=10800 |
| db_proc_stats_collection_interval=<in | Setting for database process stats collection interval. The default is 3 seconds. The minimum allowed value is 1 second. Example: db_proc_stats_collection_interval=5 |
| db_log_collection_interval=<integer> | Database log files collection interval. Default is 600. Example: db_log_collection_interval=600 |
| db_deadlock_check_interval=<integer> | How often to check for deadlocks in seconds. Deadlock detection will affect CPU usage on database nodes. Default is 0, which means disabled. Example: db_deadlock_check_interval=600 |
| db_exporter_user=<string> | Database user for Prometheus exporter. Default is db_exporter_user=cmonexporter |
| db_exporter_password=<string> | Password for db_exporter_user Example: db_exporter_password=myS3cret |
| db_exporter_use_nonlocal_address=<boo integer> | Specifies if Prometheus exporter should connect to the non-local address of the DB services, instead of 127.0.0.1. Default is 0 (false). Example: db_exporter_use_nonlocal_address=1 |
| db_hourly_stats_collection_interval=< | Database statistic collections interval in seconds. Default is 5. Example: db_hourly_stats_collection_interval=5 |
| swap_warning=<integer> | Warning alarm threshold for swap usage. Default is 5. Example: swap_warning=20 |
| swap_critical=<integer> | Critical alarm threshold for swap usage. Default is 20. Example: swap_critical=40 |
| swap_inout_period=<integer> | The interval for swap I/O alarms in seconds. 0 means disabled. Default is 600 (10 minutes). Example:swap_inout_period=120 |
| swap_inout_warning=<integer> | The number of pages swapped I/O for warning in the specified swap_inout_period. The default is 10240. To determine the page size for the host, use getconf PAGESIZE. Example: swap_inout_warning=51200 |

continues on next page

Table 4.3 – continued from previous page

| Option | Description |
|---|---|
| swap_inout_critical=<integer> | The number of pages swapped I/O for critical in the specified swap_inout_period. The default is 102400. To determine the page size for the host, use getconf PAGESIZE. Example: swap_inout_critical=102400 |
| save_history_days=<integer> | How many days the controller shall keep monitoring data collected by ClusterControl like database growth, host/database stats and logs. Default is 7. 0 means disabled. |
| mysqlmemory_warning=<integer> | Warning alarm threshold for MySQL memory. Default is 80. |
| mysqlmemory_critical=<integer> | Critical alarm threshold for MySQL memory. Default is 90. |
| ram_warning=<integer> | Warning alarm threshold for RAM usage. Default is 80. |
| ram_critical=<integer> | Critical alarm threshold for RAM usage. Default is 90. |
| diskspace_warning=<integer> | Warning alarm threshold for disk usage. Default is 80. |
| diskspace_critical=<integer> | Critical alarm threshold for disk usage. Default is 90. |
| cpu_warning=<integer> | Warning alarm threshold for CPU usage. Default is 80. |
| cpu_critical=<integer> | Critical alarm threshold for CPU usage. Default is 90. |
| cpu_steal_warning=<integer> | Warning alarm threshold for CPU steal. Default is 10. |
| cpu_steal_critical=<integer> | Critical alarm threshold for CPU steal. Default is 20. |
| cpu_iowait_warning=<integer> | Warning alarm threshold for CPU IO Wait. Default is 50. |
| cpu_iowait_critical=<integer> | Critical alarm threshold for CPU IO Wait. Default is 60. |
| monitor_cpu_temperature=<boolean integer> | Whether to monitor CPU temperature. Default is 0 (false). |
| redobuffer_warning=<integer> | Warning alarm threshold for redo buffer usage. Default is 80. |
| redobuffer_critical=<integer> | Critical alarm threshold for redo buffer usage. Default is 90. |
| indexmemory_warning=<integer> | Warning alarm threshold for index memory usage. Default is 80. |
| indexmemory_critical=<integer> | Critical alarm threshold for index memory usage. Default is 90. |
| datamemory_warning=<integer> | Warning alarm threshold for data memory usage. Default is 80. |
| datamemory_critical=<integer> | Critical alarm threshold for data memory usage. Default is 90. |

continues on next page

Table 4.3 – continued from previous page

| Option | Description |
|---|---|
| tablespace_warning=<integer> | Warning alarm threshold for table space buffer memory usage. Default is 80. |
| tablespace_critical=<integer> | Critical alarm threshold for table space buffer memory usage. Default is 90. |
| redolog_warning=<integer> | Warning alarm threshold for redo log usage. Default is 80. |
| redolog_critical=<integer> | Critical alarm threshold for redo log usage. Default is 90. |
| enable_is_queries=<boolean integer> | Specifies whether queries to the information_schema will be executed or not. Queries to the information_schema may not be suitable when having many schema objects (hundreds of databases, hundreds of tables in each database, triggers, users, events, stored procedures). If disabled, the query that would be executed will be logged so it can be determined if the query is suitable in your environment. Default is 1 (enabled). Disable with 0. |
| max_replication_lag=<integer> | Max allowed replication lag in seconds before sending an alarm. Default is 10. |
| enable_icmp_ping=<boolean integer> | Toggles if the controller shall measure the ICMP ping times to the host. Default is 1 (true). |
| galera_flush_status_interval=<integer | Interval in seconds for executing FLUSH STATUS on Galera to reset status variables. Default: 0 (disabled). |
| lb_pre_install_script=<path> | Script executed before load balancer installation/setup. |
| lb_post_install_script=<path> | Script executed after load balancer installation/setup. |
| lb_pre_start_script=<path> | Script executed before load balancer node start. |
| lb_post_start_script=<path> | Script executed after load balancer node start. |
| lb_backend_draining_grace_sec=<intege | Max time (seconds) a load-balancer backend can stay in draining/OFFLINE_SOFT before alarm. 0 disables. Default: 300. |

### Query Monitor

| Option | Description |
|---|---|
| long_query_time=<f | The threshold value for slow query checking. Default 0.5. Example: long_query_time=0.0003 |
| log_queries_not_us integer> | Set query monitor to detect queries not using indexes. Default is 0 (false). |
| query_monitor_use_ integer> | Do not override the local settings of MySQL's long_query_time and log_queries_not_using_indexes. Default is 0 (false). |
| en- able_query_monitor | Setting for query monitor interval in seconds. Default is 1 means enabled. -1 means disable. Another alias: query_monitor_enabled |
| en- able_query_monitor integer> | If enabled the Performance_Schema table events_statements_summary_by_digest will be auto-purged (using TRUNCATE TABLE) every hour. By default, this is disabled (false). Enable by setting it to '1' (true). Another alias: query_monitor_auto_purge_ps |
| query_monitor_long | Raises an alarm if a query executes for longer than this value in milliseconds. The default is 10000. The minimum value is 1000. |
| query_monitor_aler integer> | Raises an alarm if a query executes for longer than query_monitor_long_running_query_ms. Default is 0 (disabled). Enable with 1. |
| query_monitor_kill integer> | Kill the query if the query executes for longer than query_monitor_long_running_query_ms. Default is 0 (Disabled). Enable with 1. |
| query_monitor_long | Match only queries with an "Info" only matching this regex. No default value means match any Info. Example: `query_monitor_long_running_query_matching_info=(?i)(^SELECT.* |
| query_monitor_long | The opposite matching of query_monitor_long_running_query_matching_info. Example: `query_monitor_long_running_query_matching_info_negate=(?i)(^ALTER TABLE.* |
| query_monitor_long | Match only queries with a "Host" only matching this regex. No default value means match any Host. Example: `query_monitor_long_running_query_matching_host=(192.168.10..* |
| query_monitor_long | Match only queries with a "Db" only matching this regex. No default value means match any Database. Example: query_monitor_long_running_query_matching_db=^joomla_.* |
| query_monitor_long | Match only queries with a "User" only matching this regex. No default value means match any User. Example: `query_monitor_long_running_query_matching_user=(dbuser1 |
| query_monitor_long | The opposite matching of query_monitor_long_running_query_matching_user. Example: `query_monitor_long_running_query_matching_user_negate=(monitoring_user |
| query_monitor_long | Match only queries with a "Command" only matching this regex. Defaults to "Query". Example: `query_monitor_long_running_query_matching_command=(?i)(QUERY |

### Backup

| Option | Description |
|---|---|
| net-cat_port=<str | List of netcat/socat ports and port ranges used to stream backups. The first value before a comma is the preferred port. The next value is a port range where ClusterControl should pick on. Defaults to '9999,9990-9998' which means port 9999 will be preferred if available, otherwise pick the next available port in the defined range. Example: `netcat_port=9999,9990-9998` |
| backup_user=< | The database username for managing backups. Example: `backup_user=backupuser` |
| backup_user_p | The database password for the `backup_user`. Example: `backup_user_password=MyS3cretC` |
| backup_encryp | The AES encryption key used for backups in base64. |
| back-updir=<path> | The default backup directory, to be pre-filled in ClusterControl UI. Example:`backupdir=/storage/backup` |
| backup_subdir | Set the name of the backup subdirectory. For more details on the formatting, see *Backup Subdirectory Variables*. Default is "`BACKUP-%I`". Example: `backup_subdir=BACKUP-%I-%D` |
| backup_retent | How many days to keep the backups. Backups matching retention period are removed. Default is 31. Example: `backup_retention=15` |
| backup_cloud_ | How many days to keep the uploaded backups to the cloud. Backups matching retention period are removed. Default is 180. Example: `backup_cloud_retention=90` |
| backup_n_safe | How many completed full backups will be kept regardless of their retention status. Default is 1. Example: `backup_n_safety_copies=3` |
| datadir_backu | During restore/rebuild operations a backup (filesystem copy) of the existing data directory (datadir) may be performed (user decides). Unless specified, the default data directory backup location is `DATADIR_bak`, e.g `/var/lib/mysql_bak` if the `datadir` is `/var/lib/mysql`. |
| dis-able_backup_e integer> | This setting controls if emails are sent or not if a backup finished or failed. This feature is disabled by default, meaning backup emails are sent. Example: `disable_backup_email=1` Another alias: `disable_backup_emails` |
| backup_create integer> | Configures CMON if it has to calculate MD5 checksum on the created backup files and verify them. Default is 1 (true). Checksumming is a resource-intensive process and disabling this process might save some resources, especially on a huge backup. Example: `backup_create_hash=0` |
| backup_pre_sc | Script executed before backup happens. If it returns non-zero, the backup will be aborted. If the script does not exist, the backup will be aborted. Arguments passed: backup id, method, type, hostname, storage hostname, storage dir. |
| backup_post_s | Script executed after backup happens. If it returns non-zero, the backup will be aborted. If the script does not exist, the backup will be aborted. Arguments passed: backup id, method, type, hostname, storage hostname, storage dir, status. |
| backup_disksp | Sets the lower bound of free disk space (%) required for backups to start. If below this threshold, backup fails. Default: 0. |
| backup_delete | Max number of attempts for retrying DELETE_ALL_BACKUPS jobs (triggered on cluster drop). Default: 3. |
| backup_delete | Delay (in seconds) between retries for DELETE_ALL_BACKUPS jobs. Default: 1. |
| pgback-rest_cipher_p | The AES key to encrypt the PgBackRest backup repository. |
| pgback-rest_cipher_t | Encryption cipher used for the PgBackRest repository. |
| pgback-rest_repo_hos | Repository host for storing PgBackRest backup data. |
| pgback-rest_repo_pat | Directory path for PgBackRest repository data. |

### MySQL, MariaDB Nodes

| Option | Description |
|---|---|
| mysql_server_addresses=<string> | A comma-separated list of MySQL hostnames or IP addresses (with or without port is supported). For MySQL Cluster, this should be the list of MySQL API node IP addresses. In the case of Galera Cluster, you can add?slave or ?bvs (backup verification server) to the URL so ClusterControl will register the node accordingly. Example: mysql_server_addresses=192.168.0.11:3306,192.168.0.12:3306,192.168.0.13:3306,192.168.0.14:3306?slave |
| monitored_mysql_port=<integer> | MySQL port for the managed cluster. ClusterControl assumes all DB nodes are listening on the same port. Default is 3306. Example: monitored_mysql_port=3306 Other aliases: cmon_local_mysql_port,local_mysql_port |
| monitored_mysql_root_user=<string> | MySQL root user for the managed cluster. ClusterControl assumes all DB nodes share the same root user. The user must have the same privileges as root (SUPER with GRANT OPTION). This is required when you want to scale your cluster by adding a new database node or replica. Default is "root". Example: monitored_mysql_root_user=dbadmin |
| monitored_mysql_root_password=<string> | MySQL root password for the managed cluster. ClusterControl assumes all DB nodes share the same root password. This is required when you want to scale your cluster by adding a new database node or replica. Example: monitored_mysql_root_password='r00tP$@^%sw0rd' |
| skip_name_resolve=<boolean integer> | Setting to disable name resolution. Default is 1 means skip_name_resolve is enabled. |
| mysql_version=<string> | The MySQL/MariaDB major version. Example: mysql_version='8.0' Other alias: server_version |
| galera_port=<integer> | Exclusive for Galera Cluster. The Galera communication port to be used when adding nodes/garbd, and constructing wsrep_cluster_address. Default is 4567. |
| galera_version=<string> | Exclusive for Galera Cluster. The Galera API major version number. Example: galera_version=3.x |
| galera_vendor=<string> | Exclusive for Galera Cluster. The database vendor name. Supported values are "percona", "codership" and "mariadb". Example: galera_vendor=mariadb |

continues on next page

Table 4.4 – continued from previous page

| Option | Description |
|---|---|
| repl_user=<string> | The MySQL replication user. Example: repl_user=repluser |
| repl_password=<string> | Password for repl_user Example: repl_password='ZG04Z2Jnk0MUWAZK' |
| replication_failover_whitelist=<strin | A comma-separated list of MySQL slaves that should be used as potential master candidates. If no server on the whitelist is available (up/connected) the failover will fail. If this variable is set, only those hosts will be considered. This parameter takes precedence over replication_failover_blacklist Example: replication_failover_whitelist=192.168.1.11,192.168.1.12 |
| replication_failover_blacklist=<strin | A comma-separated list of MySQL replicas that will never be considered a primary candidate. You can use it to list replicas that are used for backups or analytical queries. If the hardware varies between replicas, you may want to put here the replica which use slower hardware. replication_failover_whitelist takes precedence over this parameter if it is set. Example: replication_failover_blacklist=192.168.1.101,192.168.1.102 |
| replication_skip_apply_missing_txs=<b integer> | Default is 0. Skip the check process for additional missing transactions before promoting a slave to a master and just use the most advanced slave. Such a process may result in a serious problem though - if an errant transaction is found, replication may be broken. Example: replication_skip_apply_missing_txs=1 |
| replication_stop_on_error=<boolean integer> | Controls if the failover/switchover procedures should fail if errors are encountered that may cause data loss. Default : true. |
| replication_failover_wait_to_apply_ti | The candidate waits up to this many seconds to apply outstanding relay log events (retrieved_gtids) before failing over. Default is -1, which means ClusterControl will wait indefinitely for it to apply all missing transactions from its relay logs. This is safe, but, if for some reason, the most up-to-date replica is lagging badly, failover may take hours to complete. If set to 0, failover happens immediately, no matter if the master candidate is lagging or not. Default -1 seconds (wait forever). A value higher than 0 means ClusterControl will wait for the specified seconds before failover happens. Example: replication_failover_wait_to_apply_timeout=0 |

continues on next page

Table 4.4 – continued from previous page

| Option | Description |
|---|---|
| replication_auto_rebuild_slave=<boole integer> | If the SQL THREAD is stopped and the error code is non-zero then the replica will be automatically rebuilt. 1 means enable, default is 0 (false). Example: replication_auto_rebuild_slave=1 |
| replication_onfail_failover_script=<p | Path to the failover script on ClusterControl node. This script executes as soon as it has been discovered that failover is needed. If the script returns non-zero it will force the failover to abort. If the script is defined but not found, the failover will be aborted. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: failover_script.sh "arg1" "arg2" "arg3" "arg4". The script must be accessible on the controller and executable. Example: replication_onfail_failover_script=/usr/local/bin/failover_script.sh |
| replication_pre_failover_script=<path | Path to the failover script on ClusterControl node. This script executes before the failover happens, but after a candidate has been elected and it is possible to continue the failover process. If the script returns non-zero it will force the failover to abort. If the script is defined but not found, the failover will be aborted. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: pre_failover_script.sh "arg1" "arg2" "arg3" "arg4". The script must be accessible on the controller and executable. Example: replication_pre_failover_script=/usr/local/bin/pre_failover_script.sh |

continues on next page

Table 4.4 – continued from previous page

| Option | Description |
|---|---|
| replication_post_failover_script=<pat | Path to the failover script on ClusterControl node. This script executes after the failover has happened. If the script returns non-zero, a warning will be written in the job log. The script must be accessible and executable on the controller. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: `post_failover_script.sh "arg1" "arg2" "arg3" "arg4"`. The script must be accessible on the controller and executable. Example: `replication_post_failover_script=/usr/local/bin/post_failover_script.sh` |
| replication_post_unsuccessful_failove | Path to the script on the ClusterControl node. This script is executed after the failover attempt failed. If the script returns non-zero a Warning will be written in the job log. The script must be accessible on the controller and executable. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: `post_fail_failover_script.sh "arg1" "arg2" "arg3" "arg4"`. The script must be accessible on the controller and executable. Example: `replication_post_unsuccessful_failover_script=post sh` |
| replication_failed_reslave_failover_s | Path to the script on the ClusterControl node. This script is executed after that a new master has been promoted and if the enslaving of the slaves to the new master fails. If the script returns non-zero a Warning will be written in the job log. The script must be accessible on the controller and executable. Example: `replication_failed_reslave_failover_script=/usr/local/bin/fail_reslave_failover_script.sh` |

continues on next page

Table 4.4 – continued from previous page

| Option | Description |
|---|---|
| replication_pre_switchover_script=<pa | Path to the switchover script on ClusterControl node. This script executes before the switchover happens. If the script returns non-zero it will force the switchover to fail. If the script is defined but not found, the switchover will be aborted. The script must be accessible on the controller and executable. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: `pre_switchover_script.sh "arg1" "arg2" "arg3" "arg4"`. The script must be accessible on the controller and executable. Example:`replication_pre_switchover_script=/usr/local/bin/pre_switchover_script.sh` |
| replication_post_switchover_script=<p | Path to the switchover script on ClusterControl node. This script executes after the switchover happened. If the script returns non-zero a Warning will be written in the job log. The script must be accessible on the controller and executable. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: `post_switchover_script.sh "arg1" "arg2" "arg3" "arg4"`. The script must be accessible on the controller and executable. Example: `replication_post_switchover_script=/usr/local/bin/post_switchover_script.sh` |
| replication_check_external_bf_failove integer> | Before attempting a failover, perform extended checks by checking the slave status to detect if the master is truly down, and also check if ProxySQL (if installed) can still see the master. If the master is detected to be functioning, then no failover will be performed. Default is 0 (false) meaning the checks are disabled. 1 means enable. Example: `replication_check_external_bf_failover=0` |
| replication_check_binlog_filtration_b integer> | Before attempting a failover, verify filtration (`binlog_do`/`ignore_db`) and `replication_*` are identically configured on the candidate and the slaves. Default is 0 (false) meaning the checks are disabled. 1 means enable. Example: `replication_check_binlog_filtration_bf_failover=1` |

continues on next page

Table 4.4 – continued from previous page

| Option | Description |
|---|---|
| replication_failover_events=<boolean integer> | Automatically failover events (SLAVE-SIDE_DISABLED) and enable the event_scheduler after a replication failover/switchover action. Default is disabled. Enabled by setting it to 1. Example: replication_failover_events=1 |
| replication_failover_dead_source_grac | The number of seconds the apply-relay wait tolerates with no progress at all (no position movement, no GTID movement, no InnoDB row activity, every applier parked at the end of the relay log, and the source unreachable) before it stops waiting and promotes with what is applied. Default is 90. Disabled by setting it to 0. If set to 0, the detection is off it will wait for the full replication_failover_wait_to_apply_timeout. Example: replication_failover_dead_source_grace=90 |
| auto_manage_readonly=<boolean integer> | Enable/Disable automatic management of the MySQL server read_only variable. Default is 1 (true), which means ClusterControl will set the read_only=ON if the MySQL replication role is a slave. Default is 1 (enabled). Example: auto_manage_readonly=0 |
| schema_change_detection_address=<stri | This option must be defined to use "Operational Report for Schema Change". Creating a report of thousands of database objects (schemas, tables, etc) will take some time (about 5-10 minutes) depending on the hardware. It's recommended to configure a specific host to run this job for example on a replication slave or an asynchronous slave connected to e.g a Galera or Group Replication Cluster. For NDB this option should be set to a MySQL server used for admin purposes. Example: schema_change_detection_address=192.168.111.53 |
| schema_change_detection_pause_time_ms= | Throttle the detection process by pausing every this value, in milliseconds. For example, if defined as 3000, ClusterControl will pause the operation for every 3 seconds. Example: schema_change_detection_pause_time_ms=3000 |
| schema_change_detection_databases=<st | A comma-separated string of database names and also supports wildcards. For example "DB%", will evaluate all databases starting with "DB". Example:schema_change_detection_databases=mydb%,shops_db,mymonitoring |

continues on next page

Table 4.4 – continued from previous page

| Option | Description |
|---|---|
| datanode_addresses=<string> | Exclusive for MySQL Cluster. Comma-separated list of data node hostnames or IP addresses. Example: datanode_addresses=192.168.0.41,192.168.0.42 |
| mgmnode_addresses=<string> | Exclusive for MySQL Cluster. Comma-separated list of management node hostnames or IP addresses. Example: mgmnode_addresses=192.168.0.51,192.168.0.5.2 |
| ndb_connectstring=<string> | Exclusive for MySQL Cluster. NDB connection string for the cluster. Example: ndb_connectstring=192.168.0.51:1186,192.168.0.52:1186 |
| ndb_binary=<string> | Exclusive for MySQL Cluster. NDB binary for data node. Supported values are ndbd or ndbmtd. Example: ndb_binary=ndbmtd |
| ndbd_datadir=<path> | Exclusive for MySQL Cluster. The datadir of the NDBD nodes. Example: ndbd_datadir=/var/lib/mysqlcluster |
| mgmd_datadir=<path> | Exclusive for MySQL Cluster. The datadir of the NDB MGMD nodes. Example: mgmd_datadir=/var/lib/mysql |
| db_configdir=<string> | Exclusive for MySQL Cluster. The datadir of the NDB MGMD nodes. Example: db_configdir=/etc/mysql-cluster |
| percona_client_id=<string> | Percona client identifier. |
| percona_pro_token=<string> | Token to access Percona Pro repository. |

### MongoDB Nodes

| Option | Description |
|---|---|
| mongodb_server_addr | Comma-separated list of MongoDB shard or replica IP addresses with port. Example: mongodb_server_addresses=192.168.0.11:27017,192.168.0.12:27017,192.168.0.13:27017 |
| mongoarbiter_server_add | Comma-separated list of MongoDB arbiter IP addresses with port. Example: mongoarbiter_server_addresses=192.168.0.11:27019,192.168.0.12:27019,192.168.0.13:27019 |
| mongocfg_server_add | Comma-separated list of MongoDB config server IP addresses with port. Example: mongocfg_server_addresses=192.168.0.11:27018,192.168.0.12:27018,192.168.0.13:27018 |
| mongos_server_addre | Comma-separated list of MongoDB mongos IP addresses with port. Example: mongos_server_addresses=192.168.0.11:27017,192.168.0.12:27017,192.168.0.13:27017 |
| mongodb_basedir=<pa | Location MongoDB base directory to find the client-related binaries. Example: mongodb_basedir=/usr |
| mongodb_user=<strin | MongoDB admin/root username. Example: mongodb_user=root |
| mongodb_password=<s | Password for mongodb_user Example: mongodb_password='kPo123^^#*' |
| mongodb_authdb=<str | The database containing user information to use for authentication. Default is admin. Example: mongodb_authdb=admin |
| mongodb_cluster_key | The cluster's nodes authenticating to each other using this key. Example: mongodb_cluster_key=/etc/repl.key |
| server_selection | Mongo connection URI option. Defines the timeout value in milliseconds till the mongo driver should try to do a successful server selection operation. The default is 30000. Example: server_selection_timeout_ms=40000 |
| server_selection integer> | Mongo connection URI option. Defines if server selection should be repeated on failure until a server selection timeout expires, or just return with failure at once. Default is 1 (true). Example: server_selection_try_once=0 |

### PostgreSQL, TimescaleDB Nodes

| Option | Description |
|---|---|
| post-gresql_s | Comma separated list of PostgreSQL instances with port. Example: postgresql_server_addresses=192.168.10.100:5432,192.168.10.101:5432 Other alias: postgre_server_addresses |
| post-gresql_u | The PostgreSQL admin user name. Default is Postgres. Example: post-gresql_user=postgres Other alias: postgre_user |
| post-gresql_p | The PostgreSQL admin password. Example: post-gresql_password='p4s$^#0rd123' Other alias: postgre_password |
| wal_rete inte-ger> | Retention hours to erase old WAL archive logs for PITR. Default is 0, which means WAL archive logs are kept forever. Other alias: pitr_retention_hours |
| repli-ca-tion_aut inte-ger> | If the SQL THREAD is stopped and the error code is non-zero then the slave will be automatically rebuilt. 1 means enable, default is 0 (false). Example: replica-tion_auto_rebuild_slave=1 |
| repl_use | The PostgreSQL replication user. Example: repl_user=repluser |
| repl_pas | Password for repl_user Example: repl_password='ZG04Z2Jnk0MUWAZK' |
| repli-ca-tion_fai | Comma-separated list of PostgreSQL slaves which should be used as potential master candidates. If no server on the whitelist is available (up/connected) the failover will fail. If this variable is set, only those hosts will be considered. This parameter takes precedence over replication_failover_blacklist. Example: replication_failover_whitelist=192.168.1.11,192.168.1.12 |
| repli-ca-tion_fai | Comma-separated list of PostgreSQL slaves which will never be considered a master candidate. You can use it to list slaves that are used for backups or analytical queries. If the hardware varies between slaves, you may want to put here the slaves which use slower hardware. replication_failover_whitelist takes precedence over this parameter if it is set. Example: replication_failover_blacklist=192.168.1.101,192.168.1.102 |
| repli-ca-tion_pre | Path to the failover script on ClusterControl node. This script executes before the failover happens, but after a candidate has been elected and it is possible to continue the failover process. If the script returns non-zero it will force the failover to abort. If the script is defined but not found, the failover will be aborted. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: pre_failover_script.sh "arg1" "arg2" "arg3" "arg4". The script must be accessible on the controller and executable. Example: replication_pre_failover_script=/usr/local/bin/pre_failover_script.sh |
| repli-ca-tion_pos | Path to the failover script on ClusterControl node. This script executes after the failover has happened. If the script returns non-zero, a warning will be written in the job log. The script must be accessible and executable on the controller. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: post_failover_script.sh "arg1" "arg2" "arg3" "arg4". The script must be accessible on the controller and executable. Example: replication_post_failover_script=/usr/local/bin/post_failover_script.sh |
| repli-ca-tion_pos | Path to the script on the ClusterControl node. This script is executed after the failover attempt failed. If the script returns non-zero a Warning will be written in the job log. The script must be accessible on the controller and executable. 4 arguments are supplied to the script: - arg1="All servers in the replication" - arg2="The failed master" - arg3="Selected candidate" - arg4="Slaves of old master" The arguments will be passed like this: post_fail_failover_script.sh "arg1" "arg2" "arg3" |

### Elasticsearch Nodes

| Option | Description |
|---|---|
| elastic_server_addresses=<string> | Comma-separated list of Elasticsearch server instances. |
| elastic_user=<string> | Elasticsearch user name. |
| elastic_password=<string> | Password for the Elasticsearch user. |
| elastic_https=<boolean> | Indicates whether ClusterControl communicates with Elasticsearch nodes via HTTPS. |
| elastic_certs_password=<string> | Password for TLS/HTTPS certificates generation. |
| elastic_force_built_in_user=<boole | Force use of Elasticsearch built-in user credentials. |
| elastic_shared_filesystem=<string> | Configured shared filesystems in cluster. Format: :,:. |
| elastic_snapshot_repository=<strir | Configured Elasticsearch snapshot repository name. |
| elastic_snapshot_repository_type=< | Snapshot repo type: fs (shared filesystem) or fs-nfs (filesystem with managed NFS). |

### MSSQL Nodes

| Option | Description |
|---|---|
| license=<string> | License applied to MSSQL deployments. |
| mssql_server_addresses=<strin | Comma-separated list of MSSQL instances. |
| mssql_sa_username=<string> | The MSSQL sa username. |
| mssql_sa_password=<string> | The MSSQL sa password. |
| mssql_sysadmin_username=<stri | External sysadmin username allowed to connect. |
| mssql_sysadmin_password=<stri | Password for external sysadmin user. |
| mssql_backup_cert_password=<s | Password used to encrypt the MSSQL backup certificate created by CMON. |
| mssql_master_key_password=<st | Password for the MSSQL master key created by CMON. |

## 4.3.4 LDAP

ClusterControl has a built-in support for LDAP authentication and requires another configuration file called cmon-ldap.cnf. The configuration file resides under the /etc/cmon.d directory and comes by default with the package. When an LDAP configuration change event happens in ***ClusterControl GUI → User management → LDAP → LDAP Settings***, ClusterControl will commit the change into the controller's runtime and save the configuration directly into cmon-ldap.cnf.

LDAP integration is only available in the Enterprise edition.

**Example output**

The following is an example configuration if one has an Active Directory server integrated with ClusterControl:

```
#
```

```
# Enables LDAP authentication. The native authentication in ClusterControl
# will also work.
#
enabled = true

#
# Enter the LDAP or LDAPS Uniform Resource Identifier (URI), with the port
# number (if applicable)
#
ldapServerUri = "ldap://support-ad.severalnines.com"

#
# The "Distinguished Name" is used to bind the LDAP server. This user
# requires read access to all LDAP user and group entries to work
# correctly. ClusterControl must perform a LDAP search using the DN before
# any user can log in. This field is case-sensitive.
#
ldapAdminUser = "CN=Administrator,CN=Users,DC=severalnines,DC=com"

#
# Enter the password for the "Distinguished Name" login.
#
ldapAdminPassword = "<redacted>"

#
# Enter the user's "Relative Distinguished Name" (RDN) to locate this
# user's information. This field is case-sensitive.
#
ldapUserSearchRoot = "CN=Users,DC=severalnines,DC=com"

#
# The group's "Relative Distinguished Name" (RDN) is used to locate the
# group information. ClusterControl does not support LDAP users that do
# not belong to at least one LDAP group.
#
ldapGroupSearchRoot = "OU=Groups,DC=severalnines,DC=com"

[ldap_settings]
#
# This is used to filter the object class of the LDAP users. If empty,
# all object classes will be returned.
#
# ldapUserClassName = "objectclass=*"

#
# The LDAP attribute which should contain the username, separated by a
# comma (whitespace value is not allowed). For Active Directory, this is
# commonly "sAMAccountName" and "uid" for OpenLDAP. If more than one
# attribute is specified ClusterControl will attempt to look up all of␣
↪them
```

```
# with the first non-empty reply (in the particular order) used for the
# login username.
#
ldapUsernameAttributes = "sAMAccountName"

#
# The LDAP attribute which should contain the full name of the user,
# separated by a comma. For Active Directory, this is commonly
# "displayName" and "cn" for OpenLDAP. If more than one attribute is
# specified ClusterControl will attempt to look up all of them with the
# first non-empty reply (in the particular order) used for the user's full
# name.
#
ldapRealnameAttributes = "displayName"

#
# The LDAP attribute which should contain the email address of the user,
# separated by a comma. For Active Directory, this is commonly
# "userPrincipalName" and "mail" for OpenLDAP. If more than one attribute
# is specified, ClusterControl will attempt to look up all of them with␣
↪the
# first non-empty reply (in the particular order) used for the user's␣
↪email
# address.
#
ldapEmailAttributes = "userPrincipalName"

#
# Filter the object class for the LDAP groups. If empty, all object␣
↪classes
# will be returned.
#
# ldapGroupClassName = ""

#
# The LDAP attribute which represents the group's members, separated by
# a comma. For Active Directory, this is commonly "member" and "memberUid"
# for OpenLDAP. These attributes will be used to create combinations with
# the value of Group mapping attributes when querying the LDAP server to
# retrieve the group's information.
#
ldapMemberAttributes = "member"

#
# The LDAP attribute of which name holds the group membership, separated
# by a comma. For Active Directory, this is commonly "dn" and "uid" for
# OpenLDAP. The value of these attributes will be used to create
# combinations with Static member attributes to query the LDAP server to
# retrieve the group.
#
```

```
ldapGroupIdAttributes = "dn"

#
# The LDAP attribute where the name represents the group's name. This is
# commonly "cn". The value of this attribute will be used in the group
# mapping with "ClusterControl's Team" for authorization purposes.
#
ldapGroupNameAttribute = "cn"

#
# This variable specifies the connection timeout in seconds if the LDAP
# server is unreachable or takes too long to respond.
#
ldapNetworkTimeout = 5

#
# This is the current LDAP protocol version being used.
#
ldapProtocolVersion = 3

#
# This variable specifies the limit in seconds for an LDAP query to␣
↪finish,
# any query which takes longer will be aborted.
#
ldapQueryTimeLimit = 5

[security]

#
# This is the location of the CA certification file. (Only for LDAPS).
#
# caCertFile = "/etc/ssl/certs/ldap-ca.crt"

#
# This is the location of the certification file. (Only for LDAPS).
#
# certFile = "/etc/ssl/certs/ldap.crt"

#
# This is the location of the certificate key file. (Only for LDAPS).
#
# keyFile = "/etc/ssl/certs/ldap.pem"
[DBAadmins]
ldapGroupId   = "DBA"
cmonGroupName = "admins"
```

### 4.3.5 CMON Database

The CMON database is the persistent store for all monitoring data collected from the managed nodes, as well as all ClusterControl metadata (e.g. what jobs there are in the queue, backup schedules, backup statuses, etc.). The ClusterControl Controller requires a MySQL database running on `mysql_hostname` as defined in the configuration file. The database name and the user is "cmon" and is immutable. The CMON database dump files are shipped together with the ClusterControl Controller package and can be found under `/usr/share/cmon` once it is installed.

MySQL user "cmon" needs to have proper access to the CMON database by performing the following grant:

Grant all privileges to "cmon" at `hostname` value (as defined in CMON configuration file) on ClusterControl host:

```
GRANT ALL PRIVILEGES ON *.* TO 'cmon'@'<hostname>' IDENTIFIED BY '
↪<mysql_password>' WITH GRANT OPTION;
```

Grant all privileges for "cmon" at 127.0.0.1 on ClusterControl host:

```
GRANT ALL PRIVILEGES ON *.* TO 'cmon'@'127.0.0.1' IDENTIFIED BY '
↪<mysql_password>' WITH GRANT OPTION;
```

On every managed database server, grant all privileges to cmon at the controller's `hostname` value (as defined in CMON configuration file) on each of the managed database host:

```
GRANT ALL PRIVILEGES ON *.* TO 'cmon'@'<hostname>' IDENTIFIED BY '
↪<mysql_password>' WITH GRANT OPTION;
```

If one deploys a cluster using ClusterControl deployment wizard, the above privileges will be configured automatically.

## 4.4 ClusterControl GUI

### 4.4.1 Introduction

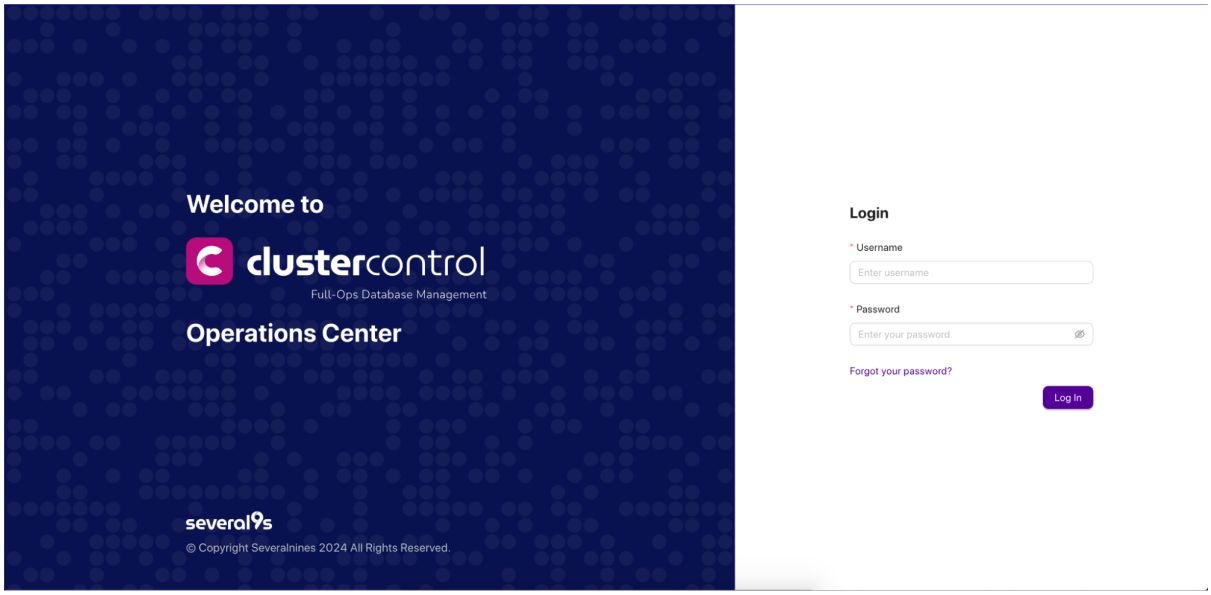
ClusterControl's Graphical User Interface (GUI) serves as a unified platform for managing database clusters. It streamlines complex database operations and ensures the reliability, performance, and security of database systems by providing a single interface to control multiple ClusterControl instances (controllers). The GUI is designed to handle large-scale environments, supporting over 1000 nodes.

The ClusterControl GUI is built upon the closely integrated `clustercontrol-mcc` and `clustercontrol-proxy` packages. This newer version of the `clustercontrol-mcc` web application no longer relies on an Apache server. The `cmon-proxy` service now serves the web application instead via a process called `ccmgr`, also known as **ClusterControl Manager**. By default, a ClusterControl installation registers the local controller.

**Info**

Prior to ClusterControl v2.3.2 (April 2025), ClusterControl GUI was provided by a different package called `clustercontrol2`. This package is still available in case users would like to use Apache as the web server (e.g, security compliance, advanced proxying and TLS, etc). However, `clustercontrol2` does not have the multi-controller and Kubernetes functionalities within.

ClusterControl GUI can be accessed via `https://<ClusterControl_host>` (or any other port passed during the initial setup). Use one of the ClusterControl users from the existing ClusterControl installation/environment to log in:




### 4.4.2 Files and location

The `clustercontrol-mcc` package installs the `clustercontrol-proxy` package as a dependency (the web server). The web application is located at `/var/www/html/clustercontrol-mcc` and the configuration file is located at `/var/www/html/clustercontrol-mcc/config.js`. Here is an example content:

```
window.FEAS_ENV = {
  CMON_API_URL: '/',
  GIT_SHA: 'f457257a3',
  GIT_BRANCH: 'release-mcc-2.3.3',
  VERSION: '2.3.3',
  BUILD_NUMBER: '541',
  USER_REGISTRATION: 1,
}
```

The service name for the web server is called `cmon-proxy` and the systemd unit definition file is located at `/etc/systemd/system/cmon-proxy.service`. The `cmon-proxy` is also responsible for managing, collecting data, and monitoring the local and external ClusterControl controllers. See *ClusterControl Proxy* for details.

### 4.4.3 Managing multiple controllers

A single ClusterControl GUI can be used to manage multiple ClusterControl controllers. ClusterControl's horizontal scaling is achieved through the ClusterControl Operation Center (Ops-C), which enables a single ClusterControl GUI to manage multiple ClusterControl controllers. This is ideal for large database farms with thousands of nodes.

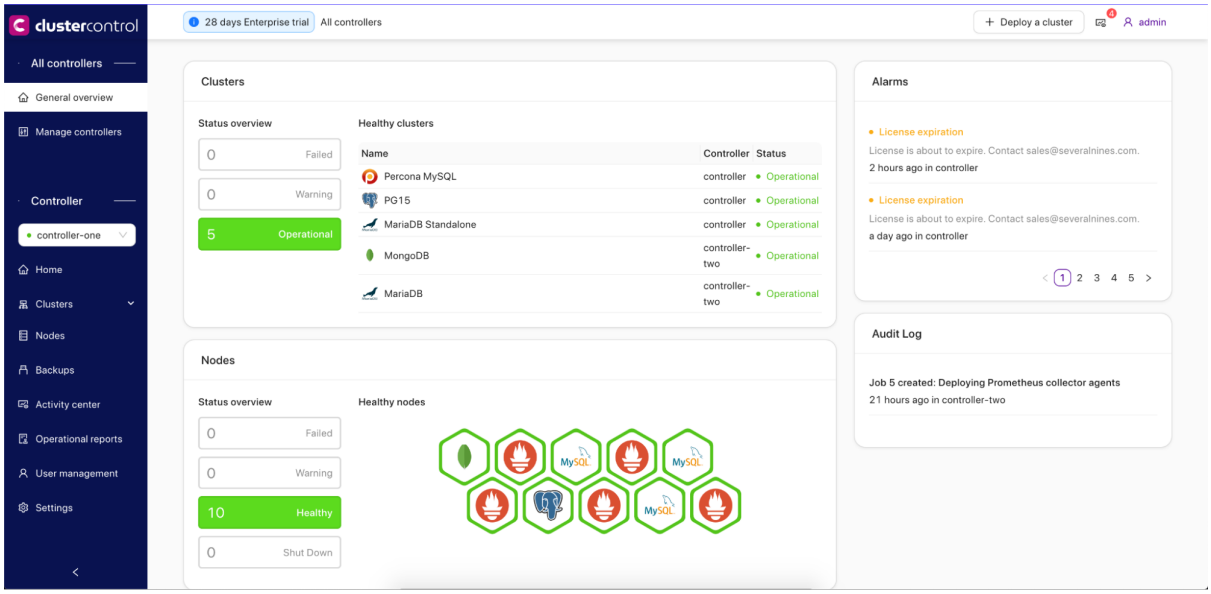
To set up Ops-C, you need to install additional ClusterControl servers, import or add clusters to them as you would with a standard ClusterControl operation. Then, activate the multi-controller GUI mode (Ops-C), allowing you to integrate these additional controllers into the same GUI for remote monitoring within a single interface.

By default, each ClusterControl GUI connects to its local controller in single-controller GUI mode. For details on activating multi-controller GUI mode (Ops-C), refer to *Integration → ClusterControl Ops-C*.

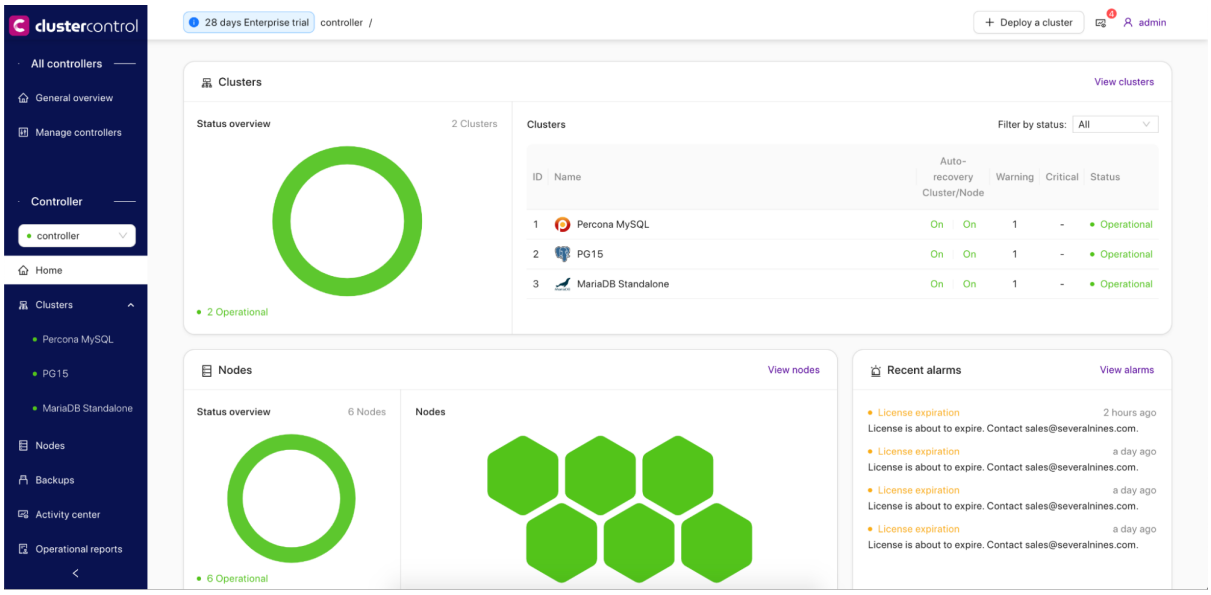
If multi-controller interface is enabled (default is disabled):

- The Controllers' Overview page features a dashboard that presents aggregated information on registered or managed CMON controllers, providing insights into the status of controllers, ongoing jobs, alarms, and backups, as well as the status of the database clusters and nodes.
- The web user interface allows users to select the "Active CMON Controller", which determines the information displayed throughout the application (excluding the Controllers' Overview) and directs user actions for clusters or nodes to the selected CMON Controller.

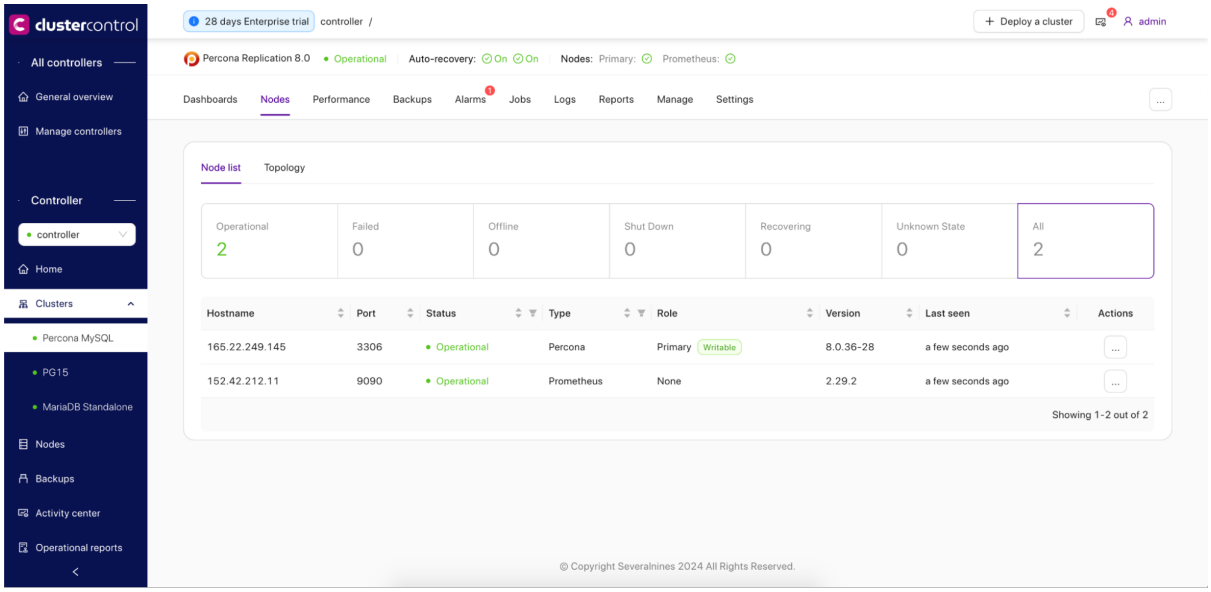
After all the ClusterControl has been added to the ClusterControl GUI, the Operations Center can give information about the status related to the Clusters, Nodes, and Controllers in the Overview Page:




Discover the status of Clusters and Nodes in specific controllers as shown below:




Go through the node list for specific clusters in the selected controller:




**See also**

- *Integration → ClusterControl Ops-C*
- *ClusterControl Proxy*

## 4.5 ClusterControl CLI

Also known as **s9s-tools**, this optional package is introduced in ClusterControl version 1.4.1, which contains a binary called `s9s`. It is a command-line tool to interact, control, and manage database clusters using the ClusterControl Database Platform. Starting from version 1.4.1, the installer script will automatically install this package on the ClusterControl node. You can also install it on another computer or workstation to manage the database cluster remotely. Communication between this client and the CMON controller is encrypted and secure through TLS. This command-line project is open source and publicly available at GitHub.

The command-line tool is invoked by executing a binary called `s9s`. The commands are basically JSON messages being sent over to the ClusterControl Controller (CMON) RPC interface. Communication between the s9s (the command-line tool) and the cmon process (ClusterControl Controller) is encrypted using TLS and requires port 9501 to be opened on the controller and the client host.

Following is the list of supported commands and options for ClusterControl CLI. You can also get the same information by accessing the manual page or using the `--help` flag in the terminal:

```
man s9s
s9s --help
```

### 4.5.1 s9s-account

Manage user accounts on clusters. The "account" term in this section can be treated as a database user account of a managed database server/cluster.

### Usage

```
s9s account {command} {options}
```

### Commands

| Name, shorthand | Description |
|---|---|
| `--create` | Creates a new account on the cluster. Note that the account is an account of the cluster and not a user of the Cmon system. |
| `--delete` | Removes an account. |
| `--list, -L` | Lists the accounts on the cluster. |
| `--grant` | Grant privileges for an account on one or more databases. |

### Options

| Name, shorthand | Description |
|---|---|
| `--accoun` | The account to be used or created on the cluster. The command-line option argument may contain a username, a password for the user and a hostname identifying the host from where the user may log in. The `s9s` command-line tool will handle the command line option argument as an URL encoded string, so if the password for example contains an `@` character, it should be encoded as `%40`. URL encoded parts are supported anywhere in the string, usernames and passwords (and even hostnames) may also have special characters. |
| `-l, --long` | This option will change its format to show more details. |
| `--privat` | Create a secure, more restricted account on the cluster. The actual interpretation of this flag depends on the controller, the current version is restricting access to the ProxySQL servers. The account that is created with the `--private` option will not be imported into the ProxySQL to have access through the ProxySQL server immediately after they are created on the cluster. |
| `--privil` | Privileges to be granted to a user account on the server. See *Privilege Expression*. |
| `--with-d` | Creates a database for the new account while creating a new user account on the cluster. The name of the database will be the same as the name of the account and all access rights will be granted for the account to use the database. |

### Privilege Expression

The privileges are specified using a simple language that is interpreted by the CMON Controller. The language is specified as follows:

```
expression: specification[;...]
specification: [object[,...]:]privilege[,...]
object: {
  *
  | *.*
```

(continues on next page)

(continued from previous page)

```
  | database_name.*
  | database_name.table_name
  | database_name
}
```

Note that an object name on itself is a database name (and not a table name) and multiple objects can be enumerated by using the `,` as a separator. It is also important that multiple specifications can be enumerated using the semicolon (`;`) as a separator.

The expression `MyDb:INSERT,UPDATE;Other:SELECT` for example, defines `INSERT` and `UPDATE` privileges on the MyDb database and `SELECT` privilege on the Other database. The expression `INSERT,UPDATE` on the other hand, would specify `INSERT` and `UPDATE` privileges on all databases and all tables.

### Examples

- Create a new MySQL user account `myuser` with password `secr3tP4ss`, and allow it to have `ALL PRIVILEGES` on database `shop_db` while `SELECT` on the table `account_db.payments`:

```
s9s account \
  --create \
  --cluster-id=1 \
  --account="myuser:Passw0rd999@192.168.0.100" \
  --privileges="shop_db.*:ALL;account_db.payments:SELECT"
```

- Create a new PostgreSQL user account called `mydbuser`, and allowed the host in the network subnet 192.168.0.0/24 to access the database `mydbshop`:

```
s9s account --create \
  --cluster-id=50 \
  --account='mydbuser:k#Passw0rd999@192.168.0.0/24' \
  --privileges="mydbshop.*:ALL"
```

- Delete a database user called `joe`:

```
s9s account \
  --delete \
  --cluster-id=1 \
  --account="joe"
```

- Lists the accounts on the cluster.

```
s9s account \
  --list \
  --long \
  --cluster-id=1
```

- Modify the existing MariaDB privileges of a user called `admin1@%` for cluster ID 4:

```
s9s account \
    --grant \
    --cluster-id=4 \
```

(continues on next page)

(continued from previous page)

```
    --account="admin1@%" \
    --privileges="REPLICATION SLAVE,PROCESS,CREATE USER,REPLICATION␣
↪SLAVE ADMIN,SLAVE MONITOR;mydb:ALL PRIVILEGES,GRANT OPTION;db1:ALL␣
↪PRIVILEGES,GRANT OPTION;db2:ALL PRIVILEGES,GRANT OPTION;db3:ALL␣
↪PRIVILEGES,GRANT OPTION;performance_schema:SELECT"
```

### 4.5.2 s9s-alarm

Manage alarms.

#### Usage

```
s9s alarm {command} {options}
```

#### Commands

| Name, shorthand | Description |
|---|---|
| `--delete` | Sets the alarm to be ignored. This will not in fact delete the alarm, but it will make the alarm to disappear from the active alarms list, hence the name of the option. |
| `--list` | Lists the active alarms. |
| `--stat` | Prints a simple list of the number of alarms. |

#### Options

| Name, shorthand | Description |
|---|---|
| `--cluster-id=ID`, `-i` | The ID of the cluster to manipulate. |
| `--cluster-name=NAM` `-n` | Sets the cluster name. If the operation creates a new cluster this will be the name of the new cluster. |
| `--alarm-id=ID` | The ID of the alarm to manipulate. |

#### Examples

- List out all alarms generated by ClusterControl for a database cluster named “PostgreSQL Cluster”:

```
s9s alarm --cluster-name="PostgreSQL Cluster" --list
```

- Delete an alarm:

```
s9s alarm --delete --alarm-id=1015
```

- Print a simple list of the number of alarms:

```
s9s alarm --stat
```

**Example output**

```
1,0,0
```

The list will contain three numbers for every line. The first is the cluster ID, the second is the number of critical alarms and the third is the warning level alarms for the given cluster.

### 4.5.3 s9s-backup

View and create database backups. Three backup methods are supported:

- mysqldump
- xtrabackup (full)
- xtrabackup (incremental)
- mariabackup (full)
- mariabackup (incremental)
- mongodump
- pg_dumpall
- pgdump

The s9s client also needs to know:

- The cluster ID (or cluster name) of the cluster to backup.
- The node to backup.
- The databases that should be included (default all databases).

By default, the backups will be stored on the controller node. If you wish to store the backup on the data node, you can set the flag `--on-node`.

**Note**

Note If you are using Percona Xtrabackup, an incremental backup requires that there is already a full backup made of the same databases (all or individually specified). Otherwise, the incremental backup will be upgraded to a full backup.

### Usage

```
s9s backup {command} {options}
```

### Commands

| Name, shorthand | Description |
|---|---|
| `--create` | Creates a new backup. This command-line option will initiate a new job that will create a new backup. |
| `--crea` | Creates a backup schedule, a backup that is repeated. Please note that there are two ways to create a repeated backup. A job that creates a backup can be scheduled and repeated and with this option a backup schedule can be created to repeat the creation of a backup. |
| `--dele` | Deletes an existing backup. |
| `--dele` | Initiates a job that checks for expired backups and removes them from the system. |
| `--dele` | Delete all backups (filtering by some property as snapshot repository name). This command line option will initiate a job that checks for all backups and removes them from the system.db_cluster_id property is used as it could be that no cluster exists but there is data on DB. With the forced option, it will delete a backup record on the DB even when backup deletion failed. |
| `--dele` | Creates a job to generate a backup only stored in the cloud, as after creating a backup on the storage location filesystem, the copy will be deleted after uploading it to the cloud. This option has to be set to indicate the deletion of the backup local copy after uploading it to the cloud. |
| `--dele` | Delete the backup schedule specified on the job id. |
| `--list` | Lists the backups. When listing the backups with the –long option, more detailed columns are listed. See Backup List. |
| `--list` | Lists the backups in database view format. This format is designed to show the archived databases in the backups. |
| `--list` | Lists the backups in file view format. This format is designed to show the archive files of the backups. |
| `--list` | Lists the backup schedules. |
| `--restore` | Restores an existing backup. |
| `--rest` | Restores the information the controller has about a cluster from a previously created archive file. |
| `--rest` | Restores the entire controller from a previously created tarball (created by using the –save-controller option). |
| `--pitr` | Timestamp specification for doing point-in-time backup recovery. |
| `--psql` | By default PostgreSQL when started in recovery mode after backup restoration applies the available complete WAL stream as well to provide the most up-to-date state of the database. By this option, we can ask PostgreSQL not to apply WAL or at least as much of it as it is a must for database consistency and to finish recovery as soon as possible. |
| `--save` | Saves information about one cluster. |
| `--save` | Saves the entire controller into a file. |
| `--verify` | Creates a job to verify a backup. When this main option is used the `--backup-id` the option has to be used to identify a backup and the `--test-server` is also necessary to provide a server where the backup will be tested |
| `--clou` | Creates a job to generate a backup streaming it directly to the cloud, without intermediate files `--cloud-only` option has to be set to indicate direct streaming of backup to the cloud container |
| `--crea` | Creates a job to create a snapshot repository on elasticsearch cluster. When this main option is used the `--cluster-id` option has to be used to identify the cluster, the `--snapshot-repository-type` defines the allowed type (example: “s3”) `--snapshot-repository` to specify the repository name, `--credential-id` to specify the cloud credentials to use, `--s3-bucket` is also necessary to provide s3 bucket to use for this repository. `--s3-region` is also necessary to provide s3 region to use for this repository. |
| `--list` | Creates a job to list the snapshot repositories on an elasticsearch cluster. When this main option is used the `--cluster-id` option has to be used to identify the cluster. |
| `--dele` | Creates a job to delete a snapshot repository on elasticsearch cluster. When this |

### Options

| Name, shorthand | Description |
|---|---|
| `--backup-directory=DIR` | The directory where the backup is placed. |
| `--backup-format[=FORMATSTRING]` | The string that controls the format of the printed information about the backups. See *Backup Format*. |
| `--backup-id=ID` | The ID of the backup. |
| `--backup-method=METHOD` | Controls what backup software is going to be used to create the backup. The controller currently supports the following methods: ndb, mysqldump, xtrabackupfull, xtrabackupincr, mariabackupfull, mariabackupincr, mongodump, pg_dump, pg_basebackup, mysqlpump. |
| `--backup-password=PASSWORD` | The password for the SQL account that will create the backup. This command-line option is not mandatory. |
| `--backup-retention=DAYS` | Controls a custom retention period for the backup, otherwise, the default global setting will be used. Specifying a positive number value here can control how long (in days) the taken backups will be preserved, -1 has a special meaning, it means the backup will be kept forever, while value 0 is the default, means prefer the global setting (configurable on UI). |
| `--backup-user=USERNAME` | The username for the SQL account that will create the backup. |
| `--cloud-retention=DAYS` | Retention used when the backup is on a cloud. |
| `--cluster-id=ID` | The ID of the cluster. |
| `--compression-level` | Compression level (used threads) to apply on the backup compression process. Value must be between1 and 9. |
| `--databases=LIST` | A comma-separated list of database names. This argument controls which databases are going to be archived into the backup file. By default, all the databases are going to be archived. |
| `--data-only` | Backup data only, no structure (PostgreSQL only). |
| `--encrypt-backup` | When this option is specified ClusterControl will attempt to encrypt the backup files using AES-256 encryption (the key will be auto-generated if not exists yet and stored in a cluster configuration file). |
| `--exclude-schemas=LIST` | Exclude specific schemas (PostgreSQL only). |
| `--full-path` | Print the full path of the files. |
| `--memory=MEGABYTES` | Controls how much memory the archiver process should use while restoring an archive. Currently, only the xtrabackup supports this option. |
| `--no-compression` | Do not compress the archive file. |
| `--no-owner` | Skip ownership information (PostgreSQL only). |
| `--no-privileges` | Skip privilege information (PostgreSQL only). |

continues on next page

Table 4.5 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--nodes=NODELIST` | The list of nodes involved in the backup. See *Node List.* |
| `--on-node` | Do not copy the created archive file to the controller, store it on the node where it was created. |
| `--on-controller` | Stream and store the created backup files on the controller. |
| `--parallelism=N` | Controls how many threads are used while creating a backup. Please note that not all the backup methods support multi-thread operations. |
| `--pitr-compatible` | Creates PITR-compatible backup. |
| `--pitr-stop-time` | Timestamp specification for doing point in time backup recovery. |
| `--recurrence=STRING` | Schedule time and frequency in cron format. |
| `--safety-copies=N` | Controls how many safety backups should be kept while deleting old backups. This command-line option can be used together with the `--delete-old` option. |
| `--schemas=LIST` | Include specific schemas (PostgreSQL only). |
| `--schemas-only` | Backup structure only, no data (PostgreSQL only). |
| `--subdirectory=MARKUPSTRING` | Sets the name of the subdirectory that holds the newly created backup files. The command-line option argument is considered to be a subpath that may contain the field specifiers using the usual %X format. See *Backup Subdirectory Variables.* |
| `--temp-dir-path=DIR` | By default, s9s backup creates temporary backup files to `/var/tmp/cmon-%` path. Specify this option with your desired path if you want to target to another location. |
| `--keep-temp-dir` | Specify this option if you want to retain your archive files from the temporary directory. |
| `--test-server=HOSTNAME` | Use the given server to verify the backup. If this option is provided while creating a new backup after the backup is created a new job is going to be created to verify the backup. During the verification, the SQL software will be installed on the test server and the backup will be restored on this server. The verification job will be successful if the backup is successfully restored. |
| `--title=STRING` | A short human-readable string that helps the user to identify the backup later. |
| `--to-individual-files` | Archive every database into individual files. Currently, only the mysqldump backup method supports this option. |
| `--use-pigz` | Use the pigz program to compress the archive. |

### Backup Subdirectory Variables

| Variable | Description |
|---|---|
| B | The date and time when the backup creation was beginning. |
| H | The name of the backup host, the host that created the backup. |
| i | The numerical ID of the cluster. |
| I | The numerical ID of the backup. |
| J | The numerical ID of the job that created the backup. |
| M | The backup method (e.g. "mysqldump"). |
| O | The name of the user who initiated the backup job. |
| S | The name of the storage host, the host that stores the backup files. |
| % | The percent sign itself. Use two percent signs, `%%` the same way the standard `printf()` function interprets it as a one percent sign. |

### Backup List

| Column | Description |
|---|---|
| ID | The numerical ID of the backup. |
| PI | The numerical ID of the parent backup if there is a parent backup for the given entry. |
| CID | The numerical ID of the cluster to which the backup belongs. |
| V | The verification status. Here `V` means the backup is verified, and – which means the backup is not verified. |
| I | The flag showing if the backup is incremental or not. Here F means the backup is a full backup, `I` means the backup is incremental, – means the backup contains no incremental or full backup files (because for example, the backup failed) and B means the backup contains both full and incremental backups files (which is impossible). |
| STAT | The state of the backup. Here "COMPLETED" means the backup is completed, "FAILED" means the backup has failed, and "RUNNING" means the backup is being created. |
| OWN | The name of the Cmon user that owns the backup. |
| HOST NAM | The name of the host where the backup was created. |
| CRE· ATEI | The date and time when the backup was created. |
| SIZE | The total size of the created backup files. |
| TI- TLE | The name or title of the backup. This is a human-readable string that helps identify the backup. |

### Backup Format

When the option `--backup-format` is used the specified information will be printed instead of the default columns. The format string uses the `%` character to mark variable fields and flag characters as they are specified in the standard `printf()` C library functions. The `%` specifiers are ended by field name letters to refer to various properties of the backups.

The `%+12I` format string for example has the `+12` flag characters in it with the standard meaning: the field will be 12 characters wide and the + or – sign will always be printed with the number. The properties of

the backup are encoded by letters. The`%16H` for example the letter`H` encodes the `hostname`. Standard `\` notation is also available, `\n` for example, encodes a new-line character.

The s9s-tools support the following fields:

| Character | Description |
|---|---|
| B | The date and time when the backup creation was beginning. The format used to print the dates and times can be set using the –date-format. |
| C | The backup file creation date and time. The format used to print the dates and times can be set using the –date-format. |
| d | The names of the databases in a comma-separated string-list. |
| D | The description of the backup. If the c the modifier is used (e.g. `%cD`) the configured description is shown. |
| e | The word “ENCRYPTED” or “UNENCRYPTED” depending on the encryption status of the backup. |
| E | The date and time when the backup creation was ended. The format used to print the dates and times can be set using the –date-format. |
| F | The archive file name. |
| H | The backup host (the host that created the backup). If the c modifier is used (e.g. %cH) the configured backup host is shown. |
| I | The numerical ID of the backup. |
| i | The numerical ID of the cluster to which the backup belongs. |
| J | The numerical ID of the job that created the backup. |
| M | The backup method used. If the c modifier is used the configured backup method will be shown. |
| O | The name of the owner of the backup. |
| P | The full path of the archive file. |
| R | The root directory of the backup. |
| S | The name of the storage host, the host where the backup was stored. |
| s | The size of the backup file measured in bytes. |
| t | The title of the backup. They can be added when the backup is created, it helps to identify the backup later. |
| v | The verification status of the backup. Possible values are “Unverified”, “Verified” and “Failed”. |
| % | The percent sign itself. Use two percent signs, `%%` the same way the standard `printf()` function interprets it as a one percent sign. |

### Examples

- Suppose we have a data node on 10.10.10.20 (port 3306) on cluster-id 2, and we want to backup all databases using mysqldump and store the backup on ClusterControl server:

```
s9s backup --create \
        --backup-method=mysqldump \
        --cluster-id=2 \
        --nodes=10.10.10.20:3306 \
        --on-controller \
        --backup-directory=/storage/backups
```

- Create a mongodump backup on 10.0.0.148 for cluster named ‘MongoDB ReplicaSet 5.0’ and store

the backup on the database node:

```
s9s backup --create \
        --backup-method=mongodump \
        --cluster-name='MongoDB ReplicaSet 5.0' \
        --nodes=10.0.0.148 \
        --backup-directory=/storage/backups
```

- Schedule a full backup using MariaDB backup every midnight at 1:10 AM:

```
s9s backup --create \
        --backup-method=mariabackupfull \
        --nodes=10.10.10.19:3306 \
        --cluster-name=MDB101 \
        --backup-dir=/home/vagrant/backups \
        --on-controller \
        --recurrence='10 1 * * *'
```

- Schedule an incremental backup using MariaDB backup every day at 1:30 AM:

```
s9s backup --create \
        --backup-method=mariabackupincr \
        --nodes=10.10.10.19:3306 \
        --cluster-name=MDB101 \
        --backup-dir=/home/vagrant/backups \
        --on-controller \
        --recurrence='30 1 * * *'
```

- Create a pg_dumpall backup on PostgreSQL master server and store the backup on ClusterControl server:

```
s9s backup --create \
        --backup-method=pgdump \
        --nodes=192.168.0.81:5432 \
        --cluster-id=43 \
        --backup-dir=/home/vagrant/backups  \
        --on-controller \
        --log
```

- Create a PostgreSQL partial backup using pgdump:

```
 s9s backup   --create \
              --cluster-id=1 \
              --backup-method=pgdump \
              --databases="app_db" \
              --schemas="public,app_data" \
              --backup-dir=/tmp
```

- Create a PostgreSQL partial backup with schema exclusions using pgdump:

```
 s9s backup   --create \
              --cluster-id=1 \
```

(continues on next page)

(continued from previous page)

```
              --backup-method=pgdump \
              --databases="app_db" \
              --exclude-schemas="temp,audit" \
              --backup-dir=/var/backups
```

- Create a PostgreSQL partial structure-only backup with removed ownership using pgdump:

```
s9s backup   --create \
             --cluster-id=1 \
             --backup-method=pgdump \
             --databases="app_db" \
             --schema-only \
             --no-owner \
             --no-privileges \
             --backup-format=custom \
             --backup-dir=/var/schema-backups
```

- List all backups for cluster ID 2:

```
s9s backup --list \
       --cluster-id=2 \
       --long \
       --human-readable
```

**Tip**

Omit the`--cluster-id=2` to see the backup records for all clusters.

- Restore backup ID 3 on cluster ID 2:

```
s9s backup --restore \
       --cluster-id=2 \
       --backup-id=3 \
       --wait
```

???+ Note

```
  If the backup is encrypted, it will be automatically decrypted when␣
↪restoring.
```

- Create a job to verify the given backup identified by the backup ID. The job will attempt to install MySQL on the test server using the same settings as for the given cluster, then restore the backup on this test server. The job returns OK only if the backup is successfully restored on the test server:

```
s9s backup --verify \
       --log \
       --backup-id=1 \
       --test-server=192.168.0.55 \
       --cluster-id=1
```

- Delete the local backup copy after uploading it to the cloud for cluster ID 1:

```
s9s backup \
        --create \
        --cluster-id=1 \
        --nodes=192.168.0.209 \
        --delete-after-upload \
        --cloud-provider="aws" \
        --s3-bucket="my-aws-bucket-eu-west-1" \
        --credential-id=2 \
        --backup-method=xtrabackupfull \
        --wait
```

- Delete old backups for cluster ID 1 that are longer than 7 days, but do not delete at least 3 of the latest backups:

```
s9s backup --delete-old \
        --cluster-id=1 \
        --backup-retention=7 \
        --safety-copies=3 \
        --log
```

- Delete a backup schedule with job ID 1:

```
s9s backup \
        --delete-schedules \
        --job-id=1
```

- Perform a point-in-time restoration at a particular date and time for backup ID 10 on cluster ID 1:

```
s9s backup \
        --restore \
        --cluster-id=1 \
        --backup-id=10 \
        --pitr-stop-time="2020-07-14 14:27:04" \
        --log
```

- Perform a PostgreSQL restoration without applying WAL backup ID 10 on cluster ID 1:

```
s9s backup \
        --restore \
        --cluster-id=1 \
        --backup-id=10 \
        --psql-immediate \
        --log
```

- Create a snapshot repository on an Elasticsearch cluster with cluster ID 1:

```
s9s backup \
      --create-snapshot-repository \
      --cluster-id=1 \
      --snapshot-repo-type=s3 \
```

(continues on next page)

(continued from previous page)

```
      --snapshot-repository=mySnapshotRepository \
      --credential-id=1 \
      --s3-bucket=elastic-s3-test \
      --s3-region=eu-west-3 \
      --wait
```

- List a snapshot repository on a elasticsearch cluster with cluster ID 1:

```
s9s backup \
      --list-snapshot-repository \
      --cluster-id=1 \
```

- Delete a snapshot repository on a elasticsearch cluster with cluster ID 1:

```
s9s backup \
      --delete-snapshot-repository \
      --cluster-id=1 \
      --snapshot-repository=mySnapshotRepository \
      --wait
```

- Generate a backup streaming it directly to the cloud with cluster ID 1:

```
s9s backup \
    --create \
    --cluster-id=1 \
    --nodes=192.168.0.209 \
    --cloud-only \
    --cloud-provider="aws" \
    --s3-bucket="my-aws-bucket-eu-west-1" \
    --credential-id=2 \
    --backup-method=xtrabackupfull \
    --wait
```

### 4.5.4 s9s-cluster

Create, manage, and manipulate clusters.

#### Usage

```
s9s cluster {command} {options}
```

#### Commands

| Name, shorthand | Description |
|---|---|
| `--add-node` | Adds a new node (server) to the cluster or to be more precise creates a new job that will eventually add a new node to the cluster. The name (or IP address) of the node should be specified using the `--nodes` command-line option. See *Node List.* |
| `--available-upgrades` | Shows the available packages to upgrade the cluster with. |
| `--change-config` | Changes the configuration values for the cluster. The cluster configuration in this context is the Cmon Controller's configuration for the given cluster. |
| `--check-hosts` | Checks the hosts before installing a cluster. |
| `--check-pkg-upgrades` | Checks available cluster packages upgrades. |
| `--collect-logs` | Creates a job that will collect the log files from the nodes of the cluster. |
| `--create` | Creates a new cluster. When this command-line option is provided the program will contact the controller and register a new job that will eventually create a new cluster. |
| `--create-account` | Creates a new account to be used on the cluster to access the database(s). |
| `--create-database` | Creates a database on the cluster. |
| `--create-report` | When this command-line option is provided a new job will be started that will create a report. After the job is executed the report will be available on the controller. If the –output-dir the command-line option is provided the report will be created in the given directory on the controller host. |
| `--delete-account` | Deletes an existing account from the cluster. |
| `--delete-database` | Creates a new job that will delete a database from the cluster. |
| `--deploy-agents` | Starts a job to deploy agent-based monitoring using Prometheus and exporters to the nodes. |
| `--deploy-cmonagents` | Starts a job to deploy Cmon Agents (TopQuery Monitoring) to the nodes. |
| `--disable-recovery` | Creates a new job that will disable the auto-recovery for the cluster (both cluster auto-recovery and node auto-recovery). The job can optionally be used to also register a maintenance period for the cluster. |
| `--disable-ssl` | Disable SSL connections on the nodes. |
| `--drop` | Drops cluster from the controller. |
| `--enable-recovery` | Creates a job that will enable auto-recovery for both the cluster and the nodes in the cluster. |
| `--enable-ssl` | Enable SSL connections on the nodes. |
| `--import-config` | Creates a job that will import all the configuration files from the nodes of the cluster. |

continues on next page

Table 4.6 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--import-sql-users` | Imports SQL users to the load balancer. Depending on the actual load balancer, this can be only import or complete update of the user authentication information known by the load balancer. This is only supported by PgBouncer at the moment. Adds a new node (server) to the cluster, or to be more precise creates a new job that will eventually add a new node to the cluster. The load balancer nodes where the users are to be imported shall be specified using the `--nodes` command line option. See *Node List.* |
| `--list, -L` | Lists the clusters. |
| `--list-config` | This command-line option can be used to print the configuration values for the cluster. The cluster configuration in this context is the Cmon Controller's configuration for the given cluster. |
| `--list-databases` | Lists the databases found on the cluster. Please note that if the cluster has a lot of databases, this option might not show some of them. Sampling a huge number of databases would generate a high load and so the controller has an upper limit built into it. |
| `--ping` | Checks the connection to the controller. |
| `--preserve-configuration` | When performing a major version upgrade, this option controls whether configuration settings are migrated from the old PostgreSQL instance to the new one. This includes performance, memory, and security parameters. The default value is false, allowing new PostgreSQL versions to use their version-specific default settings unless explicitly overridden. |
| `--promote-slave` | Promotes a slave node to become a master. This main option will, of course, work only on clusters where it is meaningful, where there are slaves and masters are possible. |
| `--reconfigure-node` | Reconfigures specified nodes of the cluster. |
| `--reinstall-node` | Reinstalls software and also reconfigures it on nodes. |
| `--register` | Registers an existing cluster in the controller. This option is very similar to the –create option, but it, of course, will not install a new cluster; it just registers one. |
| `--remove-node` | Removes a node from the cluster (creates a new job that will remove the node from the cluster). The name (or IP address) of the node should be specified using the –nodes command line option. See *Node List.* |

continues on next page

Table 4.6 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--rolling-restart` | Restarts the nodes (one node at a time) without stopping the cluster. |
| `--set-read-only` | Creates a job that, when executed, will set the entire cluster into read-only mode. Please note that not every cluster type supports the read-only mode. |
| `--setup-audit-logging` | Creates a job that will set up audit logging on the cluster nodes. This option is used to configure audit logging for database clusters, particularly PostgreSQL clusters. Additional audit event data can be specified using the –audit-log-events-data option. |
| `--start` | Creates a new job to start the cluster. |
| `--stat` | Prints the details of one or more clusters. |
| `--stop` | Creates and registers a new job that will stop the cluster when executed. |
| `--sync` | Synchronize cluster list with UI Frontend. |
| `--upgrade-cluster` | Upgrades cluster packages while keeping the same major version. |
| `--upgrade-to-version` | The new newer major version to upgrade to. Without this, only a minor upgrade will be done. |
| `--upgrade-method` | Strategy for doing a major upgrade. For PostgreSQL, there are two methods supported: copy and link. The default value is the copy method. The copy method will copy all data by doing a backup of the old version and restoring it to the new version. With the link method, the data files won't be copied, instead, hard links will be created to the old version's data files. |
| `--uninstall-cmonagents` | Starts a job to remove Cmon Agents (TopQuery Monitoring) from the nodes. |

### Options

| Name, shorthand | Description |
|---|---|
| `--access-check-cmd=NAME` | The command that will be called on hosts to confirm ability to execute commands on the node. The command specified must return 0 on successful execution. |
| `--add-publication` | Creates a new PostgreSQL publication for logical replication cluster. |
| `--add-subscription` | Creates a new PostgreSQL subscription for logical replication. |

continues on next page

Table 4.7 – continued from previous page

| Name, shorthand | Description |
|---|---|
| --all-pool | Include all cluster pool information when listing clusters. This option is used with the –list operation to retrieve extended cluster pool details from all controllers in pool. This option is mutually exclusive with –cluster-id when using the –list operation. |
| --audit-log-evenets-data=STR | Set the audit events data for PostgreSQL audit logging. This option is used in conjunction with the –setup-audit-logging main option to specify which database events should be audited. The value is passed as-is to the audit logging configuration. |
| --backup-id=NUMBER | The ID of a backup to be restored on the newly created cluster. |
| --batch | Print no messages. If the application created a job print only the job ID number and exit. If the command prints data do not use syntax highlight, headers, or totals, only the pure table to be processed using filters. |
| --cluster-format=FORMATSTRING | The string controls the format of the printed information about clusters. See *Cluster Format.* |
| --cluster-id=ID, -i | The ID of the cluster to manipulate. |
| --cluster-name=NAME, -n | Sets the cluster name. If the operation creates a new cluster, this will be the name of the new cluster. |
| --cluster-type=TYPENAME | The type of cluster to install. Currently, the following types are supported: galera, mysqlreplication, groupreplication (or group_replication), ndb (or ndbcluster), mongodb (MongoDB ReplicaSet only), and postgresql. |
| --config-template=FILENAME | Use the specified file as a configuration template to create the configuration file for the new cluster. |
| --create-local-repository | Create a local software (APT/YUM) repository mirror when installing software packages. Using this command line option, it is possible to deploy clusters and add nodes offline, without a working internet connection. |
| --datadir=DIRECTORY | The directory on the node(s) that will hold the data. The primary use for this command-line option is to set the data directory path when a cluster is created. |
| --db-admin=USERNAME | The user name of the database administrator (e.g., 'root'). |
| --db-admin-passwd=PASSWD | The password for the database admin. |
| --donor=ADDRESS | Currently, this option is used when starting a cluster. It can be used to control which node will be started first and used for the others as donors. |
| --drop-publication | Drops an existing PostgreSQL publication. |
| --drop-subscription | Drops an existing PostgreSQL subscription. |

continues on next page

Table 4.7 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--enterprise-token=TOKEN` | The customer's Repo/Download Token for an Enterprise Database. |
| `--extensions= LIST` | For Postgres, a comma-separated list of PostgreSQL extensions (pgvector and PostGIS). |
| `--job-tags=LIST` | Tags for the job if a job is created. |
| `--keep-firewall` | When not specified the CLI will pass the disable firewall option to create cluster and node addition operations. To keep your firewall settings you may pass this option. This option can also be set in the s9s configuration file using the keep_firewall keyword (check s9s.conf(5) for further details). |
| `--list-publications` | Lists all publications in a PostgreSQL Logical Replication cluster. |
| `--list-subscriptions` | Lists all subscriptions in a PostgreSQL Logical Replication cluster. |
| `--local-repository=NAME` | Use a local repository mirror created by ClusterControl for software deployment. |
| `--long`, `-l` | Print the detailed list. |
| `--modify-publication` | Modifies an existing PostgreSQL publication. |
| `--modify-subscription` | Modifies an existing PostgreSQL subscription. The subscription can be enabled or disabled using the –enable and –disable options, respectively. |
| `--no-header` | Do not print headers for tables. |
| `--no-install` | Skip the cluster software installation part. Assume all software is installed on the node(s). This command-line option is considered when installing a new cluster or adding a new node to an existing cluster. |
| `--nodes=NODE_LIST` | List of nodes to work with. See *Node List.* |
| `--os-user=USERNAME` | The name of the remote user that is used to gain SSH access on the remote nodes. If this command-line option is omitted the name of the local user will be used on the remote host too. |
| `--os-elevation=NAME` | he method for authorizing superuser access on the node, allowing the execution of commands with elevated privileges. Supported options include sudo (default) for traditional Unix-like systems, doas for a simpler alternative, and pbrun for privilege escalation in specific environments. |
| `--os-key-file=PATH` | The path of the SSH key to install on a new container to allow the user to log in. This command-line option can be passed when a new container is created, the argument of the option should be the path of the private key stored on the controller. Although the path of the private key file is passed only the public key will be uploaded to the new container. |

continues on next page

Table 4.7 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--output-dir=DIR` | The directory where the files are created. Use in conjunction with–create-report command. |
| `--percona-client-id=CLIENTID` | The client ID for the Percona Pro repository. |
| `--percona-pro-token=TOKEN` | The token for the Percona Pro repository. |
| `--provider-version=VERSION` | The version string of the software to be installed. |
| `--remote-cluster-id=ID` | The remote cluster-ID is used for cluster creation when cluster-to-cluster replication is to be installed. Please note that not all the cluster types support cluster-to-cluster replication. |
| `--semi-sync=[true/false]` | Specifies the semi-sync mode for MySQL Replication. |
| `--use-internal-repos` | Use internal repositories when installing software packages. Using this command-line option it is possible to deploy clusters and add nodes offline, without a working internet connection. The internal repositories have to be set up in advance. |
| `--vendor=VENDOR` | The name of the software vendor to be installed. |
| `--wait` | Waits for the specified job to end. While waiting, a progress bar will be shown unless the silent mode is set. |
| `--with-ssl` | Sets up SSL while creating a new cluster. |
| `--with-database` | Create a new database for the account when creating a new database user account. |
| `--without-ssl` | If this option is provided SSL will not be set up while creating a new cluster. |
| `--with-tags=LIST` | Limit the list of printed clusters by tags. See *Cluster Tagging.* |
| `--without-tags=LIST` | Limit the list of printed clusters by tags. See *Cluster Tagging.* |
| `--with-timescaledb` | Install the TimescaleDB option when creating a new cluster. This is currently only supported on PostgreSQL systems. |
| ACCOUNT, DATABASE & CONFIGURATION MANAGEMENT | |
| `--account=NAME[:PASSWD][@HOST` | Account to be created on the cluster. |
| `--db-name=NAME` | The name of the database. |
| `--opt-group=NAME` | The option group for configuration. |
| `--opt-name=NAME` | The name of the configuration item. |
| `--opt-value=VALUE` | The value for the configuration item. |
| `--with-database` | Create a database for the user too. |
| CONTAINER & CLOUD | |
| `--cloud=PROVIDER` | This option can be used when new container(s) are created. The name of the cloud provider where the new container will be created. This command-line option can also be used to filter the list of the containers when used together with one of the `--list` or `--stat` options. |

continues on next page

Table 4.7 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--containers=LIST` | A list of containers to be created and used by the created job. This command-line option can be used to create containers (virtual machines) and install clusters on them or just add them to an existing cluster as nodes. See *s9s container for details.* |
| `--credential-id=ID` | The cloud credential ID that should be used when creating a new container. This is an optional value, if not provided the controller will find the credential to be used by the cloud name and the chosen region. |
| `--firewalls=LIST` | List of firewall (security groups) IDs separated by, or; to be used for newly created containers. See *s9s-container for further details.* |
| `--generate-key` | Create a new SSH key pair when creating new containers. If this command-line option was provided a new SSH key pair will be created and registered for a new user account to provide SSH access to the new container(s). If the command creates more than one container the same one keypair will be registered for all. The username will be the username of the authenticated cmon-user. This can be overruled by the `--os-user` command line option. When the job creates a new cluster the generated keypair will be registered for the cluster and the file path will be saved into the cluster's Cmon configuration file. When adding a node to such a cluster this `--generate-key` option should not be passed, the controller will automatically re-use the previously created key pair. |
| `--image=NAME` | The name of the image from which the new container will be created. This option is not mandatory, when a new container is created the controller can choose an image if it is needed. To find out what images are supported by the registered container servers please issue the s9s server `--list-images` command. |
| `--image-os-user=NAME` | The name of the initial OS user is defined in the image for the first login. Use this option to create containers based on custom images. |
| `--os-password=PASSWORD` | This command-line option can be passed when creating new containers to set the password for the user that will be created on the container. Please note that some virtualization backends might not support passwords, only keys. |

continues on next page

Table 4.7 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--subnet-id=ID` | This option can be used when new containers are created to set the subnet ID for the container. To find out what subnets are supported by the registered container servers please issue the s9s server `--list-subnets` command. |
| `--template=NAME` | The name of the container template. See *Container Template.* |
| `--volumes=LIST` | When a new container is created this command-line option can be used to pass a list of volumes that will be created for the container. The list can contain one or more volumes separated by the `;` character. Every volume consists of three properties separated by the `:` character, a volume name, the volume size in gigabytes, and a volume type that is either "HDD" or "SSD". The string vol1:5:hdd;vol2:10:hdd for example defines two hard-disk volumes, one 5GByte, and one 10GByte. For convenience the volume name and the type can be omitted, so that automatically generated volume names are used. |
| `--vpc-id=ID` | This option can be used when new containers are created to set the VPC ID for the container. To find out what VPCs are supported by the registered container servers please issue the s9s server `--list-subnets --long` command. |
| LOGICAL REPLICATION | |
| `--subcluster-id=ID` | The PostgreSQL sub-cluster ID where the publication will be created. |
| `--subcluster-name=NAME` | The PostgreSQL sub-cluster name where the publication will be created. |
| `--pub-name=NAME` | The name of the publication to create or manage. |
| `--new-pub-name=NAME` | The new name for the publication when modifying an existing publication. |
| `--sub-name=NAME` | The name of the subscription to create or manage. |
| `--new-sub-name=NAME` | The new name for the subscription when modifying an existing subscription. |
| `--include-all-tables` | Include all tables in the publication. |
| `--db-tables=LIST` | Comma-separated list of tables to include in the publication. Not compatible with –include-all-tables. |
| LOAD BALANCER | |
| `--admin-password=PASSWORD` | The password for the administrator of load balancers. |
| `--admin-user=USERNAME` | The username for the administrator of load balancers. |

continues on next page

Table 4.7 – continued from previous page

| Name, shorthand | Description |
|---|---|
| `--dont-import-accounts` | If this option is provided the database accounts will not be imported after the load balancer is installed and added to the cluster. The accounts can be imported later, but it is not going to be part of the load balancer installation performed by the controller. |
| `--haproxy-config-template=FILENAME` | Configuration template for the HAProxy installation. |
| `--monitor-password=PASSWORD` | The password of the monitoring use of the load balancer. |
| `--monitor-user=USERNAME` | The username of the monitoring use of the load balancer. |
| `--maxscale-mysql-user=USERNAME` | The MySQL username of the Maxscale balancer. |
| `--maxscale-mysql-password=PASSWORD` | The password of the MySQL user of the Maxscale balancer. |
| SSL | |
| `--ssl-ca=PATH` | The SSL CA file path on the controller. |
| `--ssl-cert=PATH` | The SSL certificate file path on the controller. |
| `--ssl-key=PATH` | The SSL key file path on the controller. |
| `--ssl-pass=PASSWD` | The password for an existing CA private key when registering a cluster. |
| `--move-certs-dir=PATH` | The path to the directory where the SSL certificates are stored will be moved on the imported cluster. (Please ommit initial /var/lib/cmon/ca). |

### Cluster List

Using the `--list` and `--long` command-line options a detailed list of the clusters can be printed. Here is an example of such a list:

```
$ s9s cluster --list --long
ID STATE TYPE OWNER GROUP NAME COMMENT
1 STARTED replication pipas users mysqlrep All nodes are operational.
Total: 1
```

The list contains the following fields:

| Field | Description |
|---|---|
| ID | The cluster ID of the given cluster. |
| STATE | A short string describing the state of the cluster. Possible values are `MGMD_NO_CONTACT`, `STARTED`, `NOT_STARTED`, `DEGRADED`, `FAILURE`, `SHUTTING_DOWN`, `RECOVERING`, `STARTING`, `UNKNOWN`, `STOPPED`. |
| TYPE | The type of the cluster. Possible values are `mysqlcluster`, `replication`, `galera`, `group_repl`, `mongodb`, `mysql_single`, `postgresql_single`. |
| OWNE | The user name of the owner of the cluster. |
| GROU | The group owner's name. |
| NAME | The name of the cluster. |
| COMMENT | A short human-readable description of the current state of the cluster. |

### Node List

The list of nodes or hosts is enumerated in a special string using a semicolon as a field separator (e.g. 192.168.1.1;192.168.1.2). The strings in the node list are URLs that can have the following protocols:

| URI | Description |
|---|---|
| `mysql://` | The protocol to install and handle MySQL servers. |
| `ndbd://` | The protocol for MySQL Cluster (NDB) data node servers. |
| `ndb_mgmd://` | The protocol for MySQL Cluster (NDB) management node servers. The mgmd:// notation is also accepted. |
| `haproxy://` | Used to create and manipulate HaProxy servers. |
| `proxysql://` | Use this to install and handle ProxySql servers. |
| `maxscale://` | The protocol to install and handle MaxScale servers. |
| `mongos://` | The protocol to install and handle mongo router servers. |
| `mongocfg://` | The protocol to install and handle mongo config servers |
| `mongodb://` | The protocol to install and handle mongo data servers. |
| `pgbackrest:,` | The protocol to install and handle the PgBackRest backup tool. |
| `pgbouncer://` | The protocol to install and handle PgBouncer servers. |
| `pbmagent://` | The protocol to install and handle PBMagent (Percona Backup for MongoDB agent) servers. |

### Cluster Format

The string controls the format of the printed information about clusters. When this command-line option is used, the specified information will be printed instead of the default columns. The format string uses the `%` character to mark variable fields and flag characters as they are specified in the standard `printf()` C library functions. The `%` specifiers are ended by field name letters to refer to various properties of the clusters.

The `%+12I` format string for example has the `+12` flag characters in it with the standard meaning: the field will be 12 characters wide and the + or − the sign will always be printed with the number. The properties

of the message are encoded by letters. The in the `%-5I` for example the letter `I` encodes the "cluster ID" field, so the numerical ID of the cluster will be substituted. Standard `\` a notation is also available, `\n` for example encodes a new-line character.

The s9s-tools support the following fields:

| Field | Description |
|---|---|
| a | The number of active alarms on the cluster. |
| C | The configuration file for the cluster. |
| c | The total number of CPU cores in the cluster. Please note that this number may be affected by hyper-threading. When a computer has 2 identical CPUs, with four cores each, and uses 2x hyper-threading it will count as 2x4x2 = 16. |
| D | The domain name of the controller of the cluster. This is the string one would get if executed the "domain name" command on the controller host. |
| G | The name of the group owner of the cluster. |
| H | The hostname of the controller of the cluster. This is the string one would get if executed the "hostname" command on the controller host. |
| h | The number of the hosts in the cluster including the controller itself. |
| I | The numerical ID of the cluster. |
| i | The total number of monitored disk devices (partitions) in the cluster. |
| k | The total number of disk bytes found on the monitored devices in the cluster. This is a double-precision floating-point number measured in Terabytes. With the `f` modifier (e.g. `%6.2fk`) this will report the free disk space in TeraBytes. |
| L | The log file of the cluster. |
| M | A human-readable short message that describes the state of the cluster. |
| m | The size of the memory of all the hosts in the cluster added together, measured in GBytes. This value is represented by a double-precision floating-point number, so formatting it with precision (e.g. `%6.2m`) is possible. When used with the `f` modifier (e.g.`%6.2fm`) this reports the free memory, the memory that is available for allocation, used for cache, or used for buffers. |
| N | The name of the cluster. |
| n | The total number of monitored network interfaces in the cluster. |
| O | The name of the owner of the cluster. |
| P | The CDT path of the cluster. |
| S | The state of the cluster. |
| T | The type of the cluster |
| t | The total network traffic (both received and transmitted) measured in MBytes/seconds found in the cluster. |
| V | The vendor and the version of the main software (e.g. the MySQL server) on the node. |
| U | The number of physical CPUs on the host. |
| u | The CPU usage percent found on the cluster. |
| w | The total swap space found in the cluster measured in GigaBytes. With the f modifier (e.g. `%6.2fk`) this reports the free swap space in GigaBytes. |
| % | The `%` character itself. |

### Cluster Tagging

The concept is very similar to the hash-tags used by popular services such as Twitter and Instagram. A cluster can be created with tags, by using `--with-tags` the option:

```
s9s cluster --create \
        --cluster-name="MyMariaDBGaleraCluster" \
        --cluster-type=galera \
        --provider-version="10.5" \
        --vendor=mariadb  \
        --nodes="mysql://10.10.10.10?hostname_internal=2323" \
        --os-user=vagrant \
        --os-key-file=~/.ssh/id_rsa \
        --with-tags="MDB;DC1;PRODUCTION" \
        --log \
        --print-request
```

Multiple values are supported using a semi-colon as a delimiter. The tag values are case-sensitive.

An existing cluster can be tagged with `--add-tag` and `--tag` options under by specifying the CMON tree path, retrievable using the tree command. To retrieve the "tree path", one has to list out the CMON object tree and look for column "NAME", as shown in the following example:

```
$ s9s tree --list --long
MODE        SIZE OWNER   GROUP   NAME
crwxrwx---  -    system  admins  MariaDB Replication 10.3
srwxrwxrwx  -    system  admins  localhost
drwxrwxr--  1, 0 system  admins  groups
urwxr--r--  -    admin   admins  admin
urwxr--r--  -    nobody  admins  nobody
urwxr--r--  -    system  admins  system
```

Then, specify the tree path as `/` + tree name ("MariaDB Replication 10.4"), as shown in the following:

```
s9s tree --add-tag --tag="REPLICATION;PRODUCTION" "/MariaDB Replication␣
↪10.4"
```

To show all clusters having a certain tag:

```
s9s cluster --list --long --with-tags="PRODUCTION"
```

Show all cluster that does not have a certain tag:

```
s9s cluster --list --long --without-tags="PRODUCTION"
```

To filter using multiple tag values:

```
s9s cluster --list --long --with-tags="PRODUCTION;DEV"
```

### Examples

- Create a three-node Percona XtraDB Cluster 5.7 cluster, with OS user vagrant:

```
s9s cluster --create \
        --cluster-type=galera \
        --nodes="10.10.10.10;10.10.10.11;10.10.10.12" \
        --vendor=percona \
```

(continues on next page)

(continued from previous page)

```
        --provider-version=8.0 \
        --db-admin-passwd='pa$word' \
        --os-user=vagrant \
        --os-key-file=/home/vagrant/.ssh/id_rsa \
        --cluster-name='Percona XtraDB Cluster 5.7'
```

- Create a three-node MongoDB Replica Set 5.0 by MongoDB Inc (formerly 10gen) and use the default `/root/.ssh/id_rsa` as the SSH key, and let the deployment job run in the foreground:

```
s9s cluster --create \
        --cluster-type=mongodb \
        --nodes="10.0.0.148;10.0.0.189;10.0.0.219" \
        --vendor=10gen \
        --provider-version='5.0' \
        --os-user=root \
        --db-admin='admin' \
        --db-admin-passwd='MyS3cr3tPass' \
        --cluster-name='MongoDB ReplicaSet 5.0' \
        --wait
```

- Create a MongoDB Sharded Cluster with 3 mongos, 3 mongo config, and one shard consists of a three-node replica set called 'replset2':

```
s9s cluster --create \
        --cluster-type=mongodb \
        --vendor=10gen \
        --provider-version=5.0 \
        --db-admin=adminuser \
        --db-admin-passwd=adminpwd \
        --os-user=root \
        --os-key-file=/root/.ssh/id_rsa \
        --nodes=
↪"mongos://192.168.1.11;mongos://192.168.1.12;mongos://192.168.1.12;mongocfg://192.
```

- Import and existing Percona XtraDB Cluster 8.0 and let the deployment job running in the foreground (provided passwordless SSH from ClusterControl node to all database nodes have been set up correctly):

```
s9s cluster --register \
        --cluster-type=galera \
        --nodes="192.168.100.34;192.168.100.35;192.168.100.36" \
        --vendor=percona \
        --provider-version=8.0 \
        --db-admin="root" \
        --db-admin-passwd="root123" \
        --os-user=root \
        --os-key-file=/root/.ssh/id_rsa \
        --cluster-name="My DB Cluster" \
        --wait
```

- Create a MySQL 8.0 replication cluster by Oracle with multiple master and slaves (note the `?` sign

to identify the node's role in the `--nodes` parameter):

```
s9s cluster --create \
        --cluster-type=mysqlreplication \
        --nodes=
↪"192.168.1.117?master;192.168.1.113?slave;192.168.1.115?slave;192.168.1.116?master
↪\
        --vendor=oracle \
        --db-admin="root" \
        --db-admin-passwd="root123" \
        --cluster-name=ft_replication_23986 \
        --provider-version=8.0 \
        --log
```

- Create a Redis Cluster v7 with 6 Redis nodes:

```
s9s cluster --create \
        --cluster-type=redis-sharded \
        --redis-port=6379
        --redis-bus-port=16579 \
        --vendor=redis \
        --provider-version=7 \
        --node-timeout-ms=5000 \
        --replica-validity-factor=10 \
        --db-admin=s9s-user \
        --db-admin-pass=s9ss9s \
        --nodes=
↪"redis-primary://rlc2:6479;redis-replica://rlc3:6479;redis-primary://rlc4:6479;redi
↪\
        --os-user=root \
        --os-key-file=/root/.ssh/id_rsa \
        --cluster-name="My Redis Cluster v7" \
        --log \
        --print-request
```

- Create a Redis Sentinel v6 with one master and two slaves:

```
s9s cluster --create \
        --cluster-type=redis \
        --nodes=
↪"redis://rlc2;redis://rlc3;redis://rlc4;redis-sentinel://rlc2;redis-sentinel://rlc
↪\
        --os-user=root \
        --os-key-file=/root/.ssh/id_rsa \
        --vendor=redis \
        --provider-version=6  \
        --log \
        --print-request \
        --cluster-name="My Redis Sentinel v6"
```

- Import and existing Redis Cluster and let the deployment job run in the foreground (provided passwordless SSH from ClusterControl node to all database nodes have been set up correctly):

```
s9s cluster --register \
        --cluster-type=redis-sharded \
        --redis-port=6379 \
        --db-admin=s9s-user \
        --db-admin-passwd=s9ss9s \
        --os-user=root \
        --os-key-file=/root/.ssh/id_rsa \
        --nodes="redis-primary://rlc3:6479" \
        --vendor=redis \
        --log \
        --cluster-name="My Redis Cluster" \
        --print-request \
        --wait
```

- Create a PostgreSQL 12 streaming replication cluster with one master and two slaves (note the `?` sign to identify the node's role in the `--nodes` parameter):

```
s9s cluster --create \
        --cluster-type=postgresql \
        --nodes=
↪"192.168.1.81?master;192.168.1.82?slave;192.168.1.83?slave;" \
        --db-admin="postgres" \
        --db-admin-passwd="mySuperStongP455w0rd" \
        --cluster-name=ft_replication_23986 \
        --os-user=vagrant \
        --os-key-file=/home/vagrant/.ssh/id_rsa \
        --provider-version=12 \
        --log
```

- Create a new PostgreSQL publication for logical replication cluster on Cluster ID 1:

```
s9s cluster   --add-publication \
              --cluster-id=1 \
              --subcluster-id=2 \
              --pub-name=my_publication \
              --db-name=mydb \
              --db-tables="orders,customers,products"
```

- Creates a new PostgreSQL subscription for logical replication on Cluster ID 1:

```
s9s cluster   --add-subscription \
              --cluster-id=1 \
              --subcluster-id=3 \
              --sub-name=my_subscription \
              --sub-db-name=dest_db \
              --pub-name=source_pub \
              --pub-db-name=source_db \
              --origin=none \
              --copy-data=false
```

- Modify an existing PostgreSQL publication for logical replication on Cluster ID 1:

```
s9s cluster   --modify-publication \
              --cluster-id=1 \
              --subcluster-name=source_db \
              --pub-name=sales_pub \
              --db-name=sales_db \
              --include-all-tables
```

- Modify an existing PostgreSQL subscription for logical replication on Cluster ID 1:

```
s9s cluster   --modify-subscription \
              --cluster-id=1 \
              --subcluster-id=3 \
              --sub-name=old_sub_name \
              --new-sub-name=new_sub_name \
              --pub-name=new_pub_name \
              --db-name=mydb \
              --enable
```

- Drop an existing PostgreSQL publication for logical replication on Cluster ID 1:

```
s9s cluster   --drop-publication \
              --cluster-id=1 \
              --subcluster-id=2 \
              --pub-name=my_publication \
              --db-name=mydb
```

- Drop an existing PostgreSQL subscription for logical replication on Cluster ID 1:

```
s9s cluster   --drop-subscription \
              --cluster-id=1 \
              --subcluster-id=3 \
              --sub-name=my_subscription \
              --db-name=mydb
```

- List all clusters with more details:

```
s9s cluster --list --long
```

- Delete a cluster with cluster ID 1:

```
s9s cluster --delete --cluster-id=1
```

- Add a new database node on Cluster ID 1:

```
s9s cluster --add-node \
        --nodes=10.10.10.14 \
        --cluster-id=1 \
        --wait
```

- Delete a database from the cluster name galera_001.

```
s9s cluster \
        --delete-database \
        --print-request \
        --cluster-name="galera_001" \
        --db-name="my_database" \
        --log
```

- Add a data node to an existing MongoDB Sharded Cluster with cluster ID 12 having replica set name 'replset2':

```
s9s cluster --add-node \
        --cluster-id=12 \
        --nodes="mongodb://192.168.1.20?rs=replset2"
```

- Create an HAProxy load balancer, 192.168.55.198 on cluster ID 1:

```
s9s cluster --add-node \
        --cluster-id=1 \
        --nodes="haproxy://192.168.55.198" \
        --wait
```

- Remove a database node from cluster ID 1 as a background job:

```
s9s cluster --remove-node \
        --nodes=10.10.10.13 \
        --cluster-id=1
```

- Check if the hosts are part of other cluster and accessible from ClusterControl:

```
s9s cluster --check-hosts \
        --nodes="10.0.0.148;10.0.0.189;10.0.0.219"
```

- Schedule a rolling restart of the cluster 20 minutes from now:

```
s9s cluster --rolling-restart \
        --cluster-id=1 \
        --schedule="$(date -d 'now + 20 min')"
```

- Create a database on the cluster with the given name:

```
s9s cluster --create-database \
        --cluster-id=2 \
        --db-name=my_shopping_db
```

- Create a database account on the cluster and also create a new database to be used by the new user. Grant all access to the new database for the new user:

```
s9s cluster --create-account \
        --cluster-id=1 \
        --account="john:myPassword135" \
        --with-database
```

- Create a cluster and tag it using the `--with-tags` option:

```
s9s cluster --create \
        --cluster-name="TAGGED_MDB" \
        --cluster-type=galera \
        --provider-version="10.4" \
        --vendor=mariadb \
        --nodes="mysql://10.10.10.14:3306" \
        --os-user=vagrant \
        --with-tags="MDB;DC1;PRODUCTION" \
        --log
```

- List all databases for every cluster:

```
$ s9s cluster --list-databases --long
SIZE       #TBL #ROWS   OWNER  GROUP  CLUSTER              DATABASE
         0    0       - system admins MySQL Rep Oracle 8.0 sys
 381681664    7  690984 system admins MySQL Rep Oracle 8.0 db5
 381681664    7  690984 system admins MySQL Rep Oracle 8.0 db4
 599785472   11 1083295 system admins MySQL Rep Oracle 8.0 db3
 272629760    5  493560 system admins MySQL Rep Oracle 8.0 db2
 381681664    7  690984 system admins MySQL Rep Oracle 8.0 db1
   7340032    2       0 system admins PostgreSQL 13        postgres
   7340032    0       0 system admins PostgreSQL 13        template1
   7340032    0       0 system admins PostgreSQL 13        template0
   7340032    0       0 system admins PostgreSQL 13        db1
   7340032    0       0 system admins PostgreSQL 13        db2
Total: 11 databases, 2059403264, 39 tables.
```

- Show all clusters having a certain tag (an existing cluster can be tagged using s9s tree), with multiple tags separated by a semi-colon:

```
s9s cluster --list \
        --long \
        --with-tags="PRODUCTION;big_cluster"
```

- List out CMON configuration options for cluster ID 2:

```
s9s cluster --list-config --cluster-id=2
```

- Change a CMON configuration option called `diskspace_warning` for cluster ID 2 (the configuration change will be applied on CMON runtime and configuration file, `/etc/cmon.d/cmon_2.cnf`):

```
s9s cluster --change-config \
        --cluster-id=2 \
        --opt-name=diskspace_warning \
        --opt-value=70
```

- Upgrade cluster packages while keeping the major version for cluster ID 1.

```
s9s cluster \
        --upgrade-cluster \
```

(continues on next page)

(continued from previous page)

```
        --cluster-id=1 \
        --nodes="192.168.0.84:5433;192.168.0.85" \
        --wait
```

- Upgrade cluster packages to a newer major version for cluster ID 1.

```
s9s cluster \
        --upgrade-cluster \
        --upgrade-to-version=12 \
        --cluster-id=1 \
        -- log
```

- Upgrade cluster packages to a newer major version for cluster ID 1 with a specific upgrade method.

```
s9s cluster \
        --upgrade-cluster \
        --upgrade-to-version=12 \
        --upgrade-method=link \
        --cluster-id=1 \
        -- log
```

- Preserve configurations when performing a major version upgrade for cluster ID 1:

```
s9s cluster \
             --upgrade-cluster \
             --upgrade-to-version=15 \
             --upgrade-method=link \
             --preserve-configuration \
             --cluster-id=1 \
             --log
```

- Set up audit logging for cluster ID 1 with a specific log events:

```
s9s cluster \
             --setup-audit-logging \
             --audit-log-events-data="SELECT,INSERT,UPDATE,DELETE" \
             --cluster-id=1 \
             --wait
```

- Deploys CMON agents to all nodes in cluster ID 1. (This enables top query monitoring functionality for the cluster nodes.)

```
s9s cluster \
        --deploy-cmonagents \
        --cluster-id=1 \
        --print-request \
        --log \
        --wait
```

- Uninstalls CMON agents from specified nodes in cluster ID 1. (If no node is specified, CMON agents will be uninstalled from all nodes in the cluster.)

```
s9s cluster \
        --uninstall-cmonagents \
        --cluster-id=1 \
        --nodes=10.67.199.164 \
        --print-request \
        --log \
        --wait
```

### 4.5.5 s9s-container

Manage cloud and container virtualization. Multiple technologies (multiple virtualization backends) are supported (e.g. Linux LXC and AWS) providing various levels of virtualization. Throughout this documentation (and in fact in the command line options) s9s uses the word "container" to identify virtualized servers. The actual virtualization backend might use the term "virtual machine" or "Linux container" but s9s provides a high-level generic interface to interact with them, so the generic "container" term is used. So please note, the term "container" does not necessarily mean "Linux container", it means "a server that is running in some kind of virtualized environment".

In order to utilize the s9s command-line tool and the CMON Controller to manage virtualization, a virtualization host (container server) has to be installed first. The installation of such a container environment is documented in the s9s server.

#### Usage

```
s9s container {command} {options}
```

#### Commands

| Name, shorthand | Description |
|---|---|
| --create | Creates and starts a new container or virtual machine. If this option is provided, the controller will create a new job that creates a container. By default the container will also be started, an account will be created, passwordless sudo granted and the controller will wait for the controller to obtain an IP address. |
| --del | Stop and delete the container or virtual machine. |
| --lis | Lists the containers. See *Container List*. |
| --sta | Starts an existing container. |
| --sta | Prints the details of a container. |
| --sto | Stops the container. This will not remove the container by default, but it will stop it. If the container is set to be deleted on stop (temporary) it will be deleted. |

### Options

| Name, shorthand | Description |
|---|---|
| `--log` | Waits until the job is executed. While waiting the job logs will be shown unless the silent mode is set. |
| `--recurrence=` | This option can be used to create recurring jobs, jobs that are repeated over and over again until they are manually deleted. Every time the job is repeated a new job will be instantiated by copying the original recurring job and starting the copy. The options argument is a crontab style string defining the recurrence of the job. See *Crontab*. |
| `--schedule=DA` | The job will not be executed now but it is scheduled to execute later. The DateTime string is sent to the backend, so all the formats are supported by the controller. |
| `--timeout=SE` | Sets the timeout for the created job. If the execution of the job is not done before the timeout counted from the start time of the job expires the job will fail. Some jobs might not support the timeout feature, the controller might ignore this value. |
| `--wait` | Waits until the job is executed. While waiting a progress bar will be shown unless the silent mode is set. |
| `--clou` | This option can be used when a new container(s) created. The name of the cloud provider where the new container will be created. This command-line option can also be used to filter the list of the containers when used together with one of the `--list` or `--stat` options. |
| `--cont` | The string controls the format of the printed information about the containers. See *Container Format*. |
| `--containers=LI` | A list of containers to be created or managed. The containers can be passed as command-line options (suitable for simple commands) or as an optional argument for this command-line option. The `s9s container --stop node01` and the `s9s container --stop --containers=node01` commands for example are equivalent. See *Create Container List*. |
| `--crec` | The cloud credential ID that should be used when creating a new container. This is an optional value, if not provided the controller will find the credential to be used by the cloud name and the chosen region. |
| `--firewalls=` | List of firewall (AKA security groups) IDs separated by `,` or `;` to be used for newly created containers. This is not a mandatory option, if the virtualization server needs a firewall to be set one such a firewall will be automatically created. Containers created in the same job (for example in a create cluster operation) the containers will share the same firewall, so they will be able to communicate.If the container is created so that it will be added to an existing cluster (e.g. in an add node job) the controller will try to find the firewall of the existing nodes and if it exists will re-use the same ID, so that the nodes can reach each other. |
| `--gene` | Create a new SSH keypair when creating new containers. If this command line option was provided a new SSH keypair will be created and registered for a new user account to provide SSH access to the new container(s). If the command creates more than one containers the same one keypair will be registered for all. This command line option is actually useful for the cases when a new cluster is created together with the new containers. |
| `--image=NA` | The name of the image from which the new container will be created. This option is not mandatory, when a new container is created the controller can choose an image if it is needed. To find out what images are supported by the registered container servers please issue the `s9s server --list-images` command. |
| `--imag` | The name of the initial OS user defined in the image for the first login. Use this option to create containers based on custom images. |
| `--os-k` | The path of the SSH key to install on a new container to allow the user to log in. This command-line option can be passed when a new container is created, the argument of the option should be the path of the private key stored on the controller. Although the path of the private key file is passed only the public key will be uploaded to the new container. |
| `--os-p` | This command-line option can be passed when creating new containers to set the password for the user that will be created on the container. Please note that some virtualization backend might not support passwords, only keys. |
| `--os-u` | This option may be used when creating new containers to pass the name of the user that will |

### Container Format

The string controls the format of the printed information about the containers. When this command line option is used the specified information will be printed instead of the default columns. The format string uses the `%` character to mark variable fields and flag characters as they are specified in the standard `printf()` C library functions. The `%` specifiers are ended by field name letters to refer to various properties of the containers.

The `%+12i` format string for example has the `+12` flag characters in it with the standard meaning: the field will be 12 characters wide and the + or – sign will always be printed with the number. The properties of the container are encoded by letters. The in the `%16D` for example the letter `D` encodes the `data directory` field, so the full path of the data directory on the container will be substituted. Standard `\` notation is also available, `\n` for example, encodes a new-line character.

The s9s-tools support the following fields:

| Field | Description |
|---|---|
| A | The IP address of the container. This is by default the public IPv4 address of the container. Containers being deleted/created might not have any IP addresses, then the – string is substituted. |
| a | The private IP address of the container if there is any or the – string. |
| C | The full path of the configuration file that stores the container settings if such a configuration file exists. |
| c | The cloud (sometimes mentioned as 'provider') of the container, for example, AWS or AZ as it is set in the credentials file `/var/lib/cmon/cloud_credentials.json`. |
| F | The name of the first firewall (security group) if the container has such a property set, the string – otherwise. |
| G | The name of the group owner of the node. |
| I | The ID of the container. |
| i | The name of the image that was used to create the container. |
| N | The name (alias) of the container. |
| O | The username of the owner of the container. |
| S | The state of the container as a string. |
| P | The CDT path of the user. |
| p | The name of the parent server, the container server that manages the container. |
| R | The name of the region in which the container is hosted. |
| r | The address range of the subnet the container belongs to in CIDR notation (e.g.`10.0.0.0/24`) |
| T | The type of container (e.g. `cmon-cloud` or `lxc`). |
| t | The name of the template that was used to create a container or the – string if no such template was used. |
| U | The ID of the subnet of the container. |
| V | The ID of the VPC for the container. |
| z | The class name of the container object. |
| % | The `%` character itself. |

### Container List

Using the `--list` and `--long` command-line options a detailed list of the containers can be printed. Here is an example of such a list:

```
$ s9s container --list --long
S TYPE TEMPLATE OWNER GROUP     NAME                  IP ADDRESS    SERVER
```

(continues on next page)

(continued from previous page)

```
- lxc  -        pipas testgroup bestw_controller     -             core1
u lxc  -        pipas testgroup dns1                 192.168.0.2   core1
u lxc  ubuntu   pipas testgroup ft_containers_35698  192.168.0.228 core1
u lxc  -        pipas testgroup mqtt                 192.168.0.5   core1
- lxc  -        pipas testgroup ubuntu               -             core1
u lxc  -        pipas testgroup www                  192.168.0.19  core1
Total: 6 containers, 4 running.
```

The list contains the following fields:

| Field | Description |
|---|---|
| S | The abbreviated status information. This is `u` for a container that is up and running and – otherwise. |
| TYPE | Shows what kind of container or virtual machine shown in this line, the type of software that provides virtualization. |
| TEMPLATE | The name of the template that is used to create the container. |
| OWNER | The owner of the server object. |
| GROUP | The group owner of the server object. |
| NAME | The name of the container. This is not necessarily the hostname, this is a unique name to identify the container on the host. |
| IP ADDRESS | The IP address of the container or the – character if the container has no IP address. |
| SERVER | The server on which the container can be found. |

### Create Container List

The command-line option argument is one or more containers separated by the `;` character. Each container is an URL defining the container name (an alias for the container) and zero or more properties. The string `container05parent_server=core1;container06?parent_server=core2`, for example, defines two containers one on one server and the other on the other server.

To see what properties are supported in the controller for the containers, one may use the following command:

```
$ s9s metatype --list-properties --type=CmonContainer --long
ST NAME         UNIT DESCRIPTION
r- acl          -    The access control list.
r- alias        -    The name of the container.
r- architecture -    The processor architecture.
```

See Property List for details.

### Container Template

Defining a template is an easy way to set a number of complex properties without actually enumerating them in the command line one by one. The actual interpretation of the template name is up to the virtualization backend that is the protocol of the container server. The lxc backend for example considers the template to be an already created container, it simply creates the new container by copying the template container so the new container inherits everything.

The template name can also be provided as a property name for the container, so the command `s9s container --create --containers="node02?template=ubuntu;node03"--log` for example will create two containers, one using a template, the other using the default settings.

Note that the `--template` the command-line option is not mandatory, if emitted suitable default values will be chosen, but if the template is provided and the template is not found the creation of the new container will fail.

#### Volume List

The list can contain one or more volumes separated by the `;` character. Every volume consists of three properties separated by the `:` character, a volume name, the volume size in gigabytes, and a volume type that is either "HDD" or "SSD". The string `vol1:5:hdd;vol2:10:hdd` for example defines two hard-disk volumes, one 5GByte, and one 10GByte.

For convenience, the volume name and the type can be omitted, so that automatically generated volume names are used.

#### Examples

- Create a container with no special information needed, every setting will use the default values. For this, of course, at least one container server has to be pre-registered and properly working:

```
s9s container --create --wait
```

- Using the default, automatically chosen container names might not be the easiest way, so here is an example that provides a container name:

```
s9s container --create --wait node01
```

- This is equivalent to the following example that provides the container name through a command-line option:

```
s9s container --create --wait --containers="node01"
```

### 4.5.6 s9s-controller

View and handle controller (CMON instance), allows building a highly available cluster of CMON instances to achieve ClusterControl's high availability.

???+ Note

```
CMON HA feature is still a beta feature.
```

This command can help setup CMON with high availability feature using the following simple steps:

1. Install a CMON Controller together with the CMON Database serving as permanent storage for the controller. The CMON HA will not replicate the CMON Database, so it has to be accessible from all the controllers and if necessary it has to provide redundancy on itself.
2. Enable the CMON HA subsystem using the `--enable-cmon-ha` on the running controller. This will create one CmonController class object. Check the object using the `--list` or `--stat` option. The CMON HA is now enabled, but there is no redundancy, only one controller is running. The one existing controller in this stage should be a leader although there are no followers.

3. Install additional Cmon Controllers one by one and start them the usual way. The next controllers should use the same CMON Database and should have the same configuration files. When the additional controllers are started they will find the leader in the CMON Database and will ask the leader to let them join. When the join is successful one more CmonController will be created for every joining controller.

### Usage

```
s9s controller {command} {options}
```

### Commands

| Name short-hand | Description |
|---|---|
| `--cre` | Creates a job that will create a controller to controller snapshot of the Cmon HA subsystem. Creating a snapshot manually using this command-line option is not necessary for the Cmon HA to operate, this command-line option is made for testing and repairing. |
| `--ena` | Enabled the CMON HA subsystem. By default, Cmon HA is not enabled for compatibility reasons, so this command-line option is implemented to enable the controller to controller communication. When the CMON HA is enabled CmonController class objects will be created and used to implement the high availability features (e.g. the leader election). So if the controller at least one CmonController object, the CMON HA is enabled, if not, it is not enabled. |
| `--get` | Gets the LDAP configuration from the controller and prints it to the standard output. |
| `--lis` | Lists the CmonController type objects known to the controller. If the CMON HA is not enabled there will be no such objects, if it is enabled one or more controllers will be listed. With the `--long` option a more detailed list will be shown where the state of the controllers can be checked. |
| `--pin` | Sends a ping request to the controller and prints the information received. Please note that there is another ping request for the clusters, but this ping request is quite different from that. This request does not need a cluster, it is never redirected (follower controllers will also reply to this request) and it is replied with some basic information about the Cmon HA subsystem. |
| `--sta` | Prints more details about the controller objects. |

### Examples

- Create a controller to controller snapshot of CMON HA subsystem:

```
s9s controller \
    --create-snapshot \
    --log
```

- Enable CMON HA feature for the current CMON instance:

```
s9s controller --enable-cmon-ha
```

- The command line option itself is only a syntactic sugar to save a “true” boolean value into a CDT entry, so for example the following command can also be used:

```
echo "true" | \
s9s tree \
--save \
--cmon-user=system \
--password=secret \
--controller="https://10.10.1.23:9501" \
--batch \
/.runtime/cmon_ha/enabled
```

- Print out LDAP configuration from the controller:

```
s9s controller \
        --cmon-user="system" \
        --password="XXXXXXXX" \
        --get-ldap-config \
        --print-json
```

- List out all controllers participate in the CMON HA cluster:

```
s9s controller \
        --list \
        --long
```

- Send ping request to the controller and print the output in JSON format, with some text filtering on the output:

```
s9s controller \
        --ping \
        --print-json \
        --json-format='status: ${controller_status}\n'
```

- Print more details about the controller objects:

```
s9s controller --stat
```

### 4.5.7 s9s-job

View and manage jobs.

#### Usage

```
s9s job {command} {options}
```

### Commands

| Name, shorthand | Description |
|---|---|
| --clo | Creates a copy of the job to re-run it. The clone will have all the properties the original job had and will be executed the same way as new jobs are executed. If the --cluster-id command-line option is used the given cluster will execute the job, if not, the clone will have the same cluster the original job had. |
| --del | Deletes the job referenced by the job ID. |
| --dis able | This command line option can be used to disable (pause) a scheduled or recurring job instance. |
| --en- able | This command line option can be used to enable a disabled (scheduled or recurring) job instance. |
| --fai | Creates a job that does nothing and fails. |
| --kil | This command-line option can be used to send a signal to a running Cmon Job in order to abort its execution. The job subsystem is not preemptive in the controller, so the job only will be actually aborted if and when the job supports aborting what it is doing. |
| --lis | Lists the jobs. See *Job List*. |
| --log | Prints the job messages of the specified job. |
| --suc cess | Creates a job that does nothing and succeeds. |
| --wai | Waits for the specified job to end. While waiting, a progress bar will be shown unless the silent mode is set. |

### Options

| Name, shorthand | Description |
|---|---|
| *NEWLY CREATED JOB* | |
| `--f/ --follow` | Combination of `--log` and `--wait`, s9s is going to attach to an existing running job and print out its job messages while it is running. |
| `--job-tags` | List one of more strings separated by either `,` or `;` to be added as tags to a newly created job if a job is indeed created. |
| `--log` | Waits for the specified job to end. While waiting, the job logs will be shown unless the silent mode is set. |
| `--recurrence=CRON` | Creates recurring jobs, jobs that are repeated over and over again until they are manually deleted. See *Crontab*. |
| `--schedule=DATETI` | The job will not be executed now but it is scheduled to execute later. The `date-time` string is sent to the backend, so all the formats are supported that is supported by the controller. |
| `--timeout=SECOND` | Sets the timeout for the created job. If the execution of the job is not done before the timeout counted from the start time of the job expires the job will fail. Some jobs might not support the timeout feature, and the controller might ignore this value. |
| `--wait` | Waits for the specified job to end. While waiting, a progress bar will be shown unless the silent mode is set. |
| *JOB RELATED OPTIONS* | |
| `--job-id=I` | The job ID of the job to handle or view. |
| `--from=DAT` | Controls the start time of the period that will be printed in the job list. |
| `--limit=NU` | Limits the number of jobs printed. |
| `--offset=NUMBER` | Controls the relative index of the first item printed. |
| `--show-abo` | Turn on the job state filtering and show jobs that are in an aborted state. This command-line option can be used while printing job lists together with the other `--show-*` options. |
| `--show-def` | Turn on the job state filtering and show jobs that are in a defined state. This command-line option can be used while printing job lists together with the other `--show-*` options. |
| `--show-fai` | Turn on the job state filtering and show jobs that are failed. This command-line option can be used while printing job lists together with the other `--show-*` options. |
| `--show-fin` | Turn on the job state filtering and show jobs that are finished. This command-line option can be used while printing job lists together with the other `--show-*` options. |
| `--show-run` | Turn on the job state filtering and show jobs that are running. This command-line option can be used while printing job lists together with the other `--show-*` options. |
| `--show-sch` | Turn on the job state filtering and show jobs that are scheduled. This command-line option can be used while printing job lists together with the other `--show-*` options. |
| `--until=DATE&T` | Controls the end time of the period that will be printed in the job list. |
| `--log-form` | The string that controls the format of the printed log and job messages. See *Log Format Variables*. |
| `--with-tag` | List of one of more strings separated by either `,` or `;` to be used as a filter when printing information about jobs. When this command-line option is provided only the jobs that have any of the tags will be printed. |
| `--without-` | List of one of more strings separated by either `,` or `;` to be used as a filter when printing information about jobs. When this command-line option is provided the jobs that have any of the tags will not be printed. |

### Crontab

Every time the job is repeated a new job will be instantiated by copying the original recurring job and starting the copy. The option argument is a crontab-style string defining the recurrence of the job.

The crontab string must have exactly five space-separated fields as follows:

| Field | Value |
|---|---|
| minute | 0 – 59 |
| hour | 0 – 23 |
| day of the month | 1 – 31 |
| month | 1 – 12 |
| day of the week | 0 – 7 |

All the fields may be a simple expression or a list of simple expressions separated by a comma (`,`). So to clarify the fields are separated by space can contain subfields separated by a comma.

The simple expression is either a star (`*`) representing "all the possible values", an integer number representing the given minute, hour, day or month (e.g. 5 for the fifth day of the month), or two numbers separated by a dash representing an interval (e.g. 8-16 representing every hour from 8 to 16). The simple expression can also define a "step" value, so, for example, `*/2` might stand for "every other hour" or `8-16/2` might stand for "every other hour between 8 and 16 or `*/2` might say "every other hour".

Please check the crontab man page for more details.

### Job List

Using the `--list` command-line option a detailed list of jobs can be printed (the `--long` option results in even more details). Here is an example of such a list:

```
$ s9s job --list
ID CID STATE    OWNER  GROUP  CREATED             RDY  TITLE
1  0   FINISHED pipas  users  2017-04-25 14:12:31 100% Create MySQL Cluster
2  1   FINISHED system admins            03:00:15 100% Removing Old Backups
Total: 2
```

The list contains the following fields:

| Field | Description |
|---|---|
| ID | The numerical ID of the job. The `--job-id` command-line option can be used to pass such ID numbers. |
| CID | The cluster ID. Most of the jobs are related to one specific cluster so those have a cluster ID in this field. Some of the jobs are not related to any cluster, so they are shown with cluster ID 0. |
| STATE | The state of the job. The possible values are `DEFINED`, `DEQUEUED`, `RUNNING`, `SCHEDULED`, `ABORTED`, `FINISHED` and `FAILED`. |
| OWNE | The user name of the user who owns the job. |
| GROU | The name of the group owner. |
| CREATED | The date and time show when the job was created. The format of this timestamp can be set using the `--date-format` command-line option. |
| RDY | A progress indicator showing how many percent of the job was done. Please note that some jobs have no estimation available and so this value remains 0% for the entire execution time. |
| TITLE | A short, human-readable description of the job. |

### Log Format Variables

The format string uses the `%` character to mark variable fields, flag characters as they are specified in the standard `printf()` C library functions and its own field name letters to refer to the various properties of the messages.

The `%+12I` format string for example has the `+12` flag characters in it with the standard meaning: the field will be 12 characters wide and the + or – sign will always be printed with the number. Standard `\` notation is also available, \n for example encodes a new-line character.

The properties of the message are encoded by letters. The in the `%-5L` for example the letter `L` encodes the "line-number" field, so the number of the source line that produced the message will be substituted. The program supports the following fields:

| Variable | Description |
|---|---|
| B | The basename of the source file that produced the message. |
| C | The creation date and time that mark the exact moment when the message was created. The format of the date&time substituted can be set using the `--date-format` command-line option. |
| F | The name of the source file that created the message. This is similar to the `B` fields, but instead of the base name, the entire file name will be substituted. |
| I | The ID of the message, a numerical ID that can be used as a unique identifier for the message. |
| J | The Job ID. |
| L | The line number in the source file where the message was created. This property is implemented mostly for debugging purposes. |
| M | The message text. |
| S | The severity of the message in text format. This field can be "MESSAGE", "WARNING" or "FAILURE". |
| T | The creation time of the message. This is similar to the C field but shows only hours, minutes and seconds instead of the full date and time. |
| % | The `%` character itself. |

### Examples

- List jobs:

```
$ s9s job --list
10235 RUNNING dba 2017-01-09 10:10:17 2% Create Galera Cluster
10233 FAILED  dba 2017-01-09 10:09:41 0% Create Galera Cluster
```

- The s9s client will send a job that will be executed in the background by cmon. It will print out the job ID, for example: "Job with ID 57 registered." It is then possible to attach to the job to find out the progress:

```
s9s job --wait --job-id=57
```

- View job log messages of job ID 10235:

```
s9s job --log --job-id=10235
```

- Delete the job that has the job ID 42:

```
s9s job --delete --job-id=42
```

- Disable the job that has the job ID 102:

```
s9s job --disable --job-id=102
```

- Enable the job that has the job ID 102:

```
s9s job --enable --job-id=102
```

- Create a job that runs every 5 minutes and does nothing at all. This can be used for testing and demonstrating the recurring jobs without doing any significant or dangerous operations.

```
s9s job --success --recurrence="*/5 * * * *"
```

- Clone job ID 14, run it in as a new job, and see the job messages:

```
s9s job --clone \
  --job-id=14 \
  --log
  --clone
```

- Kill a running job with job ID 20:

```
s9s job --kill \
  --job-id=20
```

## 4.5.8 s9s-log

View logs.

### Usage

```
s9s log {command} {options}
```

### Commands

| Name, shorthand | Description |
|---|---|
| `--list, -L` | Lists all log entries. |

### Options

| Name, shorthand | Description |
|---|---|
| *CLUSTER RELATED OPTIONS* | |
| `--cluster-id=ID` | The ID of the cluster to check. |
| `--cluster-name=NAME` | The NAME of the cluster to check. |
| *JOB RELATED OPTIONS* | |
| `--limit=NUMBER` | Limits the number of log messages printed. |
| `--offset=NUMBER` | Controls the relative index of the first item printed. |

**Warning**

Attention

The following log-related option has been deprecated since version 2.2.0 (Sept 2024):

- `--from=DATE&TIME`
- `--until=DATE&TIME`
- `--message-id=ID`
- `--log-format=FORMATSTRING`
- `--long`

### Examples

- List log messages for Cluster ID 1:

```
s9s log \
    --list \
    --cluster-id=1
```

- List log messages for Cluster ID 1 except the 10 last log records:

```
s9s log \
    --list \
    --cluster-id=1 \
    --offset=10
```

- List only the last 20 log messages for Cluster ID 1:

```
s9s log \
    --list \
    --cluster-id=1 \
    --limit=20
```

- List 20 log messages skipping the latest 20 log records for Cluster ID 1:

```
s9s log \
    --list \
    --cluster-id=1 \
    --limit=20 \
    --offset=20
```

### 4.5.9 s9s-maintenance

View and manipulate maintenance periods.

#### Usage

```
s9s maintenance {command} {options}
```

#### Commands

| Name, short hand | Description |
|---|---|
| --create | Creates a new maintenance period. |
| --current | Prints the active maintenance for a cluster or for a host. Prints nothing if no maintenance period is active. |
| --list | Lists the registered maintenance period. See *Maintenance List*. |
| --delete | Deletes an existing maintenance period. The maintenance periods are identified by their UUID strings. The UUID strings by default, are shown in an abbreviated format. The full-length UUID strings will be shown when the command line option is provided. Deleting a maintenance period is also possible by providing only the first few characters of the UUID when these first characters are unique and enough to identify the maintenance period. |
| --next | Prints information about the next maintenance period for a cluster or host. Prints nothing if no maintenance is registered to be started in the future. |

### Options

| Name, shorthand | Description |
|---|---|
| `--be-gin=DATETI` | A string representation of the date and time when the maintenance period will start. |
| `--cluster-` | The cluster for cluster maintenance. |
| `--end=DATE` | A string representation of the date and time when the maintenance period will end. |
| `--full-uui` | Print the full UUID. |
| `--nodes=NO` | The nodes for the node maintenance. See *Node List*. |
| `--rea-son=STRING` | The reason for the maintenance. |
| `--start=DA` | A string representation of the date and time when the maintenance period will start. This option is deprecated, please use the `--begin` option instead. |
| `--uuid=UUI` | The UUID to identify the maintenance period. |

### Maintenance List

Using the `--list` and `--long` command-line options a detailed list of the registered maintenance periods can be printed:

```
$ s9s maintenance --list --long
ST UUID    OWNER  GROUP  START    END      HOST/CLUSTER  REASON
Ah a7e037a system admins 11:21:24 11:41:24 192.168.1.113 Rolling restart.
Total: 1
```

The list contains the following fields:

| Field | Description |
|---|---|
| ST | The short status information, where at the first character position `A` stands for 'active' and – stands for 'inactive'. In the second character position `h` stands for 'host-related maintenance' and `c` stands for 'cluster-related maintenance'. |
| UUID | The unique string that identifies the maintenance period. Normally only the first few characters of the UUID is shown, but if the `--full-uuid` the command-line option is provided the full-length string will be printed. |
| OWNER | The name of the owner of the given maintenance period. |
| GROUP | The name of the group owner of the maintenance period. |
| START | The date and time when the maintenance period starts. |
| END | The date and time when the maintenance period expires. |
| HOST/C | The name of the cluster or host under maintenance. |
| REA-SON | A short human-readable description showing why maintenance is required. |

### Examples

- Create a maintenance period for PostgreSQL node 10.35.112.21, starting at 05:44:55 AM for one full day (cmon expects UTC time to create a maintenance):

```
s9s maintenance \
      --create \
      --nodes=10.35.112.21:5432 \
      --begin=2024-05-15T05:44:55.000Z \
      --end=2024-05-16T05:44:55.000Z \
      --reason='Upgrading RAM' \
      --batch
```

- Create a new maintenance period for 192.168.1.121 which shall start tomorrow and finish an hour later:

```
s9s maintenance --create \
      --nodes=192.168.1.121 \
      --begin="$(date --date='now + 1 day' --utc '+%Y-%m-%d %H:%M:%S')"
↪" \
      --end="$(date --date='now + 1day + 1 hour' --utc '+%Y-%m-%d %H:
↪%M:%S')" \
      --reason="Upgrading software."
```

- List out all nodes that are under maintenance period:

```
$ s9s maintenance --list --long
ST UUID    OWNER GROUP START    END      HOST/CLUSTER REASON
-h 70346c3 dba   admin 07:42:18 08:42:18 10.0.0.209   Upgrading RAM
Total: 1
```

- Delete a maintenance period for UUID 70346c3:

```
s9s maintenance --delete --uuid=70346c3
```

- Check if there is any ongoing maintenance period for cluster ID 1:

```
s9s maintenance --current --cluster-id=1
```

- Check the next maintenance period scheduled for node 192.168.0.227 for cluster ID 1:

```
s9s maintenance \
      --next \
      --cluster-id=1 \
      --nodes="192.168.0.227"
```

### 4.5.10 s9s-metatype

Lists meta-types supported by the controller.

#### Usage

```
s9s metatype {command} {options}
```

### Commands

| Name, shorthand | Description |
|---|---|
| `--list, -L` | Lists the names of the types the controller supports. See *Property List*. |
| `--list-cluster-` | Lists the cluster types the controller supports. With the `--long` option also lists the vendors and the versions. |
| `--list-properti` | Lists the properties the given metatype has. Use the `--type` command-line option to pass the name of the metatype. |

### Options

| Name, shorthand | Description |
|---|---|
| `--type=TYPENAME` | The name of the type. |

### Property List

Using the `--list-properties` and `--long` command line options a detailed list of the metatype properties can be printed:

```
$ s9s metatype --list-properties --type=CmonUser --long
ST NAME          UNIT DESCRIPTION
rw email_address -    The email address of the user.
rw first_name    -    The first name of the user.
rw groups        -    The list of groups for the user.
rw job_title     -    The job title of the user.
r- last_login    -    The date&time of the last successful login.
```

The list contains the following fields:

| Field | Description |
|---|---|
| ST | The short status information, where at the first character position `r` stands for 'readable' and – shows that the property is not readable by the client program. At the second character position `w` stands for 'writable' and – shows that the property is not writable by the client. |
| NAME | The name of the property. |
| UNIT | The unit in which the given property is measured (e.g. 'byte'). This field shows a single – character if the unit is not applicable. |
| DESCRIPTION | The short human-readable description of the property. |

### Examples

- List the metatypes the controller supports:

```
s9s metatype --list
```

- List a detailed list of the properties the CmonUser type has:

```
s9s metatype \
        --list-properties \
        --type=CmonUser \
        --long
```

- List all cluster types currently managed by this controller:

```
s9s metatype \
        --list-cluster-types \
        --long
```

### 4.5.11 s9s-node

View and handle nodes.

#### Usage

```
s9s node {command} {options}
```

#### Commands

| Name, shorthand | Description |
|---|---|
| --change-co | Changes configuration values for the given node. |
| --enable-bi | Creates a job to enable binary logging on a specific node. Not all clusters support this feature (MySQL does). One needs to enable binary logging in order to set up a cluster to cluster replication. |
| --list, -L | List nodes. Default to all clusters. |
| --list-conf | Lists the configuration values for the given node. |
| --pull-conf | Copy the configuration file(s) from the node to the local computer. Use the --output-dir to control where the files will be created. |
| --push-conf | Copy the configuration file(s) to a node. |
| --register | Registers a node that already is working. |
| --restart | Restarts the node. This means the process that provides the main functionality on the node (e.g. the MySQL daemon on a Galera node) will be stopped then start again. |
| --set | Sets various properties of the specified node/host. |
| --set-confi | Changes configuration values for the given node. |
| --set-read- | Creates a job that sets the node to read-only mode. Please note that not all cluster types support read-only mode. |
| --set-read- | Creates a job that sets the node to read-write mode if it was previously set to read-only mode. Please note that not all cluster types support read-only mode. |
| --start | Starts the node. This means the process that provides the main functionality on the node (e.g. the MySQL daemon on a Galera node) will be started. |
| --stat | Prints detailed node information. It can be used in conjunction with --graph to produce statistical data. See *Graph Options*. |
| --stop | Stops the node. This means the process that provides the main functionality on the node (e.g. the MySQL daemon on a Galera node) will be stopped. |
| --unregis-ter | Unregisters the node from ClusterControl. |

### Options

| Name, shorthand | Description |
|---|---|
| `--cluste` | The ID of the cluster in which the node is. |
| `--cluste` | Name of the cluster to list. |
| `--nodes=` | The nodes to list or manipulate. See *Node List*. |
| `--node-f` | The format string used to print nodes. |
| `--opt-gr` | The configuration option group. |
| `--opt-na` | The name of the configuration option. |
| `--opt-va` | The value of the configuration option. |
| `--output` | The directory where the files are created. |
| `--graph=` | The name of the graph to show. See *Graph Options*. |
| `--begin=TIME` | The start of the graph interval. |
| `--end=TI` | The end of the graph interval. |
| `--force` | Force the execution of potentially dangerous operations like restarting a master node in a MySQL Replication cluster. |
| `--begin=TIME` | The start time of the graph (the X-axis). |
| `--density` | If this option is provided will be a probability density function (or histogram) instead of a timeline. The X-axis shows the measured values (e.g. MByte/s) while the Y-axis shows how many percent of the measurements contain the value. If for example, the CPU usage is between 0% and 1% at 90% of the time the graph will show a 90% bump at the lower end. |
| `--end=TI` | The end of the graph. |
| `--node-f` | The string that controls the format of the printed information about the nodes. See *Node Format*. |
| `--properties=ASS` | One or more assignments specifying property names and values. The assignment operator is the = character (e.g. `--properties='alias="newname"'`), multiple assignments are separated by the semicolon `;`. |
| `--output` | The directory where the output files will be created on the local computer. |

### Graph Options

When providing a valid graph name together with the `--stat` option a graph will be printed with statistical data. Currently, the following graphs are available:

| Option | Description |
|---|---|
| cpughz | The graph will show the CPU clock frequency measured in GHz. |
| cpu-load | Shows the average CPU load of the host computer. |
| cpusys | Percent of time the CPU spent in kernel mode. |
| cpuidle | Percent of time the CPU is idle on the host. |
| cpuiowa | Percent of time the CPU is waiting for IO operations. |
| cputemp | The temperature of the CPU measured in degree Celsius. Please note that to measure the CPU temperature some kernel module might be needed (e.g. it might be necessary to run `sudo modprobe coretemp`). On multiprocessor systems, the graph might show only the first processor. |
| cpu-user | Percent of time the CPU is running user space programs. |
| disk-free | The amount of free disk space measured in GBytes. |
| diskrea speed | Disk read speed measured in MBytes/sec. |
| diskrea write-speed | Disk read and write speed measured in MBytes/sec. |
| diskwri speed | Disk write speed measured in MBytes/sec. |
| disku-ti-liza-tion | The bandwidth utilization for the device in percent. |
| mem-free | The amount of the free memory measure in GBytes. |
| memu-til | The memory utilization of the host measured in percent. |
| neter-rors | The number of receive and transmit errors on the network interface. |
| ne-tre-ceived-speed | Network read speed in MByte/sec. |
| ne-tre-ceiveer rors | The number of packets received with error on the given network interface. |
| net-trans-miter-rors | The number of packets failed to transmit. |
| net-sentspe | Network write speed in MByte/sec. |
| net-speed | Network read and write speed in MByte/sec. |
| sql-com-mands | Shows the number of SQL commands executed measured in 1/s. |
| sql-com-mits | The number of commits measured in 1/s. |

### Node List

Using the `--list` and `--long` command-line options a detailed list of the nodes can be printed. Here is an example of such a list:

```
$ s9s node --list --long '192.168.1.1*'
STAT VERSION CID CLUSTER HOST PORT COMMENT
poM- 9.6.2 1 ft_postgresql_11794 192.168.1.117 8089 Up and running
coC- 1.4.2 1 ft_postgresql_11794 192.168.1.127 9555 Up and running
Total: 3
```

???+ Note

```
The list in the example is created using a filter (that is '192.168.1.1\*'␣
↪in the example). The last line shows a 3 as the total, the number of␣
↪nodes maintained by the controller, but only two of the nodes are printed␣
↪in the list because of the filter.
```

The list contains the following fields:

| Field | Description |
|---|---|
| STAT | **Nodetype**This is the type of node. It can be `c` for controller, `g` for Galera node, `x` for MaxScale node, `k` for Keepalived node, `p` for PostgreSQL, `m` for Mongo, `e` for MemCached, `y` for ProxySql, `h` for HaProxy, `b` for PgBouncer, `B` for PgBackRest, `t` for PBMAgent, `a` for Garbd, `r` for group replication host, `A` for cmon agent, `P` for Prometheus, `s` for generic MySQL nodes, `S` for Redis sentinel, `R` for Redis, `E` for Elasticsearch, and `?` for unknown nodes. **Hoststatus**The status of the node. It can be `o` for online, `l` for off-line, `f` for failed nodes, `r` for nodes performing recovery, `-` for nodes that are shut down and `?` for nodes in an unknown state.**Role** This field shows the role of the node in the cluster. This can be `M` for primary, `S` for replica, `U` for multi (primary and replica), `C` for controller, `V` for backup verification node, `A` for arbiter, `R` for backup repository host, `D` for Elasticsearch data host `c` for Elasticsearch coordinator only host and `-` for everything else.**Maintenance** This field shows if the node is in maintenance mode. The character is `M` for nodes in maintenance mode and `-` for nodes that are not in maintenance mode. |
| VERSION | This field shows the version string of the software that provides the service represented in the given line. |
| CID | The cluster ID of the cluster that holds the node as a member. Every node belongs to exactly one cluster. |
| CLUSTER | The name of the cluster that holds the node as a member. |
| HOST | The hostname of the host. This can be a real DNS hostname, the IP address, or the Cmon alias name of the node depending on the configuration and the command line options. The cluster is usually configured to use IP addresses (the Cmon configuration file contains IP addresses) so this field usually shows IP addresses. |
| PORT | The IP port on which the node accepts requests. The same DNS hostname or IP address can be added multiple times to the same or to multiple clusters, but the host:port pair must be unique. In other words, the same host with the same port can not be added to the same Cmon controller twice. Since the hostname:port pair is unique the nodes are identified by this and every line of the node list is representing a hostname:port node. There is one exception to this rule: the Cmon Controller can manage multiple clusters and so be a part of more than one cluster with the same hostname and port. |
| COMMEN | A short human-readable description that the Cmon Controller sets automatically to describe the host state. A single `-` character is shown if the controller did not set the message. |

### Node Format

The string that controls the format of the printed information about the nodes. When this command-line option is used the specified information will be printed instead of the default columns. The format string uses the `%` character to mark variable fields and flag characters as they are specified in the standard `printf()` C library functions. The `%` specifiers are ended by field name letters to refer to various properties of the nodes.

The `%+12i` format string for example has the `+12` flag characters in it with the standard meaning the field will be 12 character-wide and the `+` or `-` sign will always be printed with the number. The properties of the node are encoded by letters. In the `%16D` for example, the letter `D` encodes the "data directory" field, so the full path of the data directory on the node will be substituted. Standard `\` notation is also available, `\n` for example encodes a new-line character.

The s9s-tools support the following fields:

| Field | Description |
|---|---|
| A | The IP address of the node. |
| a | Maintenance mode flag. If the node is in maintenance mode a letter `M`, otherwise `-`. |
| b | The master host for this slave if it is applicable. |
| C | The configuration file for the most important process on the node (e.g. the configuration file of the MySQL daemon on a Galera node). |
| c | The total number of CPU cores in the host. Please note that this number may be affected by hyper-threading. When a computer has 2 identical CPUs, with four cores each and uses 2x hyper-threading it will count as 2x4x2 = 16. |
| D | The data directory of the node. This is usually the data directory of the SQL server. |
| d | The PID file on the node. |
| G | The name of the group owner of the node. |
| g | The log file on the node. |
| h | The CDT path of the node. |
| I | The numerical ID of the node. |
| i | The total number of monitored disk devices (partitions) in the cluster. |
| k | The total number of disk bytes found on the monitored devices in the node. This is a double-precision floating-point number measured in Terabytes. |
| L | The replay location. This field currently only has valid value in PostgreSQL clusters. |
| l | The received location. This field currently only has valid value in PostgreSQL clusters. |
| M | A message, describing the node's status in a human-readable format. |
| m | The total memory size found in the host, measured in GBytes. This value is represented by a double-precision floating-point number, so formatting it with precision (e.g. `%6.2m`) is possible. When used with the f modifier (e.g. `%6.2fm`) this reports the free memory, the memory that available for allocation, used for cache or used for buffers. |
| N | The name of the node. If the node has an alias that is used, otherwise the name of the node is used. If the node is registered using the IP address, the IP address is the name. |
| n | The total number of monitored network interfaces in the host. |
| O | The user name of the owner of the cluster that holds the node. |
| o | The name and version of the operating system together with the codename. |

continues on next page

Table 4.8 – continued from previous page

| Field | Description |
|---|---|
| P | The port on which the most important service is waiting for requests. |
| p | The PID (process ID) on the node that presents the service (e.g. the PID of the MySQL daemon on a Galera node). |
| R | The role of the node (e.g. “controller”, “master”, “slave” or “none”). |
| r | The work read-only or read-write indicating if the server is in read-only mode or not. |
| S | The status of the host (e.g. CmonHostUnknown, CmonHostOnline, CmonHostOffLine, CmonHostFailed, CmonHostRecovery, CmonHostShutDown). |
| s | The list of slaves of the given host in one string. |
| T | The type of the node, e.g. “controller”, “galera”, “postgres”. |
| t | The total network traffic (both received and transmitted) measured in MBytes/seconds. |
| U | The number of physical CPUs on the host. |
| u | The CPU usage percent found on the host. |
| V | The version string of the most important software (e.g. the version of the PostgreSQL installed on a PostgreSQL node). |
| v | The ID of the container/VM in “CLOUD/ID” format. The – string if no container ID is set for the node. |
| w | The total swap space found in the host measured in GigaBytes. With the f modifier (e.g. `%6.2fw`) this reports the free swap space in GigaBytes. |
| Z | The name of the CPU model. Should the host have multiple CPUs, this will return the model name of the first CPU. |
| % | The `%` character itself. |

## Examples

- List all nodes:

```
$ s9s node --list --long
ST  VERSION                  CID CLUSTER        HOST       PORT COMMENT
go- 10.1.22-MariaDB-1~xenial   1 MariaDB Galera 10.0.0.185 3306 Up and␣
↪running
co- 1.4.1.1856                 1 MariaDB Galera 10.0.0.205 9500 Up and␣
↪running
go- 10.1.22-MariaDB-1~xenial   1 MariaDB Galera 10.0.0.209 3306 Up and␣
↪running
go- 10.1.22-MariaDB-1~xenial   1 MariaDB Galera 10.0.0.82  3306 Up and␣
↪running
```

(continues on next page)

(continued from previous page)

```
Total: 4
```

- Print the configuration for a node:

```
$ s9s node --list-config --nodes=10.0.0.3
...
mysqldump   max_allowed_packet                      512M
mysqldump   user                                    backupuser
mysqldump   password                                nWC6NSm7PnnF8zQ9
xtrabackup  user                                    backupuser
xtrabackup  password                                nWC6NSm7PnnF8zQ9
MYSQLD_SAFE pid-file                                /
 ↪var/lib/mysql/mysql.pid
MYSQLD_SAFE basedir                                 /usr/
Total: 71
```

- The following example shows how a node in a given cluster can be restarted. When this command executed a new job will be created to restart a node. The command-line tool will stop and show the job messages until the job is finished:

```
s9s node \
        --restart \
        --cluster-id=1 \
        --nodes=192.168.1.117 \
        --log
```

- Change a configuration value for a PostgreSQL server:

```
s9s node \
        --change-config \
        --nodes=192.168.1.115 \
        --opt-name=log_line_prefix \
        --opt-value='%m '
```

- Push a configuration option inside `my.cnf` (`max_connections=500`) on node 10.0.0.3:

```
s9s node \
        --change-config \
        --nodes=10.0.0.3 \
        --opt-group=mysqld \
        --opt-name=max_connections \
        --opt-value=500
```

- Import two existing Keepalived nodes (192.168.20.56 and 192.168.20.57) where the virtual IP is 192.168.20.59 into cluster ID 13:

```
# import primary keepalived
s9s node \
        --register \
        --nodes="keepalived://192.168.20.56" \
        --virtual-ip=192.168.20.59 \
```

(continues on next page)

(continued from previous page)

```
        --cluster-id=13 \
        --wait

# import secondary keepalived
s9s node \
        --register \
        --nodes="keepalived://192.168.20.57" \
        --virtual-ip=192.168.20.59 \
        --cluster-id=13 \
        --wait
```

- Listing the Galera hosts. This can be done by filtering the list of nodes by their properties:

```
s9s node \
        --list \
        --long \
        --properties="class_name=CmonGaleraHost"
```

- Create a set of graphs, one for each node shown in the terminal about the load on the hosts. If the terminal is wide enough the graphs will be shown side by side for a compact view:

```
s9s node \
        --stat \
        --cluster-id=1 \
        --begin="08:00" \
        --end="14:00" \
        --graph=load
```

- The density function can also be printed to show what were the typical values for the given statistical data. The following example shows what was the typical values for the user mode CPU usage percent:

```
s9s node \
        --stat \
        --cluster-id=2 \
        --begin=00:00 \
        --end=16:00 \
        --density \
        --graph=cpuuser
```

- The following example shows how a custom list can be created to show some information about the CPU(s) in some specific hosts:

```
s9s node \
        --list \
        --node-format="%N %U CPU %c Cores %6.2u%% %Z\n" 192.168.1.191␣
↪192.168.1.195
192.168.1.191 2 CPU 16 Cores 22.54% Intel(R) Xeon(R) CPU L5520 @␣
↪2.27GHz
192.168.1.195 2 CPU 16 Cores 23.12% Intel(R) Xeon(R) CPU L5520 @␣
↪2.27GHz
```

- The following list shows some information about the memory, the total memory and the memory available for the applications to allocate (including cache and buffer with the free memory):

```
s9s node \
        --list \
        --node-format="%4.2m GBytes %4.2fm GBytes %N\n"
16.00 GBytes 15.53 GBytes 192.168.1.191
47.16 GBytes 38.83 GBytes 192.168.1.127
```

- Set a node to read-write mode if it was previously set to read-only mode:

```
s9s node \
        --set-read-write \
        --cluster-id=1 \
        --nodes=192.168.0.78 \
        --log
```

- Copy configuration file(s) from a PostgreSQL server 192.168.0.232 into the localhost:

```
s9s node \
        --pull-config \
        --nodes="192.168.0.232" \
        --output-dir="tmp"
```

### 4.5.12 s9s-process

View processes running on nodes.

#### Usage

```
s9s process {command} {options}
```

#### Commands

| Name, short-hand | Description |
|---|---|
| `--list` | Lists the processes. |
| `--list` | Prints statement digests together with statistical data showing how long it took them to be executed. The printed list will not contain individual SQL statements but patterns that collect multiple statements of similar form merged into groups by the similarities. |
| `--list` | Lists the queries, internal SQL processes of the cluster. |
| `--top-` | Continuously showing the internal SQL processes in an interactive UI, similar *ClusterControl GUI Top Queries* page. |
| `--top` | Continuously showing the processes in an interactive UI like the well-known `top` utility. Please note that if the terminal program supports the UI can be controlled with the mouse. |

### Options

| Name, shorthand | Description |
|---|---|
| `--cluster-id=ID` | The ID of the cluster to show. |
| `--client=PATTERN` | Shows only the processes that originate from clients that match the given pattern. |
| `--limit=N` | Limits the number of processes shown in the list. |
| `--server=PATTERN` | Shows only the processes that are executed by servers that match the given pattern. |
| `--sort-by-memory` | Sorts the processes by resident memory size instead of CPU usage. |
| `--sort-by-time` | Sorts the SQL queries by their runtime. The longer running queries are going to be on top. |
| `--update-freq=INTE(` | Update frequency for screen refresh in seconds. |

### Examples

- Continuously print aggregated view of processes (similar to `top` output) of all nodes for cluster ID 1:

```
s9s process --top --cluster-id=1
```

- List aggregated view of processes (similar to `ps` output) of all nodes for cluster ID 1:

```
s9s process --list --cluster-id=1
```

- Print out aggregated digested SQL statements on all MySQL nodes in cluster ID 1:

```
s9s process \
        --list-digests \
        --cluster-id=1 \
        --human-readable \
        --limit=10 \
        '*:3306'
```

- Print aggregated list of database top queries which contains the string "INSERT" on all nodes in cluster ID 1 with refresh rate every 1 second:

```
s9s process \
        --top-queries \
        --cluster-id=1 \
        --update-freq=1 \
        'INSERT*'
```

- Print all database queries on cluster ID 1 that is coming from a client with IP address 192.168.0.127 which having only the "INSERT" string:

```
s9s process \
        --list-queries \
        --cluster-id=1 \
        --client='192.168.0.127:*' \
        'INSERT*'
```

- Print all database queries on cluster ID 1 that are reaching the database server 192.168.0.81:

```
s9s process \
        --list-queries \
        --cluster-id=1 \
        --server='192.168.0.81:*'
```

### 4.5.13 s9s-replication

Manage database replication related functions.

???+ Note

```
Only applicable for supported database clusters namely MySQL/MariaDB␣
↪Replication and PostgreSQL Streaming Replication.
```

#### Usage

```
s9s replication {command} {options}
```

#### Commands

| Name, shorthand | Description |
|---|---|
| `--failover` | Takes over the role of master from a failed master. |
| `--list` | List the replication links. |
| `--promote` | Promotes a slave to become a master. |
| `--stage` | Rebuilds or stages a replication slave. |
| `--start` | Starts a replication slave previously stopped using the `--stop` option. |
| `--stop` | Make the slave stop replicating. This option will create a job that does not stop the server but stops the replication on it. |

#### Options

| Name, shorthand | Description |
|---|---|
| `--fail-stop-slave` | When this option is specified slave node will be stopped. |
| `--link-format=FORMA` | The format string controls the format of the printed information about the replication links. See *Link Format*. |
| `--master=NODE` | The replication master. |
| `--remote-cluster-id` | Remote cluster ID for the c2c replication. |
| `--replication-maste` | This is the same as the `--master` option. |
| `--slave=NODE` | The replication slave. |
| `--replication-slave` | This is the same as the `--slave` option. |

### Link Format

The format string controls the format of the printed information about the links. When this command-line option is used the specified information will be printed instead of the default columns. The format string uses the `%` character to mark variable fields and flag characters as they are specified in the standard `printf()` C library functions. The `%` specifiers are ended by field name letters to refer to various properties of the replication.

The `%+12p` format string for example has the `+12` flag characters in it with the standard meaning: the field will be 12 characters wide and the `+` or `-` the sign will always be printed with the number. The properties of the links are encoded by letters. The in the `%4p` for example the letter `p` encodes the `slave port` field, so the port number of the slave node will be substituted. Standard `\` the notation is also available, `\n` for example encodes a new-line character.

The following example prints out a customized format to show the replication link among nodes together with the position of master/slave in the replication log for PostgreSQL streaming replication, MySQL GTID and MariaDB GTID:

```
```
$ s9s replication \
        --list \
        --long \
        --link-format="%16h %4p <- %H %2P %o %O\n"
192.168.0.81 5432 <- 192.168.0.83 5432 0/71001BA8
192.168.0.82 5432 <- 192.168.0.83 5432 0/71001BA8
192.168.0.42 3306 <- 192.168.0.41 3306␣
↪dca7e205-90db-11ea-9143-5254008afee6:1-10 3157
192.168.0.43 3306 <- 192.168.0.41 3306␣
↪dca7e205-90db-11ea-9143-5254008afee6:1-10 3157
192.168.0.92 3306 <- 192.168.0.91 3306 0-54001-49 10285
192.168.0.93 3306 <- 192.168.0.91 3306 0-54001-49 10285
```
```

The s9s-tools support the following fields:

| Field | Description |
|---|---|
| c | The cluster ID of the cluster where the slave node can be found. |
| C | The master cluster-ID property of the slave host. This shows in which cluster the master of the represented replication link can be found. |
| d | This format specifier denotes the “seconds behind the master” property of the slave. |
| h | The hostname of the slave node in the link. |
| H | The hostname of the master node in the link. |
| o | The position of the slave in the replication log. |
| O | The position of the master in the replication log. |
| p | The port number of the slave. |
| P | The port number of the master node. |
| s | A short string representing the link status, e.g. “Online” when everything is ok. |
| m | A slightly longer, human-readable string representing the state of the link. This is actually the `slave_io_state` property of the slave node. |

### Examples

- Print all database replication links for clusters that fall under the replication category:

```
s9s replication --list
```

- Promote a slave 192.168.0.164 to become a new master for a database cluster named "MySQL Rep 5.7 – Production":

```
s9s replication \
        --promote \
        --cluster-name="MySQL Rep 5.7 - Production" \
        --slave="192.168.0.164:3306" \
        --wait
```

- Rebuild the replication slave of 192.168.0.83 from the master, 192.168.0.76, and tag the job as "stage" for cluster ID 1:

```
s9s replication \
        --stage \
        --cluster-id=1 \
        --job-tags="stage" \
        --slave="192.168.0.83:3306" \
        --master="192.168.0.76:3306" \
        --wait
```

- Stop the replication slave on node 192.168.0.80:

```
s9s replication \
        --stop \
        --cluster-id=1 \
        --slave="192.168.0.80:3306" \
        --wait
```

- Start the replication slave on node 192.168.0.80:

```
s9s replication \
        --start \
        --cluster-id=1 \
        --slave="192.168.0.80:3306" \
        --wait
```

## 4.5.14 s9s-report

Manage operational reports.

### Usage

```
s9s report {command} {options}
```

### Commands

| Name, short-hand | Description |
|---|---|
| `--cat` | Prints the report text to the standard output. |
| `--create` | Creates and prints a new report. Please provide the `--type` command-line option with the report type (that was chosen from the template list printed with the `--list-templates` option) and the `--cluster-id` with the cluster ID. |
| `--delet` | Deletes a report that already exists. The report ID should be used to specify which report deleting. |
| `--list` | Lists the reports that already created. |
| `--list-` | Lists the report templates showing what kind of reports can be created. This command-line option is usually used together with the `--long` option. |

### Options

| Name, short-hand | Description |
|---|---|
| `--cluster` | This command-line option passes the numerical ID of the cluster. |
| `--report-` | This command-line option passes the numerical ID of the report to manage. When a report is deleted (`--delete` option) or printed (`--cat` option) this option is mandatory. |
| `--type=NA` | This command-line option controls the type of report. Use the `--list-templates` to list what types are available. |

### Examples

- Lists already created reports for cluster ID 1:

```
s9s report \
        --list \
        --long \
        --cluster-id=1
```

- Print the report ID 1 for cluster ID 1:

```
s9s report \
        --cat \
        --report-id=1 \
        --cluster-id=1
```

- Create a database growth report for cluster ID 1:

```
s9s report \
        --create \
        --type=dbgrowth \
        --cluster-id=1
```

- Delete an operation report with report ID 11:

```
s9s report \
        --delete \
        --report-id=11
```

- Print the supported templates that can be used with `type` value when creating a new report:

```
s9s report \
        --list-templates \
        --long
```

### 4.5.15 s9s-script

Manage and execute Advisor scripts.

#### Usage

```
s9s script {command} {options}
```

#### Commands

| Name, shorthand | Description |
|---|---|
| `--execute` | Executes a script from a local file. |
| `--system` | Executes a shell command or an entire shell script on the nodes of the cluster showing the output of the command as job messages. See *Shell Commands*. |
| `--tree` | Print the names of the scripts stored on the controller in tree format. |

#### Options

| Name, shorthand | Description |
|---|---|
| `--cluster-i` | The target cluster-ID. |
| `--cluster-n` | The target cluster name. |
| `--shell-com` | Executes a shell command with `--system` flag. See *Shell Commands*. |
| `--timeout` | The timeout value when executing the command/script in seconds, default is 10 seconds. If the command/script keeps running, the job will be failed and the execution of the command(s) is aborted. |

#### Shell Commands

If the `--shell-command` option is provided, the options argument will be executed. If a file name is provided as a command-line argument the content of the file will be executed as a shell script. If neither found in the command line the s9s will read its standard input and execute the lines found as a shell script on the nodes (this is the way to implement self-contained remote executed shell scripts using a shebang).

Please note that this will be a job and it can be blocked by other jobs running on the same cluster.

```
s9s script \
        --system \
        --log \
        --cluster-id=1 \
        --shell-command="df -h"
```

The command/script will be executed using the pre-configured OS user of the cluster. If that user has no superuser privileges the Sudo utility can be used in the command or the script to gain superuser privileges.

By default the command/script will be executed on all nodes, all members of the cluster except the controller. This can be changed by providing a node list using the `--nodes` command-line option.

To execute shell commands the authenticated user has to have executed privileges on the nodes. If the executive privilege is not granted the credentials of another Cmon User can be passed in the command line or the privileges can be changed (see *s9s-tree* for details about owners, privileges, and ACLs).

```
s9s script \
        --system \
        --cmon-user=system \
        --password=mysecret \
        --log \
        --cluster-id=1 \
        --timeout=2 \
        --nodes='192.168.0.127;192.168.0.44' \
        --shell-command="df -h"
```

Please note that the job will by default has a 10 seconds timeout, so if the command/script keeps running the job will be failed and the execution of the command(s) aborted. The timeout value can be set using the `--timeout` command-line option.

### Examples

- Print the scripts available for cluster ID 1:

```
s9s script --tree --cluster-id=1
```

- Execute a script called `test_script.sh` on all nodes in the cluster with ID 1:

```
s9s script \
        --system \
        --log \
        --log-format="%M\n" \
        --timeout=20 \
        --cluster-id=1 \
        test_script.sh
```

- Get the disk space summary of host 192.168.0.127 and 192.168.0.44 in cluster ID 1:

```
s9s script \
        --system \
        --log \
        --cluster-id=1 \
```

(continues on next page)

(continued from previous page)

```
        --timeout=2 \
        --nodes='192.168.0.127;192.168.0.44' \
        --shell-command="df -h"
```

### 4.5.16 s9s-server

Provisions and manages virtualization host to be used to host database containers.

???+ Note

```
The supported virtualization platforms are LXC/LXD and Amazon Web Service␣
↪EC2 instances. Docker is not supported at the moment.
```

#### LXC Containers

Handling lxc containers is a new feature added to the CMON Controller and the s9s command-line tool. The basic functionality is available and tested, containers can be created, started, stopped, deleted, and even creating containers on the fly while installing clusters or cluster nodes is possible.

For the lxc containers, one needs a container server, a computer that has the lxc software installed and configured, and of course needs a proper account to access the container server from the CMON Controller. One can set up an lxc container server in two easy and one not so easy steps:

1. Install a Linux server and set it up so that the root user can ssh in from the Cmon Controller with a key, without a password. Creating such access for the superuser is of course not the only way, it is just the easiest.
2. Register the server as a container server on the CMON Controller by issuing the `s9s server --register --servers="lxc://IP_ADDRESS"` command. This will install the necessary software and register the server as a content server to be used later.
3. The hard part is the network configuration on the container server. Most of the distributions by default have a network configuration that provides a local (host only) IP address for the newly created containers. In order to provide a public IP address for the containers, the container server must have some sort of bridging or NAT configured.

A possible way to configure the network for public IP is described in this blog, Converting eth0 to br0 and getting all your LXC or LXD onto your LAN.

#### CMON-Cloud Virtualization

The cmon-cloud containers are an experimental virtualization backend currently added to the CMON Controller as a brand new feature.

#### Usage

```
s9s server {command} {options}
```

### Commands

| Name, shorthand | Description |
|---|---|
| `--add-a` | Adds a new ACL entry to the server or modifies an existing ACL entry. |
| `--create` | Creates a new server. If this option is provided the controller will use SSH to discover the server and install the necessary software packages and modify the configuration if needed so that the server can host containers. |
| `--get-a` | List the ACL of a server. |
| `--list` | List the registered servers. |
| `--list-` | List disks found in one or more servers. |
| `--list-` | List the images available on one or more servers. With the `--long` command-line option a more detailed list is available. |
| `--list-` | List memory modules from one or more servers. |
| `--list-` | List network controllers from one or more servers. |
| `--list-` | List partitions from multiple servers. |
| `--list-` | List processors from one or more servers. |
| `--list-` | Prints the list of regions the server(s) support together with some important information (e.g. if the controller has credentials to use those regions or not). |
| `--list-` | List all the subnets that exist on one or more servers. |
| `--list-` | Lists the supported templates. Various virtualization technologies handle templates differently, some even use different terminology (for example "size" is one such synonym). In the case of LXC, any stopped container can be used as a template for new containers to be created. |
| `--register` | Registers an existing container server. If this command-line option is provided the controller will register the server to be used as a container server later. No software packages are installed or configurations changed. |
| `--start` | Boot up a server. This option will try to start up a server that is physically turned off (using e.g. the wake-on-LAN feature). |
| `--stat` | Prints details about one or more servers. |
| `--stop` | Shuts down and power off a server. When this command-line option is provided the controller will run the shutdown program on the server. |
| `--unregister` | Unregisters a container server, and simply removes it from the controller. |

### Options

| Name, shorthand | Description |
|---|---|
| `--log` | If the s9s application created a job and this command-line option is provided it will wait until the job is executed. While waiting the job logs will be shown unless the silent mode is set. |
| `--recurrence=CR` | This option can be used to create recurring jobs, jobs that are repeated over and over again until they are manually deleted. Every time the job is repeated a new job will be instantiated by copying the original recurring job and starting the copy. The option's argument is a crontab-style string defining the recurrence of the job. See *Crontab*. |
| `--schedule=DATE` | The job will not be executed now but it is scheduled to execute later. The DateTime string is sent to the backend, so all the formats are supported by the controller. |
| `--timeout=SECO` | Sets the timeout for the created job. If the execution of the job is not done before the timeout counted from the start time of the job expires the job will fail. Some jobs might not support the timeout feature, the controller might ignore this value. |
| `--wait` | If the application created a job (e.g. to create a new cluster) and this command-line option is provided the s9s program will wait until the job is executed. While waiting a progress bar will be shown unless the silent mode is set. |
| `--acl=AC` | The ACL entry to set. |
| `--os-key` | The SSH key file to authenticate on the server. If none of the operating system authentication options are provided (`--os-key-file`, `--os-password`, `--os-user`) the controller will try to log in with the default settings. |
| `--os-pas` | The SSH password to authenticate on the server. If none of the operating system authentication options are provided (`--os-key-file`, `--os-password`, `--os-user`) the controller will try to log in with the default settings. |
| `--os-use` | The SSH username to authenticate on the server. If none of the operating system authentication options are provided (`--os-key-file`, `--os-password`, `--os-user`) the controller will try to log in with the default settings. |
| `--refresh` | Do not use cached data, collect information. |
| `--server` | List of servers. |

### Server List

Using the `--list` and `--long` command line options a detailed list of the servers can be printed. Here is an example of such a list:

```
$ s9s server --list --long
PRV VERSION #C OWNER GROUP     NAME         IP            COMMENT
lxc 2.0.8    5 pipas testgroup core1       192.168.0.4   Up and running.
lxc 2.0.8    5 pipas testgroup storage01   192.168.0.17  Up and running.
Total: 2 server(s)
```

The list contains the following fields:

| Field | Description |
|---|---|
| PRV | The name of the provider software. The software that will handle containers or virtual machines on the server. One server can have only one such system, but multiple servers can be registered using one physical computer. |
| VERSION | The version of the provider software. |
| #C | The number of containers/virtual machines currently hosted by the server. |
| OWNE | The owner of the server object. |
| GROU | The group owner of the server object. |
| NAME | The hostname of the server. |
| IP | The IP address of the server. |
| COMMENT | A human-readable description of the server and its state. |

### Examples

- Register a virtualization host:

```
s9s server --register --servers=lxc://storage01
```

- Check the list of virtualization hosts:

```
s9s server --list --long
```

- Create a virtualization server with an operating system username and password to be used to host containers. The controller will try to access the server using the specified credentials:

```
s9s server \
        --create \
        --os-user=testuser \
        --os-password=passw0rd \
        --servers=lxc://192.168.0.250 \
        --log
```

## 4.5.17 s9s-tree

Create, manage and manipulate CMON Directory Tree (CDT).

### Usage

```
s9s tree {command} {options}
```

### Commands

| Name shorthand | Description |
|---|---|
| `--access` | Check the access rights for the authenticated user to the given CDT entry. This main option is made to be used in shell scripts, the exit code of the s9s program shows if the user has the privileges specified in the command line. |
| `--add` | Add an access control list (ACL) entry to an object in the tree. Overwrites the ACL if the given type of ACL already exists. |
| `--add` | This main option can be used to add a new tag to the tag list of an existing object. |
| `--cat` | This option can be used to print the content of a CDT file entry similar way the `cat` standard utility is used to print normal files. |
| `--cho` | Change the ownership of an object. The new owner (and optionally the group owner) should be passed through the `--owner` command-line option. |
| `--del` | Remove CDT entries. |
| `--get` | Print the ACL of a CDT entry. |
| `--lis` | Print the Cmon Directory Tree in list format. |
| `--mkd` | Create a new directory in the tree. |
| `--mov` | Move an object to a new location in the tree or rename the entry. If the target contains the `/` character, it is assumed to be a directory and the source entry will be moved to that directory with its name unchanged. If the target contains no, it is assumed to be a new name and the source entry will be renamed while kept in the same directory. |
| `--rem` | Remove an ACL entry from the ACL of an object. |
| `--rem` | Remove a tag from the tag list of an existing object. |
| `--rmd` | Remove an empty directory from the tree. |
| `--sav` | Saves data (text) into an existing CDT entry that has the proper type (the type is a file). |
| `--tou` | Creates a CDT entry that is a file. |
| `--tre` | Print the tree in its original tree format. |
| `--wat` | Opens an interactive UI to watch and manipulate the CDT filesystem and its entries. |

### Options

| Name, shorthand | Description |
|---|---|
| `--acl=ACLS` | An ACL entry in string format. See *ACL Text Forms*. |
| `--all` | The CDT entries that have a name starting with. considered to be hidden entries. These are only printed if the `--all` command-line option is provided. |
| `--owner=US` | The user name and group name of the owner. |
| `--recursive` | The print also the sub-items of the tree. The `--chown` will change the ownership for sub-items too. Please note that the `--tree` is always recursive, no need for this command-line option there. |
| `--refresh` | Recollect the data. |
| `--tag=TAGS` | Specify one single tag when adding or removing tags of a tag list that belongs to an object. |

### ACL Text Forms

A long and a short text form for representing ACLs is defined. In both forms, ACL entries are represented as three colon-separated fields – an ACL entry tag type, an ACL entry qualifier, and discretionary access permissions. The first field contains one of the following entry tag type keywords:

| Tag type | Description |
|---|---|
| user | A user ACL entry specifies the access granted to either the file owner (entry tag type `ACL_USER_OBJ`) or a specified user (entry tag type `ACL_USER`). |
| group | A group ACL entry specifies the access granted to either the filegroup (entry tag type `ACL_GROUP_OBJ`) or a specified group (entry tag type `ACL_GROUP`). |
| mask | A mask ACL entry specifies the maximum access which can be granted by any ACL entry except the user entry for the file owner and the other entry (entry tag type `ACL_MASK`). |
| other | Another ACL entry specifies the access granted to any process that does not match any user or group ACL entries (entry tag type `ACL_OTHER`). |

The second field contains the user or group identifier of the user or group associated with the ACL entry for entries of entry tag type `ACL_USER` or `ACL_GROUP`, and is empty for all other entries. A user identifier can be a user name or a user ID number in decimal form. A group identifier can be a group name or a group ID number in decimal form.

The third field contains discretionary access permissions. The read, write and search/execute permissions are represented by the `r`, `w`and `x` characters, in this order. Each of these characters is replaced by the `-` character to denote that permission is absent in the ACL entry. When converting from the text form to the internal representation, permissions that are absent need not be specified.

White space is permitted at the beginning and end of each ACL entry, and immediately before and after a field separator (the colon character).

### LONG TEXT FORM

The long text form contains one ACL entry per line. In addition, a number sign (`#`) may start a comment that extends until the end of the line. If an `ACL_USER`, `ACL_GROUP_OBJ` or `ACL_GROUP` ACL entry contains permissions that are not also contained in the `ACL_MASK` entry, the entry is followed by a number sign, the string "effective:", and the effective access permissions defined by that entry. This is an example of the long text form:

```
user::rw-
user:lisa:rw-          #effective:r--
group::r--
group:toolies:rw-      #effective:r--
mask::r--
other::r--
```

### SHORT TEXT FORM

The short text form is a sequence of ACL entries separated by commas and is used for input. Comments are not supported. Entry tag type keywords may either appear in their full unabbreviated form, or in their single letter abbreviated form. The abbreviation for a user is `u`, the abbreviation for a group is `g`, the abbreviation for the mask is `m`, and the abbreviation for others is `o`.

The permissions may contain at most one each of the following characters in any order: `r`, `w`, `x`. These are examples of the short text form:

```
u::rw-,u:lisa:rw-,g::r--,g:toolies:rw-,m::r--,o::r--
g:toolies:rw,u:lisa:rw,u::wr,g::r,o::r,m::r
```

For more information, check out the POSIX Access Control Lists documentation by running `man acl` command.

### Examples

- List out all CDT objects in a tree structure view:

```
s9s tree \
        --tree --all
```

- Add an access control list to allow a group called "users" to have read (access the cluster) and write (make a modification to the cluster) to a cluster named "galera_001":

```
s9s tree \
        --add-acl \
        --acl="group:users:rw-" \
        galera_001
```

- Tag a cluster with a string "Production" for a cluster named `PostgreSQL_Cluster_001` which located at /PostgreSQL Cluster 1 (as a CDT object):

```
s9s tree \
        --add-tag \
        --tag="Production" \
        "/PostgreSQL Cluster 001"
```

- Change the ownership of a cluster called "MariaDB 2 QA" (located at `/MariaDB 2 QA` as a CDT object) to user john and group DBA:

```
s9s tree \
        --chown \
        --owner=john:DBA \
        --recursive \
        "/MariaDB 2 QA"
```

- Create a new directory inside the CDT tree:

```
s9s tree \
        --mkdir \
        /home/kirk
```

- List out all objects under `/MariaDB_Cluster_1` including the hidden objects:

```
s9s tree \
        --list \
        --long \
```

(continues on next page)

(continued from previous page)

```
        --recursive \
        --all \
        --full-path \
        /MariaDB_Cluster_1
```

### 4.5.18 s9s-user

Manage users.

#### Usage

```
s9s user {command} {options}
```

### Commands

| Name, shorthand | Description |
|---|---|
| --add-k | Registers a new public key for an existing user. After the command, the user will be able to authenticate with the private part of the registered public key, no password will be necessary. |
| --add-t | Adds the user to a group. |
| --chang | Modifies the password of an existing user. The password is not a simple property, so it can not be changed using the --set option, this special command-line option has to be used. |
| --create | Registers a new user (create a new user account) on the controller and grants access to the ClusterControl system. The user name of the new account should be the command line argument. |
| --delet | Deletes an existing user. |
| --disable | Disables the user (turn on the "disabled" flag of the user). The users that are disabled are not able to log in. |
| --enable | Enables the user. This will clear the "disabled" flag of the user so that the user will be able to log in again. The "suspended" flag will also be cleared, the failed login counter set to 0 and the date and time of the last failed login gets deleted, so users who are suspended for failed login attempts will also be able to log in. |
| --list | List the users registered for the ClusterControl controller. |
| --list- | List the user groups maintained by the ClusterControl controller. |
| --list- | Lists the public keys registered in the controller for the specified user. Please note that viewing the public keys require special privileges, ordinary users can not view the public keys of other users. |
| --prefe | To add/set/delete the preferences for a given --cmon-user. For example the preferences for UI. Is used with other input options like --preferences-to-set or --preferences-to-delete options. |
| --get-p | To get the preferences for a given --cmon-user. |
| --passv | Resets the password for the user using the "forgot password" email schema. This option must be used twice to change the password, once without a token to send an email about the password reset and once with the token received in the email. |
| --set | Changes the specified properties of the user. |
| --remov | Removes the user from a group. |
| --set-g | Sets the primary group for the specified user. The primary group is the first group the user belongs to. This option will remove the user from this primary group and add it to the group specified by the --group command-line option. |
| --stat | Prints detailed information about the specified user(s). |
| --whoam | Same as --list, but only lists the current user, the user that authenticated on the controller. |

### Options

| Name, shorthand | Description |
|---|---|
| --cmon-user, | Username on the CMON system |
| --group=GROU | Set the name of the group. For example, when a new user is created this option can be used to control what will be the primary group of the new user. It is also possible to filter the users by the group name while listing them. |
| --create-grc | If this command-line option is provided and the group for the new user does not exist the group will be created together with the new user. |
| --first-name | Sets the first name of the user. |
| --last-name= | Sets the last name of the user. |
| --public-key | The name of the file where the public key is stored. Please note that this currently only works with the --add-key option. |
| --preference | List of a given --cmon-user preferences to insert/update to/in the cmon database. It is used with --preferences main option. The LIST consists of the key=value pairs separated by a semicolon (;). |
| --preference | List of a given --cmon-user preferences to delete from the cmon database. It is used with --preferences main option. The LIST consists of the keys separated by a semicolon (;). |
| --title=TITLE | The title prefix (e.g. Dr.) for the user. |
| --email-addr | The email address for the user. |
| --new-passwc | The new password when changing the password. |
| --old-passwc | The old password when changing the password. |
| --user-forma | The string controls the format of the printed information about the users. See *User Formatting*. |
| --without-ta | When listing the users this option can be used to limit the list to those users that have none of the specified tags set. |
| --with-tags= | When listing the users this option can be used to limit the list to those users that have at least one of the tags set. |

### Creating the First User

To use the s9s command line tool a Cmon User Account is needed to authenticate on the Cmon Controller. These user accounts can be created using the s9s program itself by either authenticating with a pre-existing user account or bootstrapping the user management to create the very first user. The following section describes the authentication and the creation of the first user in detail.

If there is a username specified either in the command line using the --cmon-user (or -u) options or in the configuration file (either the ~/.s9s/s9s.conf or the /etc/s9s.conf file using the cmon_user variable name) the program will try to authenticate with this username. Creating the very first user account is of course not possible this way. The --cmon-user option and the cmon_user the variable is not for specifying what user we want to create, it is for setting what user we want to use for the connection.

If no user name is set for the authentication and user creation is requested the s9s will try to send a request to the controller through a named pipe. This is the only way a user account can be created without authenticating with an existing user account, this is the only way the very first user can be created. Here is an example:

```
s9s user --create \
       --group=admins \
       --generate-key \
       --controller=https://192.168.1.127:9501 \
       --new-password="MyS3cr3tpass" \
       --email-address="user@example.com" \
       admin
```

Consider the following:

1. There is no `--cmon-user` specified, this is the first user we create, we do not have any pre-existing user account. This command line is for creating the very first user. Please check out the *Examples* section to see how to create additional users.
2. This is the first run, so we assume no `~/.s9s/s9s.conf` configuration file exists, there is no user name there either.
3. In the example, we create the user with the username "admin", and it is the command-line argument of the program.
4. The command specifies the controller to be at `https://192.168.1.127:9501`. The HTTPS protocol will be used later, but to create this first user the s9s will try to use SSH and sudo to access the controller's named pipe on the specified host. For this, to work the UNIX user running this command has to have a passwordless SSH and sudo set up to the remote host. If the specified host is the local host the user does not need SSH access, but still needs to be root or have a passwordless sudo access because of course, the named pipe is not accessible for anyone.
5. Since the UNIX user has no s9s configuration file it will be created. The controller URL and the user name will be stored in it under `~/.s9s/s9s.conf`. The next time this user runs the program, it will use this "admin" user unless another user name is set in the command line and will try to connect this controller unless another controller is set in the command line.
6. The password will be set for the user on the controller, but the password will never be stored in the configuration file.
7. The `--generate-key` the option is provided, so a new private/public key pair will be generated, then stored in the `~/s9s/` directory and the public key will be registered on the controller for the new user. The next time the program runs it will find the username in the configuration file, find the private key in place for the user, and will automatically authenticate without a password. The command-line options will always have precedence, so this automatic authentication is simply the default way, the password authentication is always available.
8. The group for the new user is set to "admins", so this user will have special privileges. It is always a good idea to create the very first user with special privileges, then other users can be created by this administrator account.

### User List

Using the `--list` and `--long` command-line options a detailed list of the users can be printed. Here is an example of such a list:

```
$ s9s user --list --long worf jadzia
A ID UNAME  GNAME EMAIL           REALNAME
- 11 jadzia ds9   dax@ds9.com     Lt. Jadzia Dax
```

(continues on next page)

(continued from previous page)

```
A 12 worf   ds9   warrior@ds9.com Lt. Worf
Total: 12
```

Please note that there are a total of 12 users defined on the system, but only two of those are printed because we filtered the list with the command line arguments.

The list contains the following fields:

| Field | Description |
|---|---|
| A | Shows the authentication status. If this field shows the letter 'A' the user is authenticated with the current connection. |
| ID | Shows the user ID, a unique numerical ID identifying the user. |
| UNAME | The username. |
| GNAME | The name of the primary group of the user. All user belongs to at least one group, the primary group. |
| EMAIL | The email address of the user. |
| REALNAME | The real name of the user consists of first name, last name, and some other parts, printed here as a single string composed of all the available components. |

### User Formatting

When this command-line option is used the specified information will be printed instead of the default columns. The format string uses the `%` character to mark variable fields and flag characters as they are specified in the standard `printf()` C library functions. The '`%`' specifiers are ended by field name letters to refer to various properties of the users.

The `%+12I` format string for example has the `+12` flag characters in it with the standard meaning: the field will be 12 characters wide and the + or − the sign will always be printed with the number. The properties of the user are encoded by letters. For example, the letter `N` encodes the username field, so `username` of the user will be substituted. Standard `\` the notation is also available, `\n` for example, encodes a new-line character. The `%16N` for example, the letter `N` encodes the username field, so `username` of the user will be substituted. Standard `\` notation is also available, `\n` for example, encodes a new-line character.

The `s9s` command supports the following fields:

| Field | Description |
|---|---|
| d | The distinguished name of the user. This currently has meaning only for users originated from an LDAP server. |
| F | The full name of the user. |
| f | The first name of the user. |
| G | The names of groups the given user belongs to. |
| I | The unique numerical ID of the user. |
| j | The job title of the user. |
| l | The last name of the user. |
| M | The email address of the user. |
| m | The middle name of the user. |
| N | The username for the user. |
| o | The origin of the user, the place that used to store the original instance of the user. The possible values are "CmonDb" for users from the Cmon Database or "LDAP" for users from the LDAP server. |
| P | The CDT path of the user. |
| t | The title of the user (e.g. "Dr."). |

### Examples

- Create a remote s9s user and generate an SSH key for the user:

```
s9s user --create \
        --generate-key \
        --cmon-user=dba \
        --controller="https://10.0.1.12:9501"
```

- List out all users:

```
$ s9s user --list --long
A ID UNAME      GNAME  EMAIL REALNAME
-  1 system     admins -     System User
-  2 nobody     nobody -     -
A  3 dba        users  -     -
-  4 remote_dba users  -     -
```

- List users that have at least one of the tags set.

```
s9s user \
        --list \
        --long \
        --with-tags="ds9;tng"
```

- Register a new public key for the active user:

```
s9s user \
        --add-key \
        --public-key-file=/home/pipas/.ssh/id_rsa.pub \
        --public-key-name=The_SSH_key
```

- Add user "myuser" into group "admins":

```
s9s user \
        --add-to-group \
        --group=admins \
        myuser
```

- Set a new password for user "pipas":

```
s9s user \
        --change-password \
        --new-password="MyS3cr3tpass" \
        pipas
```

- Create a new user called "john" under group "dba":

```
s9s user \
        --create \
        --group=dba \
        --create-group \
        --generate-key \
        --new-password=s3cr3tP455 \
        --email-address=johndoe@somewhere.com \
        --first-name=John \
        --last-name=Doe \
        --batch \
        john
```

- Delete an existing user called "mydba":

```
s9s user \
        --delete \
        mydba
```

- Disable user nobody:

```
s9s user \
        --cmon-user=system \
        --password=secret \
        --disable \
        nobody
```

- Enable user nobody:

```
s9s user \
        --cmon-user=system \
        --password=secret \
        --enable \
        nobody
```

- Reset the password for a user called "dba". One has to obtain the one-time token which will be sent to the registered email address of the corresponding user, followed by the actual password modification command with `--token` and `--new-password` parameters:

```
s9s user \
        --password-reset \
        dba
## check the mail inbox of the respective user
s9s user \
        --password-reset \
        --token="98197ee4b5584cedba88ef1f583a1258" \
        --new-password="newp455w0rd"
        dba
```

- Set a primary group for user "dba":

```
s9s user \
        --set-group \
        --group=admins \
        dba
```

- To add/set/delete the preferences for a given `--cmon-user`.

```
s9s user --cmon-user=s9s --password=pass1299 --preferences --
↪preferences-to-set="key1=value1;key2=value2;key3=value3"
s9s user --cmon-user=s9s --password=pass1299 --preferences --
↪preferences-to-set="key2=QWERTY"
s9s user --cmon-user=s9s --password=pass1299 --preferences --
↪preferences-to-delete="key1;key2"
```

- To get the preferences for a given `--cmon-user`.

```
s9s user --cmon-user=s9s --password=***** --get-preferences
```

## 4.6 ClusterControl SSH

This optional package is introduced in ClusterControl v1.4.2. This module provides the ability to connect to the SSH console to any of your cluster hosts directly via ClusterControl UI. This can be very useful if you need to quickly log into a database server and access the command line.

The package installs a binary called `cmon-ssh` located under `/usr/sbin` directory and by default listens to port 9511 on the ClusterControl node. It interacts directly with the target host via SSH protocol using the credential (`os_user` and `ssh_identity`) configured when deploying or importing the cluster into ClusterControl.

Set up the RewriteRule inside the Apache configuration file `cc-frontend.conf` (above the `<Directory/>` definitions):

```
<Location /api/v2/cmon-ssh/cmon/ws/>
    RewriteEngine On
    RewriteCond %{REQUEST_URI} ^/cmon-ssh/cmon/ws/(.*)$
    RewriteCond %{REQUEST_URI} ^/api/v2/cmon-ssh/cmon/ws/(.*)$
    RewriteRule ^(.*)$ ws://127.0.0.1:9511/cmon/ws/%1 [P,L]
</Location>
<LocationMatch /api/v2/cmon-ssh/>
```

(continues on next page)

(continued from previous page)

```
    ProxyPass http://127.0.0.1:9511/
    ProxyPassReverse http://127.0.0.1:9511/
</LocationMatch>
```

Enable the following Apache modules:

```
$ a2enmod proxy proxy_http proxy_wstunnel
```

Communication is based on HTTPS, so it is possible to access your servers from behind a firewall that restricts Internet access to only port 443. Access to WebSSH is configurable by the ClusterControl admin through the GUI.

**Warning**

Warning!

ClusterControl does not provide extra layers of authentication and authorization when accessing the cluster from a web-based SSH terminal. The user who has access to the cluster in the ClusterControl UI may capable of accessing the terminal as a privileged user. Use the access privileges to limit them. See *Users and Teams*.

## 4.7 ClusterControl Notifications

This optional package is introduced in ClusterControl v1.4.2, deprecating the previous version of the ClusterControl NodeJS package (which served the same purpose). Alarms and events can now easily be sent to incident management services like PagerDuty, VictorOps, and OpsGenie. You can also run any command available in the *ClusterControl CLI* from your CCBot-enabled chat services like Slack and Telegram. Additionally, it provides a generic webhook if you want to integrate with other services to act on status changes in your clusters. The direct connections with these popular incident communication services allow you to customize how you are alerted from ClusterControl when something goes wrong with your database environments.

The package installs a binary called `cmon-events` located under `/usr/sbin` directory and by default listens to port 9510 on the ClusterControl node.

Additional resources on setting up integration with third-party tools are listed below:

| Field | Description |
|---|---|
| Blogs | - Introducing the ClusterControl Alerting Integrations - Video: ClusterControl ChatOps Integrations – Product Demonstration |
| Documentation | - *Notification Services* |

## 4.8 ClusterControl Cloud

This optional package is introduced in ClusterControl v1.5. The package name is `clustercontrol-cloud` and it provides a GUI extension in ClusterControl under *Cloud Provider Integrations*.

This package installs the following new files:

- `/etc/cron.d/cmon-cloud` – Cron's job to monitor `cmon-cloud` process.
- `/etc/rc.d/init.d/cmon-cloud` – sysvinit script.
- `/etc/systemd/system/cmon-cloud.service` – Systemd unit file.
- `/usr/sbin/cmon-cloud` – The executable binary file.

By default, this service will use port 9518 on localhost interface. Cloud credentials are stored under `/var/lib/cmon/` with permission set to root only.

**Info**

ClusterControl Cloud is a reintroduction of a feature called "Service Providers", available in ClusterControl 1.4 and older.

### 4.8.1 Commands

| Name, Shorthand | Description |
|---|---|
| upload | Upload a file to the cloud. |
| download | Download a file from the cloud. |
| rm | Delete a file on the cloud. |
| clouds | List supported cloud providers. |
| help, h | Shows a list of commands or help for one command. |

### 4.8.2 Options

| Shorthand, Option | Description |
|---|---|
| `-cloud` | Use this cloud for the action. |
| `--service` | Use this service within the cloud. |
| `--credentials` | Raw credentials JSON. |
| `--credentials-file` | Take credentials from this file instead of using STDIN input. |
| `--auto-create-bucket` | Use this flag if the bucket should be auto-created if not exists. |
| `--help, -h` | Show help. |
| `--version` | Print the version. |

## 4.9 ClusterControl Cloud File Manager

This optional package is complimentary of *ClusterControl Cloud* and is introduced in ClusterControl v1.5. The package name is `clustercontrol-clud` (an abbreviation of **CL**oud **U**pload **D**ownload) and is the command-line interface for ClusterControl to interact with the cloud providers when uploading and downloading backups. You can think of it as a cloud file manager. No extra configuration is required.

This package installs the following file:

- `/usr/sbin/clud` – The executable binary file.

This tool is used by ClusterControl in the following scenarios:

1. After an on-demand or scheduled database backup has been successfully taken and if it is configured to be uploaded to the cloud, when **Upload backup to cloud** is toggled on.
2. When you want to manually upload an existing backup to the cloud at ***Backups → choose a backup → Actions → Upload***.
3. When you want to download an existing backup from the cloud to the host (controller or database node) at ***Backups → choose a backup → Actions → Download***.

### 4.9.1 Usage

The general synopsis to execute commands using `clud` is:

```
clud {global options} command {command option} {arguments...}
```

### 4.9.2 Commands

| Name, shorthand | Description |
|---|---|
| upload | Upload a file to the cloud. |
| download | Download a file from the cloud. |
| rm | Delete a file on the cloud. |
| clouds | List supported cloud providers. |
| help, h | Shows a list of commands or help for one command. |

### 4.9.3 Options

| Name, shorthand | Description |
|---|---|
| `--cloud` | Use this cloud for the action. |
| `--service` | Use this service within the cloud. |
| `--credentials` | Raw credentials JSON. |
| `--credentials-file` | Take credentials from this file instead of using STDIN input. |
| `--auto-create-bucket` | Use this flag if the bucket should be auto-created if not exists. |
| `--help, -h` | Show help. |
| `--version` | Print the version. |

### 4.9.4 Examples

List out the supported cloud providers:

```
$ clud clouds
SUPPORTED CLOUD PROVIDERS:
     aws - Amazon Web Services     gce - Google Cloud     az - Microsoft␣
↪Azure     s3 - Any S3-Compatible Cloud
```

Commonly, ClusterControl performs the upload to cloud task by invoking the following command:

```
/usr/sbin/clud \
    --cloud <supported_cloud_provider> \
    --credentials-file '<path_to_credential.json>' \
```

(continues on next page)

(continued from previous page)

```
    --service unknown \
    --auto-create-bucket \
    upload <bucket_name> <source_file_path> <destination_file_path_in_cloud>
```

The following is an example to upload a file to S3-compatible object storage:

```
$ /usr/sbin/clud --cloud s3 --credentials-file 'mycreds.json' --service␣
↪unknown --auto-create-bucket upload mybucket1 my_data.sql␣
↪folder1/my_data.sql
UPLOAD COMPLETE
Bucket: mybucket1
Location: folder1/my_data.sql
```

Where the content of `mycreds.json` is:

```
{
  "access_key_id": "AKIAIOSFODNN7EXAMPLE",
  "access_key_secret": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
  "endpoint": "s3.us-east-1.amazonaws.com",
  "region": "us-east-1"
}
```

## 4.10 ClusterControl Proxy

ClusterControl Proxy is the next-generation ClusterControl manager which creates a unified view of multiple ClusterControl controllers and providing web services, effectively removing dependencies for Apache or nginx web servers to serve ClusterControl GUI components. It is developed to purposedly handle the ClusterControl GUI component provided by package `clustercontrol-mcc`.

Having said that, when installing ClusterControl GUI (package name: `clustercontrol-mcc`), ClusterControl Proxy (package name: `clustercontrol-proxy`) is expected be installed together.

### 4.10.1 ClusterControl Manager

The ClusterControl Manager binary called ccmgr is the main process (default port: 19501) which serves the web contents of ClusterControl GUI and acts as a proxying layer to all ClusterControl controllers added into it. The process is controlled by a systemd unit file located at `/etc/systemd/system/cmon-proxy.service`. Hence, the service management should follow the standard systemd format:

- Starting the service:

```
systemctl start cmon-proxy
```

- Stopping the service:

```
systemctl stop cmon-proxy
```

- Restart the service:

```
systemctl restart cmon-proxy
```

The main binary is installed at `/usr/bin/ccmgr` and comes with the following options:

| Option | Description |
|---|---|
| `--debug-cmon-rpc` | Debug log RPC requests to cmon |
| `--debug-web-rpc` | Debug web RPC requests to cmon-proxy |
| `--basedir=` | The basedir of configuration (default: `/usr/share/ccmgr/`) |

This binary is expected to be running as a service. Use *ClusterControl Manager Admin CLI* to manage the ClusterControl Manager.

### 4.10.2 ClusterControl Manager Admin CLI

The `ccmgradm` is a command-line interface to manage ClusterControl Manager users, passwords and controllers. Run `ccmgradm --help` to see all available commands.

**Example**

```
$ ccmgradm  --help
ClusterControl Manager - admin CLI v2.2
Usage: ccmgradm <command> [<args>]

Options:
--help, -h              display this help and exit

Commands:
dropuser
adduser
setpassword
dropcontroller
addcontroller
init
updatecontroller
listcontrollers
enableMcc
```

For every available command, there is a further help instruction:

**Example**

```
$ ccmgradm adduser --help
ClusterControl Manager - admin CLI v2.2
Usage: ccmgradm adduser [--email EMAIL] [USERNAME [PASSWORD]]

Positional arguments:
USERNAME
PASSWORD

Options:
```

```
--email EMAIL, -e EMAIL
--help, -h              display this help and exit
```

You may use the `ccmgradm` client to interact with the `ccmgr` service or use the ClusterControl Ops-C web GUI client to manage users and controllers.

### 4.10.3 Configuration file

By default, ClusterControl Proxy comes with a default configuration in YAML format, available at `/usr/share/ccmgr/ccmgr.yaml`:

```
filename: /usr/share/ccmgr/ccmgr.yaml
webapproot: /var/www
fetch_jobs_hours: 12
fetch_backups_days: 7
timeout: 30
logfile: /var/log/ccmgr.log
frontend_path: /app
port: 19051
tls_cert: /usr/share/ccmgr/server.crt
tls_key: /usr/share/ccmgr/server.key
session_ttl: 1800000000000
single_controller: ""
```

The following is an example content of the configuration file with one ClusterControl instance:

```
filename: ccmgr.yaml
instances: # List of controller
    - xid: chebjd8gfi863qqhrolg # Controller id. Generated automatically
      url: hostname:9443/api # Controller api URL
      name: Controller name
      username: admin
      password: password
      frontend_url: http://hostname:9443/ # Url of Cluster Control web␣
↪interface
      cmon_ssh_host: 127.0.0.1:9511 # cmon-ssh host and port, used for web␣
↪ssh console. Default for single controller - 127.0.0.1:9511
      cmon_ssh_secure: false # if true - use TLS for cmon-ssh connection
    - xid: cnoi4d3fo0o9e9m7hap0 # Controller id. Generated automatically
      # ....
timeout: 30 # request timeout in seconds. Default 30
logfile: ccmgr.log # cmon proxy log file. Default ccmgr.log
users: # Cmon proxy users, can be set via ccmgradm setpassword <username>
↪<password>
    - username: admin
      passwordhash: <hash>
frontend_path: /app # Path to static files that cmon proxy can serve
port: 19051
tls_cert: server.crt # Default server.crt can be set from env␣
↪TLS_CERTIFICATE_FILE variable
```

(continues on next page)

(continued from previous page)

```
tls_key: server.key # Default server.key can be set from env TLS_KEY_FILE␣
↪variable
session_ttl: 3600000000000 # Session time to live in nanoseconds. Default 1␣
↪hour
fetch_backups_days: 7 # How many days in the past cmon-proxy should look␣
↪for backup data. Default is 7 days
fetch_jobs_hours: 12 # How many hours in the past cmon-proxy should look␣
↪for jobs data. Default is 12 hours
single_controller: chebjd8gfi863qqhrolg # ID of single controller, if it is␣
↪set - multi-controller is disabled
k8s_proxy_url: http://127.0.0.1:8080 # Kubernetes proxy URL. Default is␣
↪http://127.0.0.1:8080
kubernetes_enabled: true # Enable Kubernetes proxy. Default is true
license_proxy_url: "https://severalnines.com/service/lic.php" # URL to␣
↪request demo license during registration. Default -␣
↪https://severalnines.com/service/lic.php
http_port: 80 # Port for the plain HTTP server used for ACME challenges and␣
↪redirection to HTTPS. Default is 80

# Let's Encrypt settings for automatic TLS certificates
acme_enabled: false # Enable Let's Encrypt. Default is false
acme_staging: false # Use Let's Encrypt staging environment. Default is␣
↪false
acme_domains: [] # List of domains to obtain certificates for.
acme_email: "" # Email address for Let's Encrypt registration and␣
↪notifications.
acme_cache_dir: "autocert-cache" # Directory to cache ACME certificates.␣
↪Default is "autocert-cache" in the base directory.
acme_directory_url: "" # The ACME directory URL. Defaults is empty (using␣
↪library pre-defined), or staging␣
↪(https://acme-staging-v02.api.letsencrypt.org/directory) if acme_staging␣
↪is true.
acme_accept_tos: true # Automatically accept the ACME provider's Terms of␣
↪Service. Default is true.
acme_renew_before: "720h" # Renewal window before certificate expiration (a␣
↪Go duration string, e.g., "720h" for 30 days). Default is 30 days (720h).
acme_host_policy_strict: false # If true, strictly enforce that certs are␣
↪only issued for domains in acme_domains. Recommended for production.␣
↪Default is false.
web_server:
  trusted_proxies: [] # List of trusted proxy IPs. Default is empty.
  trusted_platform: "" # Platform header for identifying client IPs (e.g.,
↪"X-Forwarded-For"). Default is empty.
  security:
    frame_deny: true # Sets X-Frame-Options to DENY. Default is true.
    sts_seconds: 31536000 # HSTS max-age in seconds. Default is 31536000 (1␣
↪year).
    sts_include_subdomains: true # Include subdomains in HSTS policy.␣
↪Default is true.
```

(continues on next page)

(continued from previous page)

```
    sts_preload: true # Enable HSTS preload. Default is true.
    force_sts_header: true # Force HSTS header even for non-TLS requests.␣
↪Default is true.
    content_type_nosniff: true # Sets X-Content-Type-Options to nosniff.␣
↪Default is true.
    browser_xss_filter: false # Disables the browser's XSS filter. Default␣
↪is false.
    content_security_policy: "default-src 'self'; ..." # Content Security␣
↪Policy. See below for details.
    content_security_policy_report_only: true # Use CSP in report-only␣
↪mode. Default is true.
    referrer_policy: "strict-origin-when-cross-origin" # Referrer-Policy␣
↪header. Default is "strict-origin-when-cross-origin".
    permissions_policy: "camera=(), microphone=(), geolocation=()" #␣
↪Permissions-Policy header. Default is "camera=(), microphone=(),␣
↪geolocation=()".
  cors:
    allow_origins: [] # List of allowed origins for CORS. Default is empty.
    allow_methods: [] # List of allowed methods for CORS. Default is empty.
    allow_headers: [] # List of allowed headers for CORS. Default is empty.
    expose_headers: [] # List of headers to expose for CORS. Default is␣
↪empty.
    allow_credentials: true # Allow credentials for CORS. Default is true.
    max_age_seconds: 0 # Max age for preflight requests. Default is 0.
  gzip:
    level: 1 # Gzip compression level (1 to 9). Default is 1 (BestSpeed).
  frontend:
    nonce_replacement_files: ["index.html"] # Files to replace CSP nonce␣
↪in. Default is ["index.html"].
```

### 4.10.4 Configure custom SSL certificates

The default directory `/usr/share/ccmgr` shall contain the configuration files and TLS certificates and keys, as well as the logs of the `cmon-proxy` application. You can put your own TLS cert (a combined full chain, so the main cert and all the CAs together, or just a self signed one) and key into your data directory using the following file names - `server.crt` and `server.key`.

### 4.10.5 Web server: headers and simple examples

The ClusterControl configuration supports security headers which can be configured in the `ccmgr.yaml`. This section simplifies `web_server` settings and details the headers they add. These settings do not change the app logic; they only affect how requests are interpreted and what security headers are sent.

#### Content Security Policy (CSP)

The Content Security Policy (CSP) limits where scripts, styles, and images can load from. The inline scripts can be safely allowed with support for a per-response nonce and it can run in report-only mode while you test your policy (report-only by default).

How to apply the configuration:

- Configure `web_server.security.content_security_policy` as a string. Use `{{nonce}}` where a nonce should appear (e.g., in `script-src`).
- For HTML files listed in `web_server.frontend.nonce_replacement_files` (default: index.html), the server:
  1. Generates a fresh nonce per request
  2. Replaces **NONCE** in your HTML with that nonce
  3. Sends CSP header with {{nonce}} replaced
  4. Disables caching for those HTML responses
- If `content_security_policy_report_only: true`, the header is sent as `Content-Security-Policy-Report-Only` and the directive `upgrade-insecure-requests` is removed automatically for compatibility.

Example configuration and usage:

```
web_server:
  security:
    content_security_policy: "default-src 'self'; script-src 'self'
↪'nonce-{{nonce}}' 'strict-dynamic'; style-src 'self' 'unsafe-inline';␣
↪img-src 'self' data:;"
    content_security_policy_report_only: true
  frontend:
    nonce_replacement_files: ["index.html"]
```

```
<!-- index.html -->
<script nonce="__NONCE__">/* inline boot script */</script>
```

Resulting header (report-only example):

```
Content-Security-Policy-Report-Only: default-src 'self'; script-src 'self'
↪'nonce-<random>' 'strict-dynamic'; style-src 'self' 'unsafe-inline';␣
↪img-src 'self' data:
```

### CORS (Cross-Origin Resource Sharing)

The CORS is a security mechanism built into web browsers that controls how resources (like APIs, images, fonts, or scripts) can be requested from a domain different from the one that served the original web page. If the frontend (browser app) is served from a different origin (domain/port) than this API.

Example configuration:

```
web_server:
  cors:
    allow_origins: ["https://ui.example.com"]
    allow_methods: ["GET", "POST", "PUT", "PATCH", "DELETE"]
    allow_headers: ["Content-Type", "Authorization"]
    expose_headers: ["X-Request-Id"]
    allow_credentials: true
    max_age_seconds: 600
```

Typical preflight (OPTIONS) response:

```
Access-Control-Allow-Origin: https://ui.clustercontrol.com
Access-Control-Allow-Methods: GET,POST,PUT,PATCH,DELETE
Access-Control-Allow-Headers: Content-Type,Authorization
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 600
```

#### HTTPS enforcement (HSTS)

The HSTS is a web security policy mechanism that tells browsers to always use HTTPS (encrypted connection) when communicating with a website. Without HSTS, if a client types http://example.com (or clicks an old http link), the browser will first connect over HTTP, then get redirected to HTTPS. That short HTTP step is vulnerable to `man-in-the-middle (MITM)` attacks, where an attacker could intercept or modify the redirect.

Example of headers sent:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

The `max-age` for 1 year (31536000) indicate the browser must only use HTTPS for the site and `includeSubDomains` configuration apply to all the subdomains.

## 4.11 ClusterControl Agent

Also known as the **query monitoring agent**. Deploys agents for monitoring the database nodes, particularly the database queries. It is only available for MySQL and PostgreSQL database systems.

Features you can use with Query Monitor/Agent are the following:

- Install/remove query monitor agents on each database node. You can remove these agents later on.
- Start/stop collecting query stats with the agents.
- New 'Query Workload' overview showing query digests, latency, throughput, and concurrency with a scatter chart.

### 4.11.1 Query Monitor Agents

#### ClusterControl GUI Install agent

Log in to your ***ClusterControl GUI → choose a database cluster → Performance → Query Monitor/Agents***.When you click the Install monitoring agent button, ClusterControl will set up and install an agent called `cmnd` to all of the database nodes in that particular cluster.

#### ClusterControl CLI Install agent

For CLI, we will use the `s9s cluster` option: Deploys CMON agents to all nodes in cluster ID 1. (This enables TopQuery Monitoring functionality for the cluster nodes.):

```
s9s cluster \
        --deploy-cmonagents \
        --cluster-id=1 \
        --print-request \
        --log \
        --wait
```

Uninstalls CMON agents from specified nodes in cluster ID 1. (If no node is specified, CMON agents will be uninstalled from all nodes in the cluster.)

```
s9s cluster \
        --uninstall-cmonagents \
        --cluster-id=1 \
        --nodes=10.67.199.164 \
        --print-request \
        --log \
        --wait
```

### Agent Configuration

The agent has the following configuration which you can find in `/etc/cmnd.conf`. This is for the agent configuration. `/etc/cmnd.d` directory for the Top Query configuration. The agent configuration file `/etc/cmnd.conf` has all the details you can find here. By default, these are the following options that are set:

```
$cat /etc/cmnd.conf | sed -e 's/#.*$//' -e '/^$/d' -e '/^$/N;/^\n$/D' | sed
↪'/^\t*$/d'

[cmnd]
data_directory = "/var/lib/cmnd"
log_file = "/var/log/cmnd.log"
pid_file = "/var/run/cmnd.pid"
plugin_names = [
"libpluginMySql.so",
"libpluginSqlite.so",
"libpluginPgSql.so"
]
port = 4433
cert_file = "/etc/ssl/cmnd/cert.pem"
key_file = "/etc/ssl/cmnd/key.pem"
```

You can change the port to your desired port number based on the firewall policy you have in your organization. So take note, you need to have port **4433** open by default to make it work properly.

You can find more of its configuration files under these files/directories:

```
$ find /etc/ -name "cmn*"

/etc/ssl/cmnd
/etc/logrotate.d/cmnd
/etc/cmnd.conf
/etc/cmnd.d
/etc/cmn.passwd
```

The Query Monitor Agents are installed using systemd, allowing you to start and stop the service using the `systemctl` command. To determine if the agents are running, you can do the following:

```
systemctl status cmon-daemon
```

Or verify it with the `netstat` just like below:

```
$ netstat -ntlvp46 | grep cmnd

tcp 0 0 0.0.0.0:4433 0.0.0.0:* LISTEN 8456/cmnd
```

#### Query Monitoring Workflow

The agent collects samples from the `performance_schema` (MySQL/MariaDB) or `pg_stat_statements` (PostgreSQL/TimescaleDB). These metrics which is collected by a database client and send the query inquiry to the agent. Then, the agent that reads the input data will produce the query statistics and make it available to the Query Monitor's Overview dashboard query metrics. That means all collected information from the Query Monitoring shall depend upon the following conditions:

1. The reply should be sent in a timely fashion no matter how big the input data is. Should the computer be very slow or the processing is too complicated, the query statistics should be produced and made available in a progressive manner so a responsive UI can be implemented.
2. The agent should be able to update the *query statistics* on itself when the *input data* is updated by the database server. A simple query or RPC call is invoked so that the agent keeps monitoring the input data and keeps the data updated.
3. The *query statistics* should be according to the *query inquiry* and the *input data*. In simpler words, the produced statistical data should be in sync with the input data and the query the user sent. Some query statistics may be derived from the input data.

## 4.12 ClusterControl CMON RPC v2 API Reference

### 4.12.1 Introduction

This is the documentation of the RPC v2, a new RPC interface for the Cmon Controller that exists beside the original (or v1) interface. The two versions of the RPC interface are independent from each other, one can be used without the services of the other.

**Connection requirements**

RPC v2 is available only over TLS (HTTPS). Connect to port **9501** on the `/v2` path prefix. Almost all endpoints require authentication first.

Please note that the RPC v2 API calls are available only using secure (TLS) connections, encryption is mandatory. The RPC v2 also uses an authentication call, very little of the calls are available to be used without authenticating first.

By default the encrypted communication can be used through port 9501 and the RPC v2 requests are served on the `/v2` path (e.g. `/v2/clusters`).

### 4.12.2 Available Endpoints

| Path | Description |
|---|---|
| `/v2/auth` | Authentication. |
| `/v2/alarm` | Information about alarms. |
| `/v2/users` | Handling Cmon users. |
| `/v2/backup` | Creating and restoring backups. |
| `/v2/clusters` | Information about clusters. |
| `/v2/host` | Information about hosts. |
| `/v2/stat` | Statistical information, measurements. |
| `/v2/jobs` | Creating and monitoring jobs. |
| `/v2/maintenance` | Maintenance periods. |
| `/v2/metatype` | Information about types used by the controller. |
| `/v2/config` | Configuration. |
| `/v2/process` | Processes. |
| `/v2/proxysql` | ProxySQL |
| `/v2/log` | Logging. |
| `/v2/imperative` | Maintaining and running scripts. |
| `/v2/discovery` | Collecting real-time data. |
| `/v2/cloud` | Managing cloud credentials and proxying to cmon-cloud. |
| `/v2/audit` | List the audit log of the controller. |
| `/v2/ca` | Certificate authority. |
| `/v2/dbversions` | Get supported database versions for specific cluster type & vendor. |
| `/v2/performance` | Get information about MySQL databaseperformance. |
| `/v2/logical_replication` | PostgreSQL Logical Replication. |
| `/v2/watchlists` | Manipulate whatchlists to visualize clusters according to user configuration. |

Here is an example showing a typical RPC v2 request:

#### Request structure

```
{
    "cluster_name": "ft_galera_18262",
    "operation": "getClusterInfo",
    "request_id": 2,
```

(continues on next page)

(continued from previous page)

```
    "request_created": "2017-06-22T06:13:46.393Z",
}
```

Most of the requests contain at least the following fields:

- `operation`: This field identifies what kind of action the controller should take to serve the request.
- `request_id`: Optional field that will be sent back with the reply.
- `request_created`: This is an optional field which will be sent back to the client with the reply.
- `cluster_name`: Many of the requests are closely related to one cluster and that cluster can be identified by the cluster ID or the cluster ID.
- `cluster_name`: Many of the requests are closely related to one cluster and that cluster can be identified by the cluster ID or the cluster ID.

The RPC v2 reply usually contains many fields, but some fields can be found in all the replies. Here is an example showing some of the fields that are always sent with the reply:

### Response structure

```
{
    "request_id": 2,
    "request_created": "2017-06-22T06:13:46.339Z",
    "request_processed": "2017-06-22T06:13:46.385Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

- `messages`: A list of strings with human readable messages describing what happened while processing the request.
- `error_string`: A human readable error message describing the error if an error happened. Should a request result in more than one message strings the "messages" field will be sent.
- `request_id`: If the request held this field it will be simply sent back in the reply.
- `request_created`: This is just a field that is sent back by the controller without modification.
- `request_processed`: The timestamp shows what time the reply was created.
- `request_status`: A string showing if the request was successfully processed and the error code if not. The possible values are:
  - `Ok`: The request was successfully processed.
  - `InvalidRequest`: Something was fundamentally wrong with the request.
  - `ObjectNotFound`: The referenced object (e.g. the cluster) was not found.
  - `TryAgain`: The request cannot be processed at this time; the client should retry.
  - `ClusterNotFound`: The specified cluster was not found or is not running.
  - `UnknownError`: The exact error could not be identified.
  - `AccessDenied`: The authenticated user has insufficient rights.
  - `AuthRequired`: The client must authenticate first.

- `Redirect`: The request should be sent to a different controller (see `controllers` field in the reply).

- `error_id`: An optional integer that identifies a specific error condition within a broader `request_status` category. Present only when the error warrants programmatic handling beyond what `request_status` alone provides. Clients should use `error_id` to decide on recovery actions and `error_string` for the human-readable description.

  The current values are:

| error_ | Name | request_st | Meaning and recommended client action |
|---|---|---|---|
| 1 | ClusterDoesNotExist | ClusterNotFound | The cluster does not exist in the pool at all. Stop retrying — refreshing the controller–cluster map will not help. |
| 2 | AuthenticationRequired | AuthRequired / AccessDenied | The session is missing or has expired. Re-authenticate before retrying. |
| 3 | InvalidCredentials | AccessDenied | The username or password was incorrect. Prompt the user for new credentials. |
| 4 | LicenseRequ | AccessDenied | The operation is not permitted with the current license. Direct the user to license management. |
| 5 | ControllerInitializing | TryAgain | The controller is still starting up. Retry after a short delay. |
| 6 | ClusterNotAssignedToController | ObjectNotFound | The target cluster exists but is managed by a different controller. Refresh the controller–cluster map and re-send the request to the correct controller. The owning controller's numeric id is provided in the `owner_controller_id` field when known (see below). |

  When `error_id` is `ClusterNotAssignedToController` (6), the reply may also include:

  - `owner_controller_id`: The numeric id of the controller that currently owns the cluster, when it is known. Clients can use it as a redirect hint to re-send the request to the owning controller directly, without re-discovering the whole controller–cluster map. (This is distinct from the `controller_id` field, which always carries the id of the controller that produced the reply.)

  Example error reply for a cluster owned by another controller (carries `error_id` 6 and the `owner_controller_id` redirect hint):

```
{
    "controller_id": "10207",
    "error_id": 6,
    "error_string": "Request aborted. Cluster 59 is not assigned to␣
↪the controller.",
    "owner_controller_id": 10208,
    "request_processed": "2025-05-01T19:57:13.000Z",
    "request_status": "ObjectNotFound"
}
```


Example error reply for a cluster that does not exist (carries `error_id` 1):


```
{
    "error_id": 1,
    "error_string": "Request aborted. Cluster 1 does not exist.",
    "request_processed": "2025-05-01T19:57:13.000Z",
    "request_status": "ClusterNotFound"
}
```


- `request_user_id`: The numeric ID of the authenticated user. Using this field the client can identify who sent the request.

### 4.12.3 Alarms

| Operation | Description |
|---|---|
| *getStatistics* | Returns statistical information about alarms for one or more clusters, including a count of active critical and warning alarms. |
| *getAlarm* | Get information about one particular alarm. |
| *getAlarms* | Get information about multiple alarms. |
| *ignoreAlarm* | Sets the ignored state of an alarm. |

#### getStatistics

!!! post "POST /v2/alarm" The "getStatistics` returns statistical information about alarms for one or more clusters, including a count of active critical and warning alarms. This is a quick way to check whether there are active alarms without fetching the full alarm list.

Most importantly this is a quick way to check if there are active alarms on the cluster.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Optional | The numerical ID of the cluster to check. The cluster can also be identified by the cluster name. |
| `cluster_name` | string | Optional | The name of the cluster to check. The cluster ID can also be used to identify the cluster. |
| `cluster_ids` | array | Optional | Multiple clusters can also be requested in one call. |

```
{
    "cluster_id": 1,
    "operation": "getStatistics",
    "request_created": "2017-08-02T05:22:29.566Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `alarm_statistics` | array | A list of `CmonAlarmStatistics` objects containing per-cluster alarm counts. |

**Response example**


```
{
    "alarm_statistics": [
    {
        "class_name": "CmonAlarmStatistics",
        "cluster_id": 1,
        "critical": 2,
        "warning": 1
    } ],
    "request_created": "2017-08-02T05:22:29.566Z",
    "request_id": 3,
    "request_processed": "2017-08-02T05:22:29.615Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### getAlarm

!!! post "POST /v2/alarm" Get information about one particular alarm.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `alarm_id` | integer | Required | The numerical ID of the alarm to return. |
| `cluster_id` | integer | Optional | This field is not required for the call, but if provided it will control which cluster to read. |
| `cluster_name` | string | Optional | This field is not required for the call, but if provided it will control which cluster to read. |


```
{
    "alarm_id": 1,
    "operation": "getAlarm",
    "request_created": "2017-08-02T04:52:33.983Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `alarm` | object | A `CmonAlarm` object with the details of one particular alarm. |

**Response example**


```
{
    "alarm":
    {
        "alarm_id": 1,
        "class_name": "CmonAlarm",
        "cluster_id": 1,
        "component": 3,
        "component_name": "Cluster",
        "counter": 1,
        "created": "2017-08-02T04:50:52.000Z",
        "ignored": 0,
        "measured": 0,
        "message": "System time is drifting between servers and the␣
↪distance between the highest system time and lowest is more than one␣
↪minute.",
        "recommendation": "Synchronize the system clock on the servers␣
↪using e.g NTP or make sure NTP is working. Time drifting may lead to␣
↪unexpected failures or problems, and makes debugging hard.\n\nThe last␣
↪seen host time values (in controller's time-zone):\n192.168.1.127: Aug␣
↪02 06:49:55\n192.168.1.199: Aug 02 06:51:10\n",
        "reported": "2017-08-02T04:50:52.000Z",
        "severity": 1,
        "severity_name": "ALARM_WARNING",
        "title": "System time is drifting",
        "type": 40000,
        "type_name": "ClusterTimeDrift"
    },
    "request_created": "2017-08-02T04:52:33.983Z",
    "request_id": 3,
    "request_processed": "2017-08-02T04:52:34.030Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### getAlarms

!!! post “POST /v2/alarm” Get information about multiple alarms.

### Request


```
{
  "class_name": "CmonRpcRequest",
  "cluster_id": 1,
  "operation": "getAlarms",
  "request_created": "2021-06-03T05:25:49.100Z",
  "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `alarms` | array | A list of `CmonAlarm` objects with the details of the alarms. |

**Response example**


```
{
    "alarms": [
    {
        "alarm_id": 1,
        "class_name": "CmonAlarm",
        "cluster_id": 1,
        "component": 3,
        "component_name": "Cluster",
        "counter": 1,
        "created": "2017-07-27T06:33:38.000Z",
        "ignored": 0,
        "measured": 0,
        "message": "System time is drifting between servers and the␣
↪distance between the highest system time and lowest is more than one␣
↪minute.",
        "recommendation": "Synchronize the system clock on the servers␣
↪using e.g NTP or make sure NTP is working. Time drifting may lead to␣
↪unexpected failures or problems, and makes debugging hard.\n\nThe last␣
↪seen host time values (in controller's time-zone):\n192.168.1.127: Jul␣
↪27 08:32:40\n192.168.1.169: Jul 27 08:33:44\n",
        "reported": "2017-07-27T06:33:38.000Z",
        "severity": 1,
        "severity_name": "ALARM_WARNING",
        "title": "System time is drifting",
        "type": 40000,
        "type_name": "ClusterTimeDrift"
    },
    . . .
    {
        "alarm_id": 3,
        "class_name": "CmonAlarm",
        "cluster_id": 1,
        "component": 0,
        "component_name": "Network",
        "counter": 12,
        "created": "2017-07-27T06:35:56.000Z",
        "host_id": 1,
        "hostname": "192.168.1.169",
        "ignored": 0,
        "measured": 0,
        "message": "Server 192.168.1.169 reports: Host 192.168.1.169 is␣
↪not responding to ping after 3 cycles, the host is most likely␣
```

```
↪unreachable.",
        "recommendation": "Restart failed host, check firewall.",
        "reported": "2017-07-27T06:35:56.000Z",
        "severity": 2,
        "severity_name": "ALARM_CRITICAL",
        "title": "Host is not responding",
        "type": 10006,
        "type_name": "HostUnreachable"
    } ],
    "cluster_id": 1,
    "request_created": "2017-07-27T06:36:03.627Z",
    "request_id": 3,
    "request_processed": "2017-07-27T06:36:03.675Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### ignoreAlarm

!!! post "POST /v2/alarm" Sets the ignored state of an alarm. When `ignore` is `true` the alarm is suppressed; set it to `false` to re-enable a previously suppressed alarm.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `alarm_ic` | integer | Required | The numerical ID of the alarm to update. |
| `ignore` | boolean | Required | Set to `true` to suppress the alarm, or `false` to re-enable a previously suppressed alarm. |

```
{
    "alarm_id": 5,
    "ignore": true,
    "operation": "ignoreAlarm",
    "request_created": "2018-09-25T08:03:17.416Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `alarm` | object | A `CmonAlarm` object reflecting the updated state of the alarm. |

**Response example**


```
{
    "alarm":
    {
        "alarm_id": 5,
        "class_name": "CmonAlarm",
        "cluster_id": 1,
        "component": 0,
        "component_name": "Network",
        "counter": 1,
        "created": "2018-09-25T08:03:17.000Z",
        "host_id": 1,
        "ignored": 1,
        "measured": 0,
        "message": "Server 192.168.0.79 reports: SSH failure to␣
↪192.168.0.79: No route to host.",
        "recommendation": "Please check SSH access to host. The host may␣
↪also be down.",
        "reported": "2018-09-25T08:03:17.000Z",
        "severity": 2,
        "severity_name": "ALARM_CRITICAL",
        "title": "SSH failed",
        "type": 10008,
        "type_name": "HostSshFailed"
    },
    "request_created": "2018-09-25T08:03:17.416Z",
    "request_id": 3,
    "request_processed": "2018-09-25T08:03:17.425Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```


### 4.12.4 Information about Controller's audit log

The `/v2/audit` path can be used to retrieve audit log entries for one or more clusters.

| Operation | Description |
|---|---|
| *getEntries* | Returns audit log entries for one or more clusters. |

### getEntries

!!! post "POST /v2/audit" Returns audit log entries for one or more clusters. Results can be sorted ascending or descending and filtered by cluster ID.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `ascending` | boolean | Required | Sort results ascending if true, descending if false. |
| `cluster_ids` | array | Required | List of cluster IDs to filter audit log entries. |

```
{
    "ascending": true,
    "cluster_ids": [1,3],
    "operation": "getEntries",
    "request_created": "2022-02-26T09:07:48.374Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of audit entries returned. |
| `audit_entries` | array | List of audit log entry objects. |

**Response example**

```
{
  "request_processed": "2022-03-03T18:24:38.304Z",
  "request_status": "Ok",
  "total": 4,
  "audit_entries":
  [
      {
          "class_name": "CmonAuditEntry",
          "client_hostname": "127.0.0.1",
          "cluster_id": 0,
          "entry_type": "users",
          "id": 1000,
          "message_text": "Deleted user 'user.name.123.",
          "report_ts": "2022-03-03T18:11:40.800Z",
          "username": "kedz"
      },
      {
          "class_name": "CmonAuditEntry",
          "client_hostname": "127.0.0.1",
          "cluster_id": 0,
```

```
            "entry_type": "users",
            "id": 1001,
            "message_text": "Group removed 'fancygroup.",
            "report_ts": "2022-03-03T18:11:35.498Z",
            "username": "kedz"
        },
        {
            "class_name": "CmonAuditEntry",
            "client_hostname": "127.0.0.1",
            "cluster_id": 0,
            "entry_type": "users",
            "id": 1002,
            "message_text": "User 'user.name.123' has been removed from␣
↪group 'fancygroup.",
            "report_ts": "2022-03-03T18:11:29.324Z",
            "username": "kedz"
        },
        {
            "class_name": "CmonAuditEntry",
            "client_hostname": "127.0.0.1",
            "cluster_id": 0,
            "entry_type": "users",
            "id": 1003,
            "message_text": "User 'user.name.123' has been added to group
↪'fancygroup.",
            "report_ts": "2022-03-03T18:11:18.168Z",
            "username": "kedz"
        }
    ]
}
```

### 4.12.5 User Authentication

The client authentication methods are available on this path (`/v2/auth`).

There are currently two separate way is available for user authentication, one that uses a public/private key pair and one that uses a password. The key-based authentication has two phases, the password based authentication uses one request.

You may send `with_extended_privileges: true` to this request to get extended user information in case of successful authentication.

The password based authentication is implemented using the "authenticateWithPassword" request:

```
{
    "operation": "authenticateWithPassword",
    "password": "secret",
    "request_created": "2017-06-29T05:46:10.622Z",
    "request_id": 1,
    "user_name": "system"
}
```

The key based authentication starts with the client sending an "authenticate" request to the controller:

```
{
  "operation": "authenticate",
  "request_created": "2017-06-22T07:50:18.117Z",
  "request_id": 1,
  "user_name": "worf"
}
```

The controller replies by sending a challenge to the client:

```
{
  "challenge": "3694b3af-e2f0-e49e-4f32-1549fd824ea9",
  "request_created": "2017-06-22T07:50:18.117Z",
  "request_id": 1,
  "request_processed": "2017-06-22T07:50:18.163Z",
  "request_status": "Ok",
  "request_user_id": 0,
  "user_name": "worf"
}
```

The client then encrypts the challenge string with its private key and sends it back in a new request:

```
{
  "operation": "authenticateresponse",
  "request_created": "2017-06-22T07:50:18.169Z",
  "request_id": 2,
  "signature":
↪"XjiGlQNbq/yN9vzgS5uPjJfKw8DLXGD/uwlw+cec/iC9GJdgs1rZO6GNQzR2Mk6StgopX7gCDQwNVF426v3kGV
}
```

Should the signature match the controller sends back a reply notifying the client that it is now authenticated for the session:

```
{
    "is_superuser": true,
    "request_created": "2017-08-23T09:22:36.857Z",
    "request_id": 2,
    "request_processed": "2017-08-23T09:22:36.906Z",
    "request_status": "Ok",
    "user":
    {
        "class_name": "CmonUser",
        "email_address": "laszlo@severalnines.com",
        "first_name": "Laszlo",
        "groups": [
        {
            "class_name": "CmonGroup",
            "group_id": 1,
            "group_name": "admins"
        } ],
        "last_name": "Pere",
```

(continues on next page)

(continued from previous page)

```
        "title": "",
        "user_id": 3,
        "user_name": "admin"
    }
}
```

- `is_superuser`: Shows if the authenticated user actually has the superuser privileges.
- `request_status`: Shows that authentication was successful and the session now has an authenticated user.
- `request_user_id`: The request that produced this reply had no authenticated user ID, so the reply will not have this field.
- `user`: The details about the user authenticated for the session.

The client can terminate the session when it is no longer needed, on server side using the `logout` operation on auth.

```
{
  "operation": "logout",
  "request_created": "2017-06-22T07:50:18.117Z",
  "request_id": 4,
}
```

| Operation | Description |
|---|---|
| *authenticate* | Initiates key-based authentication by requesting a challenge from the server. |
| *passwordReset* | Resets a user's password via a two-step email-token process. |
| *authenticateResponse* | Completes key-based authentication by submitting the signed challenge. |
| *authenticateWithPassword* | Authenticates a user using their plain-text password. |

### authenticate

!!! post "POST /v2/auth" The first step in the process where the client initiates the authentication by sending the `authenticate` request.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `user_name` | string | Required | The user name of the Cmon user. |

```
{
  "operation": "authenticate",
  "user_name": "worf"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| challenge | string | The string that should be encrypted by the client with the private key. |
| user_name | string | The name of the user which is going to be authenticated. |

**Response example**


```
{
  "challenge": "3694b3af-e2f0-e49e-4f32-1549fd824ea9",
  "request_processed": "2017-06-22T07:50:18.163Z",
  "request_status": "Ok",
  "request_user_id": 0,
  "user_name": "worf"
}
```


The next expected request from client is a 'response' call, with the signature created using the client private key.

### passwordReset

!!! post "POST /v2/auth" Resets a user's password using a two-step process. The first request triggers a password-reset email to be sent to the user's registered address. The second request applies the new password using the token received in that email. Both requests use the same operation name; the first contains only `user_name` (and optionally `base_url`), while the second adds `password_reset_token` and `new_password`.

### Request

**Step 1 - Trigger reset**

| Parameter | Type | Required | Description |
|---|---|---|---|
| user | string | Required | The name of the user for whom the password reset email will be sent. The email address can not be in the request for security reasons, the controller will use the pre-registered email address of the user or if no such email address is registered the call will fail. |
| base | string | Optional | The first part of the URL that will be sent in the email. |


```
{
    "operation": "passwordReset",
    "request_created": "2019-05-31T10:14:30.457Z",
    "request_id": 1,
```


(continues on next page)

(continued from previous page)

```
    "user_name": "system"
}
```

**Step 2 - Apply new password**

| Parameter | Type | Required | Description |
|---|---|---|---|
| user_name | string | Required | The name of the user whose password is being reset. |
| new_password | string | Required | The new password that will be set for the user. |
| password_reset_token | string | Required | The password change token that was received in the email. |

```
{
    "new_password": "newpassword",
    "operation": "passwordReset",
    "request_created": "2019-05-31T10:14:30.540Z",
    "request_id": 1,
    "password_reset_token": "5a018ede8d9e486aadc700fc241e9590",
    "user_name": "system"
}
```

### authenticateResponse

!!! post "POST /v2/auth" This method shall be called by the client once it receives a security challenge (in reply to "authenticate" call) from the server.

In reply backend sends either an error message if authentication failed, or just a success message showing that the authentication is finished and the user is authenticated for the session.

Here is an example for a reply that sent after the authentication succeeded:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| signature | string | Required | A base64 RSA-SHA256 signature using the client's private key of the 'challenge' text. |

```
{
  "operation": "authenticateresponse",
  "request_created": "2017-06-22T07:50:18.169Z",
  "request_id": 2,
  "signature":
↪"XjiGlQNbq/yN9vzgS5uPjJfKw8DLXGD/uwlw+cec/iC9GJdgs1rZO6GNQzR2Mk6StgopX7gCDQwNVF426v3kGV
}
```

### Response

| Field | Type | Description |
|---|---|---|
| user | object | The details about the user authenticated for the session. |

**Response example**


```
{
    "request_created": "2017-06-29T13:25:32.885Z",
    "request_id": 2,
    "request_processed": "2017-06-29T13:25:32.936Z",
    "request_status": "Ok",
    "user":
    {
        "class_name": "CmonUser",
        "email_address": "warrior@ds9.com",
        "first_name": "Worf",
        "groups": [
        {
            "class_name": "CmonGroup",
            "group_id": 4,
            "group_name": "ds9"
        } ],
        "last_login": "2017-06-29T13:25:21.906Z",
        "last_name": "",
        "title": "Lt.",
        "user_id": 12,
        "user_name": "worf"
    }
}
```


### authenticateWithPassword

!!! post “POST /v2/auth” It is possible to authenticate the user using the user’s own password.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| user_ | string | Required | The username of the user to authenticate. |
| password | string | Required | The plain password of the user. Please note that the plain password is not a property of the CmonUser class, it is not stored by the controller, it is only used temporarily. |

```
{
    "operation": "authenticateWithPassword",
    "password": "secret",
    "request_created": "2017-06-29T05:46:10.622Z",
    "request_id": 1,
    "user_name": "system"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `user` | object | The details about the user authenticated for the session. |

**Response example**

```
{
    "request_created": "2017-06-29T13:22:58.592Z",
    "request_id": 1,
    "request_processed": "2017-06-29T13:22:58.645Z",
    "request_status": "Ok",
    "user":
    {
        "class_name": "CmonUser",
        "first_name": "System",
        "groups": [
        {
            "class_name": "CmonGroup",
            "group_id": 1,
            "group_name": "admins"
        } ],
        "last_login": "2017-06-29T13:22:42.624Z",
        "last_name": "User",
        "user_id": 1,
        "user_name": "system"
    }
}
```

And here is a reply that sent about a failed attempt:

```
{
    "error_string": "Wrong username or password.",
    "reply_received": "2017-06-29T05:57:27.128Z",
    "request_created": "2017-06-29T05:57:27.118Z",
    "request_id": 1,
    "request_processed": "2017-06-29T05:57:27.193Z",
    "request_status": "AccessDenied"
}
```

Please note that the reply contains no `request_user_id` field because the session is not authenticated and there is no known user to associate with it. It is also worth noting that the error string does not specify what exactly was the problem with the authentication. Since there is no known user we only send a generic error message. In this case the user was found, but the password did not match, but we are not sending this information with the reply.

### 4.12.6 Information about Backups

The `/v2/backup` path can be used to get information about backups.

Please note that creating and restoring a backup is a job and so is not implemented here.

| Operation | Description |
|---|---|
| *getBackups* | Returns the backups of a cluster |
| *getBackupSet* | Returns the backup set of a backup id, including its increments |
| *getBackupSchedules* | Returns the backup schedules from the controller |
| *scheduleBackup* | Creates or updates a backup schedule for a cluster |
| *deleteBackupRecord* | Deletes a single backup record by ID |
| *deleteBackupList* | Deletes multiple backup records in a single call |
| *snapshotRepositories* | Manages Elasticsearch snapshot repositories |
| *getAllSnapshotRepositories* | Lists all Elasticsearch snapshot repositories |
| *getSnapshotRepositories* | Retrieves snapshot repositories for a specific cluster |
| *deleteSnapshotRepository* | Deletes an existing Elasticsearch snapshot repository |

---

#### getBackups

!!! post "POST /v2/backup" An RPC call that returns the backups of a cluster. For backup locations use `backup_record_version: 2`.

#### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `ascending` | boolean | Optional | Sort order for the results |
| `cluster_id` | integer | Required | The ID of the cluster |
| `backup_record_version` | integer | Optional | Use `2` for backup locations |

```
{
    "ascending": true,
    "cluster_id": 1,
    "operation": "getBackups",
    "backup_record_version": 1,
    "request_created": "2017-06-26T09:07:48.374Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| backup_records | array | List of backup record objects |
| total | integer | Total number of backup records |

**Response example**


```
{
    "backup_records": [
    {
        "backup": [
        {
            "db": "all",
            "files": [
            {
                "class_name": "CmonBackupFile",
                "created": "2017-06-26T09:07:35.000Z",
                "hash": "md5:938bacc9699b362eb1be355eace1281d",
                "path": "pg_dump_2017-06-26_110732.sql.gz",
                "size": 796,
                "type": "full"
            } ],
            "start_time": "2017-06-26T09:07:35.000Z"
        } ],
        "backup_host": "192.168.1.198",
        "chain_up": 0,
        "cid": 1,
        "class_name": "CmonBackupRecord",
        "compressed": true,
        "config":
        {
            "backupDir": "/tmp",
            "backupHost": "192.168.1.198",
            "backupMethod": "",
            "backupToIndividualFiles": false,
            "backup_failover": false,
            "backup_failover_host": "",
            "ccStorage": true,
            "compression": true,
            "createdBy": "pipas",
            "description": "Backup created by s9s-tools.",
            "includeDatabases": "",
            "netcat_port": 9999,
            "origBackupDir": "/tmp",
            "port": null,
            "scheduleId": 0,
            "set_gtid_purged_off": true,
            "storageHost": "",
```

```
            "throttle_rate_iops": 0,
            "throttle_rate_netbw": 0,
            "usePigz": false,
            "wsrep_desync": false,
            "xtrabackupParallellism": 1,
            "xtrabackup_locks": false
        },
        "created": "2017-06-26T09:07:32.000Z",
        "created_by": "",
        "description": "",
        "finished": "2017-06-26T09:07:35.930Z",
        "id": 1,
        "job_id": 2,
        "log_file": "",
        "lsn": 0,
        "method": "pgdump",
        "parent_id": 0,
        "root_dir": "/tmp/BACKUP-1",
        "schedule_id": 0,
        "status": "Completed",
        "storage_host": "192.168.1.127",
        "total_datadir_size": 38959683,
        "verified":
        {
            "message": "",
            "status": "Unverified",
            "verified_time": "1969-12-31T23:59:59.000Z"
        }
    },

    . . .

    ],
    "request_created": "2017-06-26T09:07:48.374Z",
    "request_id": 3,
    "request_processed": "2017-06-26T09:07:48.423Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 2
}
```


### getBackupSet

!!! post "POST /v2/backup" An RPC call that returns the backup set of a backup id, i.e a full backup including its increments are returned. If the backup id belongs to an incremental backup, then all the returned list of chained backups up to the parent (full backup) is returned. If the backup id belongs to a full backup, only the full backup is returned. For backup locations use `backup_record_version: 2`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster |
| `backup_record_version` | integer | Optional | Use `2` for backup locations |

```
{
    "cluster_id": 1,
    "operation": "getBackupSet",
    "backup_record_version": 1,
    "request_created": "2017-06-26T09:07:48.374Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `backup_records` | array | List of backup record objects in the set |
| `total` | integer | Total number of backup records returned |

**Response example**

```
{
    "backup_records": [
    {
        "backup": [
        {
            "db": "all",
            "files": [
            {
                "class_name": "CmonBackupFile",
                "created": "2017-06-26T09:07:35.000Z",
                "hash": "md5:938bacc9699b362eb1be355eace1281d",
                "path": "pg_dump_2017-06-26_110732.sql.gz",
                "size": 796,
                "type": "full"
            } ],
            "start_time": "2017-06-26T09:07:35.000Z"
        } ],
        "backup_host": "192.168.1.198",
        "chain_up": 0,
        "cid": 1,
        "class_name": "CmonBackupRecord",
        "compressed": true,
        "config":
        {
            "backupDir": "/tmp",
```

```
            "backupHost": "192.168.1.198",
            "backupMethod": "",
            "backupToIndividualFiles": false,
            "backup_failover": false,
            "backup_failover_host": "",
            "ccStorage": true,
            "compression": true,
            "createdBy": "pipas",
            "description": "Backup created by s9s-tools.",
            "includeDatabases": "",
            "netcat_port": 9999,
            "origBackupDir": "/tmp",
            "port": null,
            "scheduleId": 0,
            "set_gtid_purged_off": true,
            "storageHost": "",
            "throttle_rate_iops": 0,
            "throttle_rate_netbw": 0,
            "usePigz": false,
            "wsrep_desync": false,
            "xtrabackupParallellism": 1,
            "xtrabackup_locks": false
        },
        "created": "2017-06-26T09:07:32.000Z",
        "created_by": "",
        "description": "",
        "finished": "2017-06-26T09:07:35.930Z",
        "id": 1,
        "job_id": 2,
        "log_file": "",
        "lsn": 0,
        "method": "pgdump",
        "parent_id": 0,
        "root_dir": "/tmp/BACKUP-1",
        "schedule_id": 0,
        "status": "Completed",
        "storage_host": "192.168.1.127",
        "total_datadir_size": 38959683,
        "verified":
        {
            "message": "",
            "status": "Unverified",
            "verified_time": "1969-12-31T23:59:59.000Z"
        }
    },

    . . .

    ],
    "request_created": "2017-06-26T09:07:48.374Z",
```

```
    "request_id": 3,
    "request_processed": "2017-06-26T09:07:48.423Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 2
}
```


---

### getBackupSchedules

!!! post "POST /v2/backup" An RPC call to get the backup schedules from the controller.

Please note that the backup schedules are deprecated. Instead of creating scheduled backups creating of scheduled jobs (that create backups) is advisable. This RPC call was only created for backward compatibility.

### Request


```
{
    "operation": "getBackupSchedules",
    "request_created": "2020-01-29T11:01:25.887Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| backup_schedules | array | List of backup schedule objects |
| total | integer | Total number of backup schedules |

**Response example**


```
{
  "backup_schedules": [ {
      "class_name": "CmonBackupSchedule",
      "cluster_id": 1,
      "enabled": true,
      "id": 1,
      "job": {
        "command": "backup",
        "job_data": {
          "description": "Backup created by s9s-tools."
        },
        "scheduleId": 1
      },
      "lastExec": "1970-01-01T00:00:00.000Z",
      "schedule": "0 12 * * 5"
```

```
    }  ],
  "debug_messages": [ "RPC V2 authenticated user is 'pipas'."  ],
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 3,
  "request_processed": "2020-01-29T11:01:25.893Z",
  "request_status": "Ok",
  "request_user_id": 4,
  "total": 1
}
```

---

### scheduleBackup

!!! post "POST /v2/backup" Creates or updates a backup schedule for a cluster. The `schedule` field uses a standard cron expression to define when the backup job runs. The `job` field within the schedule object defines the backup command and its parameters.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster |
| `schedule` | object | Required | The backup schedule object |
| `schedule.class_name` | string | Required | Must be `CmonBackupSchedule` |
| `schedule.enabled` | boolean | Required | Whether the schedule is enabled |
| `schedule.job` | object | Required | The job definition for the backup |
| `schedule.job.command` | string | Required | The job command, e.g. `backup` |
| `schedule.job.job_data` | object | Optional | Additional job parameters |
| `schedule.schedule` | string | Required | Cron expression for the schedule |

```
{
    "cluster_id": 1,
    "operation": "scheduleBackup",
    "request_created": "2020-01-29T10:57:52.916Z",
    "request_id": 3,
    "schedule":
    {
        "class_name": "CmonBackupSchedule",
        "enabled": true,
        "job":
        {
            "command": "backup",
            "job_data":
            {
                "description": "Backup created by s9s-tools."
            }
```

(continues on next page)

(continued from previous page)

```
        },
        "schedule": "0 12 * * 5"
    }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| schedule | object | The created or updated schedule object |
| schedule.class_name | string | Class name of the schedule |
| schedule.cluster_id | integer | The cluster the schedule belongs to |
| schedule.enabled | boolean | Whether the schedule is enabled |
| schedule.id | integer | The ID of the schedule |
| schedule.job | object | The job definition |
| schedule.lastExec | string | Timestamp of last execution |
| schedule.schedule | string | Cron expression |

**Response example**


```
{
  "debug_messages": [ "RPC V2 authenticated user is 'pipas'."  ],
  "reply_received": "2020-01-29T10:57:52.927Z",
  "request_created": "2020-01-29T10:57:52.916Z",
  "request_id": 3,
  "request_processed": "2020-01-29T10:57:52.926Z",
  "request_status": "Ok",
  "request_user_id": 4,
  "schedule": {
    "class_name": "CmonBackupSchedule",
    "cluster_id": 1,
    "enabled": true,
    "id": 1,
    "job": {
      "command": "backup",
      "job_data": {
        "description": "Backup created by s9s-tools."
      },
      "scheduleId": 1
    },
    "lastExec": null,
    "schedule": "0 12 * * 5"
  }
}
```

### deleteBackupRecord

!!! post "POST /v2/backup" Deletes a single backup record identified by `backup_id`. This removes the backup metadata from the controller database but does not automatically delete the backup files from storage.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `backup_record` | object | Required | Object containing the backup ID to delete |
| `backup_record.backup_id` | integer | Required | The ID of the backup record to delete |

```
{
    "backup_record":
    {
        "backup_id": 2
    },
    "operation": "deleteBackupRecord",
    "request_created": "2019-09-20T12:15:50.682Z",
    "request_id": 3
}
```

---

### deleteBackupList

!!! post "POST /v2/backup" Deletes multiple backup records in a single call by providing a comma-separated list of backup IDs in the `backup_id_list` field. More efficient than calling `deleteBackupRecord` repeatedly.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `backup_id_list` | string | Required | Comma-separated list of backup IDs to delete |

```
{
    "operation": "deleteBackupList",
    "backup_id_list": "2,3,4",
    "request_created": "2024-12-20T12:15:50.682Z",
    "request_id": 3
}
```

---

### snapshotRepositories

!!! post "POST /v2/backup" Manages Elasticsearch snapshot repositories used for backup storage. Supports creating a new snapshot repository (`snapshotRepository`), listing all repositories (`getAllSnapshotRepositories`), retrieving repositories for a specific cluster (`getSnapshotRepositories`), and deleting an existing repository (`deleteSnapshotRepository`).

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `class_name` | string | Required | Must be `CmonRpcRequest` |
| `cluster_id` | integer | Required | The ID of the cluster |
| `repository` | object | Required | The snapshot repository definition |
| `repository.class_name` | string | Required | Must be `CmonBackupSnapshotRepository` |
| `repository.job` | object | Required | The job definition for creating the repository |
| `repository.job.command` | string | Required | The job command, e.g. `backup` |
| `repository.job.description` | string | Optional | Description of the repository job |
| `repository.job.job_data` | object | Required | Job data including repository configuration |
| `repository.job.job_data.credential_id` | integer | Optional | ID of the credentials to use |
| `repository.job.job_data.description` | string | Optional | Description of the backup |
| `repository.job.job_data.s3_bucket` | string | Optional | S3 bucket name for s3-type repositories |
| `repository.job.job_data.snapshot_repository` | string | Required | Name of the snapshot repository |
| `repository.job.job_data.snapshot_repository_type` | string | Required | Type of repository, e.g. `s3`, `fs` |

```
{
  "class_name": "CmonRpcRequest",
  "cluster_id": 108,
  "operation": "snapshotRepository",
  "request_created": "2022-05-24T09:36:57.784Z",
  "request_id": 2,
  "repository": {
    "class_name": "CmonBackupSnapshotRepository",
    "job": {
      "command": "backup",
      "description": "Snapshot repository created by s9s-tools on␣
↪elasticsearch cluster",
      "job_data": {
        "credential_id": 1,
        "description": "Backup created by s9s-tools.",
        "s3_bucket": "elastic-s3-test",
        "snapshot_repository": "cliRepo",
        "snapshot_repository_type": "s3"
```

(continues on next page)

(continued from previous page)

```
        }
      }
    }
}
```

### getAllSnapshotRepositories

!!! post "POST /v2/backup" List all Elasticsearch snapshot repositories.

### Request

```
{
    "operation": "getAllSnapshotRepositories",
    "request_created": "2022-05-26T09:42:31.844Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| total | integer | Total number of snapshot repositories returned |
| snapshot_repositories_list | object | Map of cluster IDs to their snapshot repositories |

**Response example**


```
{
  "controller_id": "1084dffd-be4d-4c03-bca4-00accbcc200b",
  "reply_received": "2022-07-08T10:04:17.952Z",
  "request_created": "2022-07-08T10:04:17.846Z",
  "request_id": 2,
  "request_processed": "2022-07-08T10:04:17.952Z",
  "request_status": "Ok",
  "request_user_id": 5,
  "total": 2,
  "debug_messages": [ "RPC V2 authenticated user is 'ccalvaro'."  ],
  "messages": [ "Snapshot repositories on all clusters has been
↪retrieved"  ],
  "snapshot_repositories_list": {
    "215": {
      "fsRepo": {
        "type": "fs",
        "settings": {
          "location": "/mnt/data/backups/es-snapshot-repositories"
        }
      },
      "test_s3_repo": {
```

```
        "type": "s3",
        "uuid": "QbQ00rxEQJqwnzcgoK_OAg",
        "settings": {
          "bucket": "elastic-s3-test",
          "client": "secondary",
          "region": "eu-west-3"
        }
      }
    },
    "216": {
      "fsRepo": {
        "type": "fs",
        "storage_host": "kvm2",
        "settings": {
          "location": "/mnt/data/backups/es-snapshot-repositories"
        }
      },
      "s3Repo": {
        "type": "s3",
        "storage_host": "s3",
        "uuid": "QbQ00rxEQJqwnzcgoK_OAg",
        "settings": {
          "bucket": "elastic-s3-test",
          "client": "secondary",
          "region": "eu-west-3"
        }
      }
    }
  }
}
```


### getSnapshotRepositories

!!! post "POST /v2/backup" Retrieve Elasticsearch repositories for a specific cluster.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster |


```
{
    "cluster_id": 108,
    "operation": "getSnapshotRepositories",
    "request_created": "2022-05-26T09:42:31.844Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of snapshot repositories returned |
| `snapshot_repositories` | object | Map of repository names to their configuration objects |

**Response example**


```
{
  "controller_id": "1084dffd-be4d-4c03-bca4-00accbcc200b",
  "reply_received": "2022-05-26T09:42:31.898Z",
  "request_created": "2022-05-26T09:42:31.844Z",
  "request_id": 2,
  "request_processed": "2022-05-26T09:42:31.898Z",
  "request_status": "Ok",
  "request_user_id": 5,
  "total": 1,
  "debug_messages": [ "RPC V2 authenticated user is 'ccalvaro'."  ],
  "messages": [ "Snapshot repositories has been retrieved"  ],
  "snapshot_repositories": {
    "sharedFs": {
      "type": "fs",
      "storage_host": "s3",
      "uuid": "yGaLnQU5TFaAX96ZBAgN_Q",
      "settings": {
        "location": "/mnt/data/"
      }
    }
  }
}
```


### deleteSnapshotRepository

!!! post “POST /v2/backup” Delete an existing Elasticsearch repository.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `class_name` | string | Required | Must be `CmonRpcRequest` |
| `cluster_id` | integer | Required | The ID of the cluster |
| `snapshot_repository` | string | Required | Name of the snapshot repository to delete |


```
{
 "class_name": "CmonRpcRequest",
```


(continues on next page)

(continued from previous page)

```
  "cluster_id": 108,
  "operation": "deleteSnapshotRepository",
  "request_created": "2022-05-26T09:17:13.385Z",
  "request_id": 2,
  "snapshot_repository": "s3_cli_repo"
}
```

### 4.12.7 Certificate Authority (CA)

With these API calls you can list and manage your certificates managed by ClusterControl. These methods can be found on path `/v2/ca`.

| Operation | Description |
|---|---|
| *listCerts* | List CA, server, and client certificates |
| *certInfo* | Get details about a specific certificate |
| *create* | Create a signed (self or CA signed) certificate |
| *revoke* | Mark a certificate as revoked |
| *delete* | Delete certificate and private key files |
| *move* | Move a certificate within the ClusterControl CA storage |
| *crl* | Create a certificate revocation list signed by a specified CA |
| *importlocal* | Import a certificate from the controller host |

#### listCerts

!!! post "POST /v2/ca" A method to list the (CA, server and client) certificates.

#### Request

```
{ "operation": "listCerts" }
```

#### Response

**Response example**

```
{
    "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
    "is_superuser": true,
    "request_processed": "2023-02-16T14:28:37.281Z",
    "request_status": "Ok",
    "total": 1,
    "data":
    [
        {
            "certfile": "galera/cluster_1/server_ca.crt",
            "id": 1,
            "isCA": true,
            "isClient": true,
            "isServer": true,
```

```
            "issued": 1620992185,
            "issuerId": 0,
            "keybits": 2048,
            "keyfile": "galera/cluster_1/server_ca.key",
            "serialNumber": 1,
            "status": "Issued",
            "validFrom": 1620905785,
            "validUntil": 1936352185,
            "inUseBy":
            {
                "hosts": [],
                "clusters": []
            },
            "subjectName":
            {
               "CN": "ClusterControl_AutoGenerated_CA_Certificate",
               "basicConstraints": "ca",
               "serialNumber": "1",
               "extendedKeyUsage":
               [
                   "ClientAuth",
                   "ServerAuth",
                   "OCSPSigning"
               ],
               "keyUsage":
               [
                   "DigitalSignature",
                   "NonRepudiation",
                   "KeyEncipherment",
                   "KeyAgreement",
                   "KeyCertificateSign",
                   "CRLSign"
               ]
           }
       }
  ]
}
```


### certInfo

!!! post “POST /v2/ca” A method to give details about a certificate.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `id` | integer | Required | The ID of the certificate to retrieve details for |

```
{
  "operation": "certInfo",
  "id": 2
}
```

### Response

**Response example**


```
{
    "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
    "is_superuser": true,
    "request_processed": "2023-02-16T14:57:24.882Z",
    "request_status": "Ok",
    "total": 1,
    "data":
    {
        "certfile": "galera/cluster_1/server.crt",
        "id": 2,
        "isCA": false,
        "isClient": true,
        "isServer": true,
        "issued": 1620992185,
        "issuerId": 1,
        "keybits": 2048,
        "keyfile": "galera/cluster_1/server.key",
        "serialNumber": 2,
        "status": "Issued",
        "validFrom": 1620905785,
        "validUntil": 1936352185,
        "inUseBy":
        {
            "hosts": [],
            "clusters": []
        },
        "subjectName":
        {
            "CN": "MySQL_Server_Cmon_Auto_Generated_Server_Certificate",
            "description": "Generated by ClusterControl",
            "serialNumber": "2",
            "extendedKeyUsage":
            [
                "ClientAuth",
                "ServerAuth"
            ],
            "keyUsage":
            [
                "DigitalSignature",
                "NonRepudiation",
                "KeyEncipherment",
```

```
                "KeyAgreement"
            ]
        }
    }
}
```

### create

!!! post "POST /v2/ca" Creates a signed (self or CA signed) certificate.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `type` | string | Required | The type of certificate to create (e.g. `ca`) |
| `name` | string | Required | The name/path for the certificate |
| `validity` | integer | Required | Validity period in days |
| `data` | object | Required | Certificate data object containing subject details |
| `data.keybits` | integer | Required | Key size in bits |
| `data.CN` | string | Required | Common Name for the certificate |
| `data.description` | string | Optional | Description of the certificate |
| `data.emailAddress` | string | Optional | Email address to associate with the certificate |

```
{
  "operation":"create",
  "type":"ca",
  "name":"exampleCa1",
  "validity":3650,
  "data":{
    "keybits":4096,
    "CN":"My CA certificate.",
    "description":"This is an example CA certificate.",
    "emailAddress":"john.doe@severalnines.com"
  }
}
```

### Response

**Response example**

```
{
    "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
    "is_superuser": true,
    "request_processed": "2023-02-23T08:55:21.030Z",
    "request_status": "Ok",
    "total": 1,
    "data":
    {
```

```
        "certfile": "exampleCa1.crt",
        "id": 228,
        "isCA": true,
        "isClient": true,
        "isServer": true,
        "issued": 1677142521,
        "issuerId": 0,
        "keybits": 4096,
        "keyfile": "exampleCa1.key",
        "requesterUserId": 68,
        "serialNumber": 228,
        "status": "Issued",
        "validFrom": 1677052521,
        "validUntil": 1992498921,
        "inUseBy":
        {
            "hosts": [],
            "clusters": []
        },
        "subjectName":
        {
            "CN": "My CA certificate.",
            "basicConstraints": "ca",
            "description": "This is an example CA certificate.",
            "emailAddress": "john.doe@severalnines.com",
            "serialNumber": "228",
            "extendedKeyUsage":
            [
                "ClientAuth",
                "ServerAuth",
                "OCSPSigning"
            ],
            "keyUsage":
            [
                "DigitalSignature",
                "NonRepudiation",
                "KeyEncipherment",
                "KeyAgreement",
                "KeyCertificateSign",
                "CRLSign"
            ]
        }
    }
}
```

### revoke

!!! post "POST /v2/ca" Marks the specified certificate as revoked. You might need to regenerate the CRL list of the issuer (CA) certificate after this operation.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| id | integer | Required | The ID of the certificate to revoke |

```
{
   "operation":"revoke",
   "id": 227
}
```

### Response

**Response example**


```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-02-23T14:12:39.292Z",
  "request_status": "Ok",
  "total": 1,
  "data":
  {
      "certfile": "replication/cluster_117/client.crt",
      "id": 227,
      "isCA": false,
      "isClient": true,
      "isServer": false,
      "issued": 1674744928,
      "issuerId": 225,
      "keybits": 2048,
      "keyfile": "replication/cluster_117/client.key",
      "revoked": 1677161559,
      "revokerUserId": 68,
      "serialNumber": 227,
      "status": "Revoked",
      "validFrom": 1674654928,
      "validUntil": 1990101328,
      "inUseBy":
      {
          "hosts": [],
          "clusters": []
      },
      "subjectName":
      {
```

```
            "CN": "MySQL_Server_Cmon_Auto_Generated_Client_Certificate",
            "description": "Generated by ClusterControl",
            "serialNumber": "227",
            "extendedKeyUsage":
            [
                "ClientAuth"
            ],
            "keyUsage":
            [
                "DigitalSignature",
                "NonRepudiation",
                "KeyEncipherment",
                "KeyAgreement"
            ]
        }
    }
}
```

### delete

!!! post "POST /v2/ca" Deletes the certificate (cert and private key) files. Please note that this is an unusual and dangerous operation, please use revoke instead of this.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| id | integer | Required | The ID of the certificate to delete |

```
{
    "operation":"delete",
    "id": 1001
}
```

### Response

**Response example**

```
{
    "request_status": "Ok"
}
```

### move

!!! post "POST /v2/ca" Moves a certificate within the ClusterControl CA storage.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| id | integer | Required | The ID of the certificate to move |
| name | string | Required | The new name/path for the certificate |

```
{
   "operation":"move",
   "id": 228,
   "name": "examples/demoCA/cert01"
}
```

### Response

**Response example**


```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-02-23T13:32:09.206Z",
  "request_status": "Ok",
  "total": 1,
  "data":
  {
      "certfile": "examples/demoCA/cert01.crt",
      "id": 228,
      "isCA": true,
      "isClient": true,
      "isServer": true,
      "issued": 1677142521,
      "issuerId": 0,
      "keybits": 4096,
      "keyfile": "examples/demoCA/cert01.key",
      "requesterUserId": 68,
      "serialNumber": 228,
      "status": "Issued",
      "validFrom": 1677052521,
      "validUntil": 1992498921,
      "inUseBy":
      {
          "hosts": [],
          "clusters": []
      },
      "subjectName":
      {
          "CN": "My CA certificate.",
          "basicConstraints": "ca",
          "description": "This is an example CA certificate.",
          "emailAddress": "john.doe@severalnines.com",
```

```
            "serialNumber": "228",
            "extendedKeyUsage":
            [
                "ClientAuth",
                "ServerAuth",
                "OCSPSigning"
            ],
            "keyUsage":
            [
                "DigitalSignature",
                "NonRepudiation",
                "KeyEncipherment",
                "KeyAgreement",
                "KeyCertificateSign",
                "CRLSign"
            ]
        }
    }
}
```

### crl

!!! post "POST /v2/ca" Creates a certificate revokation list (CRL) signed by the specified CA.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| id | integer | Required | The ID of the CA certificate to use for signing the CRL |

```
{
    "operation":"crl",
    "id": 229
}
```

### Response

**Response example**

```
{
    "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
    "is_superuser": true,
    "request_processed": "2023-02-23T13:29:00.685Z",
    "request_status": "Ok",
    "total": 1,
    "data":
    {
        "certfile": "replication/cluster_118/server_ca.crt",
```

```
        "crlfile": "replication/cluster_118/server_ca.crl"
    }
}
```

### importlocal

!!! post "POST /v2/ca" Imports a certificate (.crt and .key file) from controller host. CA is optional in the request, as not needed for example in case of the certs are self-signed or CA certs.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| cert_file | string | Required | Path to the certificate file on the controller host |
| key_file | string | Required | Path to the private key file on the controller host |
| ca_file | string | Optional | Path to the CA certificate file on the controller host |
| name | string | Required | Name/path to store the imported certificate under |
| name_ca | string | Optional | Name/path to store the imported CA certificate under |

```
{
   "operation":"importlocal",
   "cert_file": "/tmp/TestCLIENT.crt",
   "key_file": "/tmp/TestCLIENT.key",
   "ca_file": "/tmp/TestingCA007.crt",
   "name": "testing/import/testClient",
   "name_ca": "testing/import/testCA"
}
```

### Response

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-02-23T14:20:39.058Z",
  "request_status": "Ok",
  "total": 1,
  "data":
  {
      "certfile": "testing/import/testClient.crt",
      "id": 236,
      "isCA": false,
      "isClient": true,
      "isServer": false,
      "issued": 1677162039,
      "issuerId": 235,
      "keybits": 2048,
```

```
        "keyfile": "testing/import/testClient.key",
        "serialNumber": 234,
        "status": "Issued",
        "validFrom": 1677071895,
        "validUntil": 1992518295,
        "wasCaKnown": false,
        "inUseBy":
        {
            "hosts": [],
            "clusters": []
        },
        "subjectName":
        {
            "description": "fdsfdsf",
            "serialNumber": "234",
            "extendedKeyUsage":
            [
                "ClientAuth"
            ],
            "keyUsage":
            [
                "DigitalSignature",
                "NonRepudiation",
                "KeyEncipherment",
                "KeyAgreement"
            ]
        }
    }
}
```


### 4.12.8 Cloud credentials API

The `/v2/cloud` path provides calls to manage the cloud-credentials in cmon.

| Operation | Description |
|---|---|
| *proxy* | Proxies a request to the cmon-cloud service and returns the raw response. |
| *verifyCredentials* | Verifies that a set of cloud provider credentials are valid. |
| *listCredentials* | Returns all stored cloud provider credentials, grouped by provider. |
| *getCredentials* | Returns a specific set of stored cloud credentials identified by `id` and `provider`. |
| *addCredentials* | Stores a new set of cloud provider credentials in the controller. |
| *updateCredentials* | Updates an existing set of cloud provider credentials identified by `id`. |
| *removeCredentials* | Removes a stored set of cloud provider credentials identified by `id` and `provider`. |

### proxy

!!! post "POST /v2/cloud" Proxies a request to the cmon-cloud service and returns the raw response. Use this to interact with cloud provider APIs through the controller without exposing credentials directly to the client.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `uri` | string | Required | The URI path to proxy to the cmon-cloud service. |
| `method` | string | Required | The HTTP method to use for the proxied request. |
| `provider` | string | Required | The cloud provider (e.g. `aws`). |
| `credentials_id` | integer | Required | The ID of the stored credentials to use for the request. |
| `body` | string | Required | The request body to forward. |

```
{
  "operation":"proxy",
  "uri":"/aws/vm/sizes?type=all&mem_size_mb=2048",
  "method":"GET",
  "provider":"aws",
  "credentials_id":1,
  "body": "",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `reply_received` | string | Timestamp when the reply was received. |
| `response` | object | The raw response from the cmon-cloud service. |
| `response.reason_phrase` | string | The HTTP reason phrase from the proxied response. |
| `response.status_code` | integer | The HTTP status code from the proxied response. |
| `response.headers` | object | The HTTP headers from the proxied response. |
| `response.json` | array | The JSON body of the proxied response. |

**Response example**

```
{
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 10,
  "request_processed": "2020-01-29T11:01:25.893Z",
  "request_status": "Ok",
  "response":{
    "reason_phrase":"OK",
    "status_code":200,
    "headers":{
      "content-length":"1016",
```

```
      "content-type":"application\/json; charset=UTF-8",
      "date":"Tue, 30 Nov 2021 12:40:43 GMT"
    },
    "json":["t3.small","t3.medium","t3.large"],
  }
}
```


### verifyCredentials

!!! post "POST /v2/cloud" Verifies that a set of cloud provider credentials are valid by issuing a test request (such as listing storage buckets) through the cmon-cloud service.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `provider` | string | Required | The cloud provider (e.g. `aws`). |
| `credentials` | object | Required | The credentials to verify. |
| `credentials.access_key_id` | string | Required | The cloud provider access key ID. |
| `credentials.access_key_secret` | string | Required | The cloud provider access key secret. |
| `credentials.access_key_region` | string | Required | The cloud provider region. |


```
{
  "operation": "verifyCredentials",
  "provider":"aws",
  "credentials":
  {
    "access_key_id":"DEADBAADDEADBAAD0x0x0x0x",
    "access_key_secret":"SECRET1SECRET0",
    "access_key_region":"ap-southeast-2"
  }
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `added_id` | integer | The ID assigned to the verified credentials. |

**Response example**


```
{
  "added_id": 12313,
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 3,
  "request_processed": "2020-01-29T11:01:25.893Z",
```

```
  "request_status": "Ok",
  "request_user_id": 4
}
```

### listCredentials

!!! post "POST /v2/cloud" Returns all stored cloud provider credentials, grouped by provider. Ensure this call is made over a secure (TLS) connection as credential secrets are included in the response.

### Request

```
{
  "operation": "listCredentials"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `result` | object | Credentials grouped by provider name. |
| `result.aws` | array | List of stored AWS credential entries. |
| `result.gce` | array | List of stored GCE credential entries. |

**Response example**

```
{
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 10,
  "request_processed": "2020-01-29T11:01:25.893Z",
  "request_status": "Ok",
  "request_user_id": 4,
  "result":{
    "aws":[
      {
        "name":"MyAWS",
        "comment":"this is a comment",
        "id":1,
        "credentials":
        {
          "access_key_id":"DEADBAADDEADBAAD0x0x0x0x",
          "access_key_region":"ap-southeast-2",
          "access_key_secret":"SECRET1SECRET0"
        }
      }
    ],
    "gce":[
      {
```

```
          "name":"GCE1",
          "comment":"DDDD",
          "id":2,
          "credentials":
          {
            "auth_provider_x509_cert_url":
↪"https:\/\/www.googleapis.com\/oauth2\/v1\/certs",
            "auth_uri":"https:\/\/accounts.google.com\/o\/oauth2\/auth",
            "client_email":
↪"490013373999-compute@developer.gserviceaccount.com",
            "client_id":"100000568876229481111",
            "client_x509_cert_url":
↪"https:\/\/www.googleapis.com\/robot\/v1\/metadata\/x509\/1234-compute%40developer.gse
↪
            "private_key":"-----BEGIN PRIVATE␣
↪KEY-----\nFDSFSFDSFSDFDSFDSFDS\n-----END PRIVATE KEY-----\n",
            "private_key_id":"b0a4df1c54fff4410c33c9bca57bd0fb01679000",
            "project_id":"brave-night-121210",
            "token_uri":"https:\/\/accounts.google.com\/o\/oauth2\/token",
            "type":"service_account"
          }
        }
      ]
    }
}
```


### getCredentials

!!! post "POST /v2/cloud" Returns a specific set of stored cloud credentials identified by `id` and `provider`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `id` | integer | Required | The ID of the stored credentials to retrieve. |
| `provider` | string | Required | The cloud provider (e.g. `aws`). |


```
{
  "operation": "getCredentials",
  "id": 10,
  "provider": "aws"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `data` | object | The stored credential entry. |
| `data.name` | string | The name assigned to the credential set. |
| `data.comment` | string | An optional comment for the credential set. |
| `data.id` | integer | The ID of the credential set. |
| `data.credentials` | object | The credential values for the provider. |

**Response example**


```
{
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 10,
  "request_processed": "2020-01-29T11:01:25.893Z",
  "request_status": "Ok",
  "request_user_id": 4,
  "data":{
    "name":"MyAWS",
    "comment":"this is a comment",
    "id":1,
    "credentials":
    {
      "access_key_id":"DEADBAADDEADBAAD0x0x0x0x",
      "access_key_region":"ap-southeast-2",
      "access_key_secret":"SECRET1SECRET0"
    }
  }
}
```


### addCredentials

!!! post “POST /v2/cloud” Stores a new set of cloud provider credentials in the controller. Returns the assigned `added_id` on success.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `provider` | string | Required | The cloud provider (e.g. `aws`). |
| `name` | string | Required | A name to assign to this credential set. |
| `credentials` | object | Required | The credential values for the provider. |
| `credentials.access_key_id` | string | Required | The cloud provider access key ID. |
| `credentials.access_key_secret` | string | Required | The cloud provider access key secret. |
| `credentials.access_key_region` | string | Required | The cloud provider region. |
| `comment` | string | Optional | A comment or description for this credential set. |

```
{
  "operation": "addCredentials",
  "provider":"aws",
  "name":"MyAWS",
  "credentials":
  {
    "access_key_id":"DEADBAADDEADBAAD0x0x0x0x",
    "access_key_secret":"SECRET1SECRET0",
    "access_key_region":"ap-southeast-2"
  },
  "comment":"this is a comment"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `added_id` | integer | The ID assigned to the newly stored credential set. |

**Response example**

```
{
  "added_id": 12313,
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 3,
  "request_processed": "2020-01-29T11:01:25.893Z",
  "request_status": "Ok",
```

```
  "request_user_id": 4
}
```

### updateCredentials

!!! post "POST /v2/cloud" Updates an existing set of cloud provider credentials identified by `id`. All credential fields are replaced with the supplied values.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `id` | integer | Required | The ID of the credential set to update. |
| `provider` | string | Required | The cloud provider (e.g. `aws`). |
| `name` | string | Required | The name to assign to this credential set. |
| `credentials` | object | Required | The updated credential values for the provider. |
| `credentials.access_key_id` | string | Required | The cloud provider access key ID. |
| `credentials.access_key_secret` | string | Required | The cloud provider access key secret. |
| `credentials.access_key_region` | string | Required | The cloud provider region. |
| `comment` | string | Optional | A comment or description for this credential set. |

```
{
  "operation": "updateCredentials",
  "id": 12313,
  "provider":"aws",
  "name":"MyAWS",
  "credentials":
  {
    "access_key_id":"DEADBAADDEADBAAD0x0x0x0x",
    "access_key_secret":"SECRET1SECRET0",
    "access_key_region":"ap-southeast-2"
  },
  "comment":"this is a comment"
}
```

### Response

**Response example**


```
{
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 4,
  "request_processed": "2020-01-29T11:01:25.893Z",
  "request_status": "Ok",
  "request_user_id": 4
}
```


### removeCredentials

!!! post "POST /v2/cloud" Removes a stored set of cloud provider credentials identified by `id` and `provider`. Clusters or resources that depend on these credentials will no longer be able to access them.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `id` | integer | Required | The ID of the credential set to remove. |
| `provider` | string | Required | The cloud provider (e.g. `aws`). |


```
{
  "operation": "removeCredentials",
  "id": 12313,
  "provider":"aws",
}
```


### Response

**Response example**


```
{
  "reply_received": "2020-01-29T11:01:25.893Z",
  "request_created": "2020-01-29T11:01:25.887Z",
  "request_id": 4,
  "request_processed": "2020-01-29T11:01:25.893Z",
  "request_status": "Ok",
  "request_user_id": 4
}
```


### 4.12.9 Cluster Information and Manipulation

Cluster information and cluster wide operations can be found on the path `/v2/clusters`. Information about hosts, alarms, license, statistics can also be sent back for convenience.

| Operation | Description |
|---|---|
| *getConfig* | Retrieve cluster configuration settings. |
| *setConfig* | Update cluster configuration settings. |
| *getClusterInfo* | Read information about a single cluster. |
| *getAllClusterInfo* | Get information about one or more clusters. |
| *getAllClusterInfoWithAllPool* | Get cluster information aggregated from all controllers in the pool. |

#### getConfig

!!! post "POST /v2/clusters" This is the "getConfig" call for clusters. Please note that there is an other "getConfig" call for nodes and these two should not be mixed up. There are some similarities since the controller is itself a node, but this is an entirely different call because it is on a different path.

Here is how a getConfig call should look like:

#### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster to get configuration for. |


```
{
    "cluster_id": 1,
    "operation": "getConfig",
    "request_created": "2019-09-25T06:53:50.012Z",
    "request_id": 3
}
```



```
{
    "cluster_id": 1,
    "operation": "getConfigGrouped",
    "request_created": "2019-09-25T06:53:50.012Z",
```


(continues on next page)

(continued from previous page)

```
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `configuration` | object | Holds the cluster configuration. |
| `configuration.values` | array | List of configuration items showing possible settings. |

**Response example**


```
{
  "configuration": {
    "values": [
      {
        "current_value": "",
        "default_value": null,
        "description": "An arbitrary identifier string of this controller␣
↪instance.",
        "keys": [ "controller_id"  ],
        "name": "controller_id",
        "sql_keys": [ "CONTROLLER_ID"  ]
      },

      . . .

      {
        "current_value": null,
        "default_value": null,
        "description": "Specify a lock file and if present on a node, ␣
↪the node will not recover. It is the responsibilty of the administrator␣
↪to create/remove the file.",
        "keys": [ "node_recovery_lock_file"  ],
        "name": "node_recovery_lock_file",
        "sql_keys": [ "NODE_RECOVERY_LOCK_FILE"  ]
      }
    ]
  },
  "debug_messages": [ "RPC V2 authenticated user is 'pipas'."  ],
  "reply_received": "2019-09-25T12:21:30.909Z",
  "request_created": "2019-09-25T12:21:30.904Z",
  "request_id": 3,
  "request_processed": "2019-09-25T12:21:30.944Z",
  "request_status": "Ok",
  "request_user_id": 4
}
```

As it is shown the "configuration" field holds the cluster configuration and inside it there is a list "values" showing the possible settings. The structure of the data here might be enhanced later, we might want to send back more information.

### setConfig

!!! post "POST /v2/clusters" This is the "setConfig" call for clusters. There is a similar call for nodes, but that has slightly different requests and replies.

Here is how a setConfig request should look like:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster to set configuration for. |
| `configuration` | array | Required | List of configuration items with names and values to change. |

```
{
    "cluster_id": 1,
    "configuration": [
    {
        "name": "host_stats_collection_interval",
        "value": "60"
    } ],
    "operation": "setConfig",
    "request_created": "2019-09-25T07:53:11.644Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
  "debug_messages": [ "RPC V2 authenticated user is 'pipas'."  ],
  "reply_received": "2019-09-25T12:19:52.392Z",
  "request_created": "2019-09-25T12:19:52.387Z",
  "request_id": 3,
  "request_processed": "2019-09-25T12:19:52.430Z",
  "request_status": "Ok",
  "request_user_id": 4
}
```

Here the changes are passed in a list with names and values. The names can be either any of the keys or any of the sql_keys for the configuration item as they are returned for the getConfig call. So one can issue a getConfig call, study, find the key to be changed and send back for the setConfig request.

One needs write access on the configuration (same as the write access on the cluster) to change the configuration.

### getClusterInfo

!!! post "POST /v2/clusters" This request is for reading information about one cluster. The cluster can be identified by the cluster ID or the cluster name.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Optional | The numerical ID of the cluster to get. The cluster can also be identified by its name. |
| `cluster_name` | string | Optional | The name of the cluster to get. The cluster can also be identified by its numerical ID. |
| `with_hosts` | boolean | Optional | Send also the list of hosts. |
| `with_sheet_inf` | boolean | Optional | Send also the information from the sheet manager. |
| `with_databases` | boolean | Optional | Send also the databases we found on the cluster. |
| `with_license_cl` | boolean | Optional | Send also the license info. |

### getAllClusterInfo

!!! post "POST /v2/clusters" This request can be used to get information about one or more clusters. The clusters can be identified by their cluster ID passed as a list of integers in the request.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| cluster_ids | array | Optional | An optional list of cluster IDs represented as integers to control which clusters should be detailed in the reply. If this argument is not provided the controller will return information about clusters that the user can see. |
| with_hos | boole | Optional | Send also the list of hosts. |
| with_cor | boole | Optional | Send also the information about the containers/VM-s. |
| with_she | boole | Optional | Send also the information from the sheet manager. |
| with_dat | boole | Optional | Send also the databases we found on the cluster. |
| with_lic | boole | Optional | Send also the license info. |
| with_tag | array | Optional | The request may contain this field to limit the list of the returned clusters to those that have at least one of the tags passed here. |
| without_tags | array | Optional | The request may contain this field to limit the list of the returned users to those that has none of the tags passed here. |

```
{
   "operation": "getAllClusterInfo",
   "request_created": "2017-06-20T06:19:58.451Z",
   "with_hosts": true,
   "with_sheet_info": true
}
```

### Response

| Field | Type | Description |
|---|---|---|
| clusters | array | A list of clusters with all details categorised under CmonClusterInfo objects. |
| total | integer | The total number of clusters managed by the controller. |

**Response example**

```
{
    "clusters": [
    {
        "alarm_statistics":
        {
            "class_name": "CmonAlarmStatistics",
            "cluster_id": 1,
            "critical": 2,
            "warning": 1
        },
```

```
        "class_name": "CmonClusterInfo",
        "cluster_auto_recovery": true,
        "cluster_id": 1,
        "cluster_name": "ft_postgresql_59238",
        "cluster_type": "POSTGRESQL_SINGLE",
        "configuration_file": "/tmp/cmon_1.cnf",
        "group_owner":
        {
            "class_name": "CmonGroup",
            "group_id": 4,
            "group_name": "testgroup"
        },
        "hosts": [
        {
            "class_name": "CmonPostgreSqlHost",
            "clusterid": 1,
            "configfile": [ "/etc/postgresql/9.6/main/postgresql.conf" ],
            "connected": true,
            "container": true,
            "data_directory": "/var/lib/postgresql/9.6/main",
            "datadir": "/var/lib/postgresql/9.6/main",
            "description": "",
            "distribution":
            {
                "codename": "xenial",
                "name": "ubuntu",
                "release": "16.04",
                "type": "debian"
            },
            "hostId": 1,
            "hostname": "192.168.1.177",
            "hoststatus": "CmonHostOnline",
            "hot_standby": true,
            "ip": "192.168.1.177",
            "lastseen": 1501216153,
            "logfile": "/var/log/postgresql/postgresql-9.6-main.log[0m",
            "maintenance_mode_active": false,
            "message": "Up and running",
            "nodetype": "postgres",
            "pid": 5248,
            "pidfile": "/var/run/postgresql/9.6-main.pid",
            "pingstatus": 0,
            "pingtime": 0,
            "port": 8089,
            "readonly": false,
            "received_location": "0/1520838",
            "replay_location": "0/1520838",
            "role": "master",
            "sshfailcount": 0,
            "ssl_certs":
```

```
            {
                "server":
                {
                    "ca": "",
                    "id": 0,
                    "key": "/etc/ssl/private/ssl-cert-snakeoil.key",
                    "path": "/etc/ssl/certs/ssl-cert-snakeoil.pem"
                }
            },
            "timestamp": 1501216153,
            "unique_id": 1,
            "uptime": 220,
            "version": "9.6.3",
            "wallclock": 1501216175,
            "wallclocktimestamp": 1501216110
        },
        {
            "class_name": "CmonHost",
            "clusterid": 1,
            "configfile": "/tmp/cmon_1.cnf",
            "connected": true,
            "container": false,
            "distribution":
            {
                "codename": "trusty",
                "name": "ubuntu",
                "release": "14.04",
                "type": "debian"
            },
            "hostId": 2,
            "hostname": "192.168.1.127",
            "hoststatus": "CmonHostOnline",
            "ip": "192.168.1.127",
            "lastseen": 1501216153,
            "logfile": "/tmp/cmon_1.log",
            "maintenance_mode_active": false,
            "message": "Up and running",
            "nodetype": "controller",
            "pid": 59196,
            "pingstatus": 0,
            "pingtime": 0,
            "port": 9555,
            "role": "controller",
            "timestamp": 1501216153,
            "unique_id": 2,
            "uptime": 413,
            "version": "1.4.3",
            "wallclock": 1501216110,
            "wallclocktimestamp": 1501216110
        } ],
```

```
        "info":
        {
            "cluster.status": 2,
            "cluster.statustext": "Cluster started.",
            "cmon.domainname": "",
            "cmon.hostname": "t7500",
            "cmon.running": true,
            "cmon.starttime": 1501215924,
            "cmon.uptime": 227,
            "conf.backup_retention": 31,
            "conf.clusterid": 1,
            "conf.clustername": "ft_postgresql_59238",
            "conf.clustertype": 5,
            "conf.configfile": "/tmp/cmon_1.cnf",
            "conf.hostname": "192.168.1.127",
            "conf.os": "debian",
            "conf.statustext": "Configuration loaded.",
            "host.1.connected": true,
            "host.1.cpu_io_wait_percent": 0,
            "host.1.cpu_steal_percent": 0,
            "host.1.cpu_usage_percent": 7.92751,
            "host.1.cpucores": 16,
            "host.1.cpuinfo": [
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 4,
                "cpumaxmhz": 2.268e+06,
                "cpumhz": 1600,
                "cpumodel": "Intel(R) Xeon(R) CPU           L5520  @␣
↪2.27GHz",
                "cputemp": 54.5,
                "hostid": 1,
                "physical_cpu_id": 0,
                "siblings": 8,
                "vendor": "GenuineIntel"
            },
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 4,
                "cpumaxmhz": 2.268e+06,
                "cpumhz": 1600,
                "cpumodel": "Intel(R) Xeon(R) CPU           L5520  @␣
↪2.27GHz",
                "cputemp": 54.5,
                "hostid": 1,
                "physical_cpu_id": 1,
                "siblings": 8,
                "vendor": "GenuineIntel"
            } ],
            "host.1.cpumaxmhz": 2268,
```

```
            "host.1.cpumhz": 1600,
            "host.1.cpumodel": "Intel(R) Xeon(R) CPU           L5520  @␣
↪2.27GHz",
            "host.1.cputemp": 54.5,
            "host.1.devices": [ "/dev/mapper/core1--vg-root" ],
            "host.1.free_disk_bytes": 156237512704,
            "host.1.hostname": "192.168.1.177",
            "host.1.interfaces": [ "eth0" ],
            "host.1.ip": "192.168.1.177",
            "host.1.membuffer": 0,
            "host.1.memcached": 3,
            "host.1.memfree": 16450528,
            "host.1.memtotal": 16777216,
            "host.1.network_interfaces": [
            {
                "interface_name": "eth0",
                "rx_bytes_per_sec": 10438.1,
                "tx_bytes_per_sec": 15937.2
            } ],
            "host.1.pingdelay": -1,
            "host.1.pingstatustext": "Creating ICMP socket (to ping
↪'192.168.1.177') failed: Operation not permitted.",
            "host.1.port": 8089,
            "host.1.rx_bytes_per_second": 10438.1,
            "host.1.swapfree": 0,
            "host.1.swaptotal": 0,
            "host.1.total_disk_bytes": 208033853440,
            "host.1.tx_bytes_per_second": 15937.2,
            "host.1.uptime": 390,
            "host.1.wallclock": 1501216175,
            "host.1.wallclocksampled": 1501216110,
            "host.2.class_name": "controller",
            "host.2.connected": true,
            "host.2.cpu_io_wait_percent": 4.14259,
            "host.2.cpu_steal_percent": 0,
            "host.2.cpu_usage_percent": 178.599,
            "host.2.cpucores": 24,
            "host.2.cpuinfo": [
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 6,
                "cpumaxmhz": 2.661e+06,
                "cpumhz": 2660,
                "cpumodel": "Intel(R) Xeon(R) CPU           X5650  @␣
↪2.67GHz",
                "cputemp": 0,
                "hostid": 2,
                "physical_cpu_id": 0,
                "siblings": 12,
                "vendor": "GenuineIntel"
```

```
            },
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 6,
                "cpumaxmhz": 2.661e+06,
                "cpumhz": 2661,
                "cpumodel": "Intel(R) Xeon(R) CPU           X5650  @␣
↪2.67GHz",
                "cputemp": 0,
                "hostid": 2,
                "physical_cpu_id": 1,
                "siblings": 12,
                "vendor": "GenuineIntel"
            } ],
            "host.2.cpumaxmhz": 2661,
            "host.2.cpumhz": 2660,
            "host.2.cpumodel": "Intel(R) Xeon(R) CPU           X5650  @␣
↪2.67GHz",
            "host.2.cputemp": 0,
            "host.2.devices": [ "/dev/sda1" ],
            "host.2.free_disk_bytes": 1404706975744,
            "host.2.hostname": "192.168.1.127",
            "host.2.interfaces": [ "eth0" ],
            "host.2.ip": "192.168.1.127",
            "host.2.membuffer": 232240,
            "host.2.memcached": 13097476,
            "host.2.memfree": 14108052,
            "host.2.memtotal": 49453276,
            "host.2.network_interfaces": [
            {
                "interface_name": "eth0",
                "rx_bytes_per_sec": 17268.2,
                "tx_bytes_per_sec": 11158.5
            } ],
            "host.2.pingdelay": -1,
            "host.2.pingstatustext": "Creating ICMP socket (to ping
↪'192.168.1.127') failed: Operation not permitted.",
            "host.2.port": 9555,
            "host.2.rx_bytes_per_second": 17268.2,
            "host.2.swapfree": 0,
            "host.2.swaptotal": 0,
            "host.2.total_disk_bytes": 2125259440128,
            "host.2.tx_bytes_per_second": 11158.5,
            "host.2.uptime": 2.65439e+06,
            "host.2.version": "1.4.3",
            "host.2.wallclock": 1501216110,
            "host.2.wallclocksampled": 1501216110,
            "license.expires": -1,
            "license.status": false,
            "license.statustext": "No license found.",
```

```
            "mail.statustext": "Created.",
            "netStat.1.eth0.rxBytes": 5238632,
            "netStat.1.eth0.txBytes": 33144284,
            "netStat.2.eth0.rxBytes": 531337507751,
            "netStat.2.eth0.txBytes": 505661940612
        },
        "job_statistics":
        {
            "by_state":
            {
                "ABORTED": 0,
                "DEFINED": 0,
                "DEQUEUED": 0,
                "FAILED": 0,
                "FINISHED": 0,
                "RUNNING": 0
            },
            "class_name": "CmonJobStatistics",
            "cluster_id": 1
        },
        "log_file": "/tmp/cmon_1.log",
        "maintenance_mode_active": false,
        "managed": true,
        "node_auto_recovery": true,
        "owner":
        {
            "class_name": "CmonUser",
            "email_address": "laszlo@severalnines.com",
            "first_name": "Laszlo",
            "groups": [
            {
                "class_name": "CmonGroup",
                "group_id": 4,
                "group_name": "testgroup"
            } ],
            "last_login": "2017-07-28T04:29:14.185Z",
            "last_name": "Pere",
            "title": "",
            "user_id": 3,
            "user_name": "pipas"
        },
        "state": "STARTED",
        "status_text": "All nodes are operational.",
        "vendor": "postgres",
        "version": "9.6"
    } ],
    "reply_received": "2017-07-28T04:29:14.197Z",
    "request_created": "2017-07-28T04:29:14.193Z",
    "request_id": 3,
    "request_processed": "2017-07-28T04:29:14.241Z",
```

```
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 1
}
```

For the "getAllClusterInfo" request the controller is able to send back a lot of information. A full view of the cluster can be implemented using the reply. The following main categories can be found here:

- `cluster`: A list of clusters. All the details are categorised under these cmonclusterinfo objects.
- `hosts`: The list of hosts that are part of the cluster. This piece of information is sent if the request had the "with_hosts" property set to true.
- `info`: The information that the sheet manager collected about the cluster.
- `alarm_statistics`: Shows the number of alarms on the syste. further information can be found in the CmonSimpleObject documentation.
- `job_statistics`: Statistical information about the jobs the cluster has. Further information can be found in the CmonJobStatistics page.
- `total`: The total number of clusters managed by the controller.

### getAllClusterInfoWithAllPool

!!! post "POST /v2/clusters" This request behaves exactly like getAllClusterInfo but aggregates data from all active controllers in the pool. It applies the same filtering logic, access controls, and generates the same response structure as the normal getAllClusterInfo method.

The difference is that instead of getting data from the local controller only, it retrieves cluster information from all active controllers and presents them in a unified response.

### createAccount

!!! post "POST /v2/clusters" This request can be used to create an account on the cluster. Please note that this is not going to be a Cmon User, this RPC is for creating SQL users.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| `account` | object | Require | A CmonAccount class object that defines the properties of the account that will be created. To get a comprehensive list of the properties the controller supports in the CmonAccount class one can run the following command (or check the cmonaccount section): `s9s metatype --list-properties --type=CmonAccount --long` |
| `cluster_` | integer | Require | The numerical ID of the cluster on which the account will be created. The cluster can also be identified by its name. |
| `cluster_` | string | Optional | The name of the cluster on which the new account will be created. The cluster can also be identified by its numerical ID. |

```
{
    "account":
    {
        "class_name": "CmonAccount",
        "grants": "john:ALL;pipas:INSERT",
        "host_allow": "1.2.3.4",
        "own_database": "laszlo",
        "password": "passwd",
        "user_name": "laszlo"
    },
    "cluster_id": 1,
    "operation": "createAccount",
    "request_created": "2017-01-27T14:03:09.491Z"
}
```

### updateAccount

!!! post "POST /v2/clusters" This request can be used to update an account on the cluster. Please note that this is not going to be a Cmon User, this RPC is for creating SQL users.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| `account` | object | Require | A CmonAccount class object that defines the properties of the account that will be updated. To get a comprehensive list of the properties the controller supports in the CmonAccount class one can run the following command (or check the cmonaccount section): `s9s metatype --list-properties --type=CmonAccount --long` |
| `cluster_id` | integer | Require | The numerical ID of the cluster on which the account will be updated. The cluster can also be identified by its name. |
| `cluster_name` | string | Optional | The name of the cluster on which the account will be updated. The cluster can also be identified by its numerical ID. |

```
{
    "account":
    {
        "class_name": "CmonAccount",
        "grants": "john:ALL;pipas:INSERT",
        "host_allow": "1.2.3.4",
        "own_database": "laszlo",
        "password": "passwd",
        "user_name": "laszlo",
        "ssl_type": "SSL",
        "max_user_connections": 0,
```

(continues on next page)

(continued from previous page)

```
        "max_questions": 0,
        "max_updates": 0,
        "max_connections": 0
    },
    "cluster_id": 1,
    "operation": "updateAccount",
    "request_created": "2017-01-27T14:03:09.491Z"
}
```

### getAccounts

!!! post "POST /v2/clusters" The getAccounts call can be used to get the accounts from the cluster.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The numerical ID of the cluster to get. The cluster can also be identified by its name. |
| `cluster_name` | string | Optional | The name of the cluster to get. The cluster can also be identified by its numerical ID. |
| `limit` | integer | Optional | Limits the number of accounts sent back in the reply. |
| `offset` | integer | Optional | The offset of the first account sent back in the reply. |

```
{
    "cluster_id": 1,
    "operation": "getAccounts",
    "request_created": "2017-08-10T03:03:49.592Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `accounts` | array | A list of cmonaccount objects, the accounts found on the cluster. |
| `total` | integer | The total number of accounts found on the cluster regardless the number of accounts sent back in the reply. |

**Response example**

```
{
    "accounts": [
```

```
    {
        "class_name": "CmonAccount",
        "connections": 1,
        "grants": "CREATE USER,CREATE␣
↪DATABASE,LOGIN,REPLICATION,SUPER;template1:CREATE,TEMPORARY,CONNECT;template0:CREATE,T
↪
        "host_allow": "",
        "max_connections": 0,
        "password": "",
        "user_name": "postgres"
    },

    . . .

    {
        "class_name": "CmonAccount",
        "grants": "LOGIN;joe:CREATE,TEMPORARY,CONNECT",
        "host_allow": "",
        "max_connections": 0,
        "password": "",
        "user_name": "joe"
    } ],
    "request_created": "2017-08-10T03:03:49.592Z",
    "request_id": 3,
    "request_processed": "2017-08-10T03:03:49.639Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 5
}
```


### listAccounts

!!! post “POST /v2/clusters” The listAccounts call can be used to get the accounts from the cluster.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The numerical ID of the cluster to get. The cluster can also be identified by its name. |
| `cluster_name` | string | Optional | The name of the cluster to get. The cluster can also be identified by its numerical ID. |
| `limit` | integer | Optional | Limits the number of accounts sent back in the reply. |
| `offset` | integer | Optional | The offset of the first account sent back in the reply. |

```
{
    "cluster_id": 1,
    "operation": "listAccounts",
    "request_created": "2017-08-10T03:03:49.592Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| accounts | array | A list of cmonaccount objects, the accounts found on the cluster. |
| total | integer | The total number of accounts found on the cluster regardless the number of accounts sent back in the reply. |

**Response example**

```
{
    "queryResults": [
      {
        "accounts": [
         {
              "class_name": "CmonAccount",
              "connections": 1,
              "grants": "CREATE USER,CREATE␣
↪DATABASE,LOGIN,REPLICATION,SUPER;template1:CREATE,TEMPORARY,CONNECT;template0:CREATE,T
↪
              "host_allow": "",
              "max_connections": 0,
              "password": "",
              "user_name": "postgres"
         },

         . . .

         {
              "class_name": "CmonAccount",
              "grants": "LOGIN;joe:CREATE,TEMPORARY,CONNECT",
              "host_allow": "",
              "max_connections": 0,
              "password": "",
              "user_name": "joe"
         } ],
        "hostname": "masterHostName123",
        "port": 3306
       }
    ],
    "request_created": "2017-08-10T03:03:49.592Z",
```

```
    "request_id": 3,
    "request_processed": "2017-08-10T03:03:49.639Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 5
}
```

### deleteAccount

!!! post "POST /v2/clusters" This request is for deleting a user account from a cluster.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| account | object | Require | The account that should be deleted. To get a comprehensive list of the properties the controller supports in the CmonAccount class one can run the following command (or check the cmonaccount section): `s9s metatype --list-properties --type=CmonAccount --long` |
| cluster_i | integer | Require | The numerical ID of the cluster from which the account should be removed. The cluster can also be identified by the cluster name. |
| cluster_r | string | Optiona | The name of the cluster from which the account will be removed. The cluster can also be identified by its numerical ID. |

```
{
    "account":
    {
        "class_name": "CmonAccount",
        "user_name": "pipas"
    },
    "cluster_id": 1,
    "operation": "deleteAccount",
    "request_created": "2017-01-30T13:12:30.192Z"
}
```

### grantPrivileges

!!! post "POST /v2/clusters" The request can be used to grant privileges for a existing user to an existing database.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| account | object | Require | The account that will be granted some privileges on some database. To get a comprehensive list of the properties the controller supports in the CmonAccount class one can run the following command (or check the cmonaccount section): `s9s metatype --list-properties --type=CmonAccount --long` |
| cluster_ | integer | Require | The numerical ID of the cluster on which the privileges will be granted. The cluster can also be identified by name. |
| cluster_ | string | Optional | The name of the cluster on which the privileges will be granted. The cluster can also be identified by the cluster ID. |
| privilege | string | Require | The privileges to grant. This property should be a string in pdl_language format. |

```
{
    "account":
    {
        "class_name": "CmonAccount",
        "user_name": "pipas"
    },
    "cluster_id": 1,
    "operation": "grantPrivileges",
    "privileges": "testCreateDatabase.*:DELETE,TRUNCATE",
}
```

### Response

**Response example**

```
{
    "messages": [ "Grant processed: GRANT DELETE,TRUNCATE ON DATABASE␣
↪testCreateDatabase TO pipas" ],
    "request_processed": "2017-06-22T08:49:25.548Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### revokePrivileges

!!! post “POST /v2/clusters” The request can be used to revoke privileges from an existing user to an existing database.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| account | object | Required | The account from privileges will be revoked on some database. To get a comprehensive list of the properties the controller supports in the CmonAccount class one can run the following command (or check the cmonaccount section): `s9s metatype --list-properties --type=CmonAccount --long` |
| cluster_ | integer | Required | The numerical ID of the cluster on which the privileges will be granted. The cluster can also be identified by name. |
| cluster_ | string | Optional | The name of the cluster on which the privileges will be granted. The cluster can also be identified by the cluster ID. |
| privilege | string | Required | The privileges to grant. This property should be a string in pdl_language format. |

```
{
    "account":
    {
        "class_name": "CmonAccount",
        "user_name": "pipas"
    },
    "cluster_id": 1,
    "operation": "revokePrivileges",
    "privileges": "testCreateDatabase.*:DELETE,TRUNCATE",
}
```

### Response

**Response example**


```
{
    "messages": [ "Revoke processed: REVOKE DELETE,TRUNCATE ON DATABASE␣
↪testCreateDatabase FROM pipas" ],
    "request_processed": "2017-06-22T08:49:25.548Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### revokeAllPrivileges

!!! post "POST /v2/clusters" The request can be used to revoke all privileges from an existing user.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| account | object | Require | The account from all privileges will be revoked. To get a comprehensive list of the properties the controller supports in the CmonAccount class one can run the following command (or check the cmonaccount section): `s9s metatype --list-properties --type=CmonAccount --long` |
| cluster_i | integer | Require | The numerical ID of the cluster on which the privileges will be granted. The cluster can also be identified by name. |
| cluster_n | string | Optional | The name of the cluster on which the privileges will be granted. The cluster can also be identified by the cluster ID. |


```
{
    "account":
    {
        "class_name": "CmonAccount",
        "user_name": "pipas"
    },
    "cluster_id": 1,
    "operation": "revokePrivileges"
}
```


### Response

**Response example**


```
{
    "messages": [ "Revoke processed: REVOKE ALL PRIVILEGES ON DATABASE␣
↪testCreateDatabase FROM pipas" ],
    "request_processed": "2017-06-22T08:49:25.548Z",
```

```
    "request_status": "Ok",
    "request_user_id": 3
}
```

### getSqlProcesses

!!! post "POST /v2/clusters" Returns the list of currently active SQL processes (connections and running queries) across all nodes in a cluster. Useful for monitoring active workloads and identifying long-running or blocking queries.

### Request

```
{
    "cluster_id": 1,
    "operation": "getSqlProcesses",
    "request_created": "2020-02-28T06:24:36.355Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `processes` | array | List of active SQL process objects across all nodes in the cluster. |

**Response example**

```
{
    "debug_messages": [ "RPC V2 authenticated user is 'pipas'." ],
    "processes": [
    {
        "blocked_by_trx_id": "",
        "client": "",
        "command": "Sleep",
        "currentTime": 1582871076,
        "db": "",
        "duration": 139914683312432,
        "host": "",
        "hostId": 1,
        "hostname": "192.168.0.227",
        "info": "",
        "innodb_status": "",
        "innodb_trx_id": "",
        "instance": "192.168.0.227:3306",
        "lastseen": 139914683312176,
        "message": "",
        "mysql_trx_id": 1,
        "pid": 1,
```

```
        "query": "",
        "reportTs": 1582871076,
        "sql": "",
        "state": "wsrep aborter idle",
        "time": 664,
        "user": "system user"
    },
    {
        "blocked_by_trx_id": "",
        "client": "",
        "command": "Sleep",
        "currentTime": 1582871076,
        "db": "",
        "duration": 139914683312432,
        "host": "",
        "hostId": 3,
        "hostname": "192.168.0.220",
        "info": "",
        "innodb_status": "",
        "innodb_trx_id": "",
        "instance": "192.168.0.220:3306",
        "lastseen": 139914683312176,
        "message": "",
        "mysql_trx_id": 5,
        "pid": 5,
        "query": "",
        "reportTs": 1582871076,
        "sql": "",
        "state": "",
        "time": 130,
        "user": "system user"
    } ],
    "request_created": "2020-02-28T06:24:36.355Z",
    "request_id": 3,
    "request_processed": "2020-02-28T06:24:36.378Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```


### getTopQueries

!!! post “POST /v2/clusters” This request can be used to get the top queries, actually the top query patterns (AKA digests).

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The numerical ID of the cluster to query. |
| `limit` | integer | Optional | Optional number that limits the number of items returned. |
| `offset` | integer | Optional | Optional number, the offset of the first item returned. |
| `filterStrings` | array | Optional | Optional list of filters to be applied on the server name and port (the "instance" of the entry) the same way the file name patterns are handled in the UNIX shell. |


```
{
    "cluster_id": 1,
    "filterStrings": [ "*:3306" ],
    "limit": 10,
    "offset": 10,
    "operation": "getTopQueries",
    "request_created": "2020-03-30T09:57:12.971Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `digests` | array | List of CmonStatementDigest objects representing the top query patterns. |
| `total` | integer | The total number of digests found. |

**Response example**


```
{
    "debug_messages": [ "RPC V2 authenticated user is 'pipas'." ],
    "digests": [
    {
        "affectedRows": 9020,
        "class_name": "CmonStatementDigest",
        "count": 9020,
        "databaseName": "pipas1_3198",
        "dtFirstSeen": "2020-03-25T14:37:24.000Z",
        "dtLastSeen": "2020-03-25T14:44:02.000Z",
        "instance": "192.168.0.215:6032",
        "statementPattern": "INSERT INTO test_table(NAME, VALUE)␣
```

```
↪VALUES(?, ?)",
        "waitMillisMax": 610.891,
        "waitMillisMin": 8.935,
        "waitMillisSum": 324158
    },
    {
        "affectedRows": 9020,
        "class_name": "CmonStatementDigest",
        "count": 9020,
        "databaseName": "pipas1_3198",
        "dtFirstSeen": "2020-03-25T14:37:24.000Z",
        "dtLastSeen": "2020-03-25T14:44:02.000Z",
        "instance": "192.168.0.227:3306",
        "statementPattern": "INSERT INTO `test_table` ( NAME , VALUE )␣
↪VALUES (...) ",
        "waitMillisAvg": 35.7737,
        "waitMillisMax": 610.721,
        "waitMillisMin": 8.78106,
        "waitMillisSum": 322679
    } ],
    "request_created": "2020-03-26T08:33:01.146Z",
    "request_id": 3,
    "request_processed": "2020-03-26T08:33:01.202Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "total": 37
}
```

### createDatabase

!!! post "POST /v2/clusters" This request can be used to create a database on the cluster.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The numerical ID of the cluster on which the new database will be created. The cluster can also be identified by its name. |
| `cluster_name` | string | Optional | The name of the cluster on which the new database will be created. The cluster can also be identified by its name. |
| `database` | object | Required | The database to be created. |

```
{
    "cluster_id": 1,
    "database":
    {
```

(continues on next page)

(continued from previous page)

```
        "class_name": "CmonDataBase",
        "database_name": "dbname"
    },
    "operation": "createDatabase",
    "request_created": "2017-06-20T07:24:15.815Z"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| database | object | The created CmonDataBase object. |

**Response example**

```
{
    "database":
    {
        "class_name": "CmonDataBase",
        "database_name": "testDatabase"
    },
    "messages": [ "Database 'testDatabase' created." ],
    "reply_received": "2017-08-10T08:16:17.949Z",
    "request_created": "2017-08-10T08:16:17.945Z",
    "request_id": 3,
    "request_processed": "2017-08-10T08:16:18.098Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### ping

!!! post “POST /v2/clusters” The “ping” request can be used to check the connection to the Cmon Controller or keep it alive. Every ping request will be replied with a simple message that holds some general information about the controller and the connection.

### Request

```
{
    "operation": "ping",
    "request_created": "2017-06-22T11:55:39.854Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| authenticated | boolean | Whether the connection is authenticated. |
| package_bugreport | string | Contact address for bug reports. |
| package_name | string | The name of the package. |
| package_version | string | The version of the package. |
| rpc_version | string | The RPC API version. |

**Response example**

```
{
    "authenticated": true,
    "package_bugreport": "support@severalnines.com",
    "package_name": "cmon",
    "package_version": "1.4.3",
    "request_created": "2017-06-22T11:55:39.854Z",
    "request_id": 3,
    "request_processed": "2017-06-22T11:55:39.900Z",
    "request_status": "Ok",
    "request_user_id": 12,
    "rpc_version": "2.0"
}
```

### availableUpgrades

!!! post "POST /v2/clusters" This request is for reading a list of available package upgrades for the cluster. The cluster can be identified by the cluster ID or the cluster name.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| nodes | array | Optional | List of nodes (hosts) to query available package upgrades on. |
| cluster_id | integer | Required | The numerical ID of the cluster to get. The cluster can also be identified by its name. |
| cluster_name | string | Optional | The name of the cluster to get. The cluster can also be identified by its numerical ID. |

```
{
    "cluster_id": 1,
    "operation": "availableUpgrades",
}
```

### 4.12.10 Configuration

The `/v2/config` path provides access to various configuration files.

| Operation | Description |
|---|---|
| *getConfig* | Get the configuration of one specific node in a cluster. |
| *getConfigs* | Obtain all configuration files of a cluster. |
| *setConfig* | Change individual configuration values for a host. |
| *unsetConfig* | Unset or comment out individual configuration values for a host. |
| *getLdapConfig* | Read the LDAP configuration. |
| *setLdapConfig* | Write the LDAP configuration. |
| *listTemplates* | List the ClusterControl provided configuration templates. |

#### getConfig

!!! post "POST /v2/config" This call is for getting the configuration of one specific node in a cluster.

Access rights: The authenticated user has to have read access on both the host and the cluster to get successful reply on this call.

#### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `hostname` | string | Required | The hostname of the node to get configuration for. |

```
{
    "hostname": "192.168.1.127",
    "operation": "getConfig",
    "request_created": "2017-07-28T09:48:30.491Z",
    "request_id": 3
}
```

#### Response

| Field | Type | Description |
|---|---|---|
| `config` | object | Contains the configuration of the host. |
| `file` | array | One host can have multiple configuration files, so this part contains a list one element for every configuration file. |
| `values` | array | The parsed version of the configuration. Every item is one configuration value in some configuration file. |
| `content` | string | The literal content of the configuration file escaped for JSON format. |
| `path` | string | The full path of the configuration file. |
| `syntax` | string | The syntax of the configuration file. This can be GenericConfigSyntax, MySqlConfigSyntax, ProxySqlConfigSyntax, HaProxyConfigSyntax, YamlSyntax, UnknownSyntax or MongoConfigSyntax which is the same as YamlSyntax. |

**Response example**


```
{
    "config":
    {
        "className": "CmonClusterConfig",
        "files": [
        {
            "filename": "cmon_1.cnf",
            "values": [
            {
                "filepath": "/tmp/cmon_1.cnf",
                "linenumber": 1,
                "section": "",
                "value": "1",
                "variablename": "cluster_id"
            },

            . . .

            {
                "filepath": "/tmp/cmon_1.cnf",
                "linenumber": 29,
                "section": "",
                "value": "postgres",
                "variablename": "vendor"
            } ]
        } ],
        "host":
        {
            "class_name": "CmonHost",
            "hostname": "192.168.1.127",
            "port": 9555
        }
    },
    "files": [
    {
        "content":
↪"cluster_id=1\ncluster_type=postgresql_single\ncmon_db=ft_install_testdb\ncmon_user=rc
↪'p'\nmysql_port=3306\nname='ft_postgresql_63564'\
↪nos=debian\nosuser=pipas\nowner=3\npidfile=/var/run\npostgresql_password=passwd12\npos
↪
        "filename": "cmon_1.cnf",
        "path": "/tmp/cmon_1.cnf",
        "syntax": "MySqlConfigSyntax"
    } ],
    "request_created": "2017-07-28T09:48:30.491Z",
    "request_id": 3,
    "request_processed": "2017-07-28T09:48:30.541Z",
    "request_status": "Ok",
    "request_user_id": 3
```

```
}
```

### getConfigs

!!! post "POST /v2/config" A method to obtain all configuration files of a cluster.

Access rights: The authenticated user has to have read access on the cluster to get successful reply on this call.

### Request

```
{
    "operation": "getConfigs",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `configs` | array | The configurations of the nodes, one element per node. |

**Response example**

```
{
    "configs":
    [
      {"config": {
        "className": "CmonClusterConfig",
        "files": [
        {
            "filename": "cmon_1.cnf",
            "values": [
            {
                "filepath": "/tmp/cmon_1.cnf",
                "linenumber": 1,
                "section": "",
                "value": "1",
                "variablename": "cluster_id"
            },

            . . .

            {
                "filepath": "/tmp/cmon_1.cnf",
                "linenumber": 29,
                "section": "",
                "value": "postgres",
                "variablename": "vendor"
            } ]
```

```
        } ],
        "host":
        {
            "class_name": "CmonHost",
            "hostname": "192.168.1.127",
            "port": 9555
        }
      },
      "files": [ {
        "content":
↪"cluster_id=1\ncluster_type=postgresql_single\ncmon_db=ft_install_testdb\ncmon_user=rc
↪'p'\nmysql_port=3306\nname='ft_postgresql_63564'\
↪nos=debian\nosuser=pipas\nowner=3\npidfile=/var/run\npostgresql_password=passwd12\npos
↪
        "filename": "cmon_1.cnf",
        "path": "/tmp/cmon_1.cnf",
        "syntax": "MySqlConfigSyntax"
      } ]},
    ],
    "request_created": "2023-02-23T09:48:30.491Z",
    "request_id": 3,
    "request_processed": "2023-02-23T09:48:30.541Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### setConfig

!!! post "POST /v2/config" This call is for changing individual configuration values for a host. Multiple values can be changed by sending one request.

Access rights: The authenticated user has to have write access on both the host and the cluster to get successful reply on this call.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `configuration` | array | Required | A list of values to be changed. |
| `hostname` | string | Required | The name of the host where the configuration should be changed. |
| `port` | integer | Optional | If the hostname is not enough to identify the node the sender can provide the port number to identify the node. The port is not mandatory, but might be needed. |

```
{
    "configuration": [
    {
        "name": "log_line_prefix",
        "value": "'%m'"
    } ],
    "hostname": "192.168.1.127",
    "operation": "setConfig",
    "port": 9555,
    "request_created": "2017-07-28T12:06:53.093Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| messages | array | This call can send multiple messages back. Changing the configuration is a complex step and may result in more than one messages. |

**Response example**


```
{
    "messages": [ "Node restart is required for change to take effect." ],
    "request_created": "2017-07-28T12:06:53.093Z",
    "request_id": 3,
    "request_processed": "2017-07-28T12:06:53.157Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### unsetConfig

!!! post “POST /v2/config” This call is for unsetting, commenting out individual configuration values for a host. Multiple values can be unset by sending one request.

Access rights: The authenticated user has to have write access on both the host and the cluster to get successful reply on this call.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| configuration | array | Required | A list of variable names to be changed. |
| hostname | string | Required | The name of the host where the configuration should be changed. |
| port | integer | Optional | If the hostname is not enough to identify the node the sender can provide the port number to identify the node. The port is not mandatory, but might be needed. |

```
{
    "configuration": [
    {
        "group": "groupname",
        "name": "log_line_prefix",
    } ],
    "hostname": "192.168.1.127",
    "operation": "unsetConfig",
    "port": 9555,
    "request_created": "2017-07-28T12:06:53.093Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| messages | array | This call can send multiple messages back. Changing the configuration is a complex step and may result in more than one messages. |

**Response example**


```
{
    "messages": [ "Node restart is required for change to take effect." ],
    "request_created": "2017-07-28T12:06:53.093Z",
    "request_id": 3,
    "request_processed": "2017-07-28T12:06:53.157Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### getLdapConfig

!!! post "POST /v2/config" This call is for reading the LDAP configuration. Access to the LDAP configuration is controlled by the access rights for the "/.runtime/LDAP/configuration" CDT entry, whoever has read access to that CDT entry can get the LDAP configuration through this call.

### Request


```
{
  "operation": "getLdapConfig",
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 2
}
```


### Response

| Field | Type | Description |
|---|---|---|
| effective_privil | string | Shows what access privileges the authenticated user has. So with this field the caller can immediately decide if the user has the rights to change the settings or not. |
| ldap_config | object | The LDAP configuration. |

**Response example**


```
{
  "controller_id": "ae03400a-b1d1-42eb-98ad-3196acc1ec23",
  "effective_privileges": "rwx",
  "reply_received": "2021-01-13T07:22:33.878Z",
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 2,
  "request_processed": "2021-01-13T07:22:33.877Z",
  "request_status": "Ok",
  "request_user_id": 1,
  "debug_messages": [ "RPC V2 authenticated user is 'system'."  ],
  "ldap_configuration": {
```

```
    "enabled": false,
    "ldapAdminPassword": null,
    "ldapAdminUser": null,
    "ldapGroupSearchRoot": null,
    "ldapServerUri": null,
    "ldapUserSearchRoot": null,
    "defaults": {
      "enabled": false,
      "ldapAdminPassword": "",
      "ldapAdminUser":
↪"CN=Administrator,CN=Users,DC=ldap,DC=mydomain,DC=com",
      "ldapGroupSearchRoot": "CN=Groups,OU=IT,DC=ldap,DC=mydomain,DC=com",
      "ldapServerUri": "ldaps://ldap.mydomain.com:686",
      "ldapUserSearchRoot": "CN=Users,DC=ldap,DC=mydomain,DC=com",
      "groupMappings": [  ],
      "ldapSettings": {
        "ldapEmailAttributes": "mail,userPrincipalName",
        "ldapGroupClassName": "",
        "ldapGroupIdAttributes": "dn,uid",
        "ldapGroupNameAttribute": "cn",
        "ldapGroupSearchRoot":
↪"CN=Groups,OU=IT,DC=ldap,DC=mydomain,DC=com",
        "ldapMemberAttributes": "member,memberUid",
        "ldapNetworkTimeout": 5,
        "ldapProtocolVersion": 3,
        "ldapQueryTimeLimit": 5,
        "ldapRealnameAttributes": "cn,displayName",
        "ldapUserClassName": "objectclass=*",
        "ldapUsernameAttributes": "uid,sAMAccountName"
      },
      "security": {
        "caCertFile": "/etc/ssl/certs/ldap-ca.crt",
        "certFile": "/etc/ssl/certs/ldap.crt",
        "keyFile": "/etc/ssl/certs/ldap.pem"
      }
    },
    "groupMappings": [  ],
    "ldapSettings": {
      "ldapEmailAttributes": null,
      "ldapGroupClassName": null,
      "ldapGroupIdAttributes": null,
      "ldapGroupNameAttribute": null,
      "ldapMemberAttributes": null,
      "ldapNetworkTimeout": null,
      "ldapProtocolVersion": null,
      "ldapQueryTimeLimit": null,
      "ldapRealnameAttributes": null,
      "ldapUserClassName": null,
      "ldapUsernameAttributes": null
    },
```

```
    "security": {
      "caCertFile": null,
      "certFile": null,
      "keyFile": null
    }
  }
}
```

### setLdapConfig

!!! post "POST /v2/config" This call is for writing the LDAP configuration. Access to the LDAP configuration is controlled by the access rights for the "/.runtime/LDAP/configuration" CDT entry, whoever has write access to that CDT entry can set the LDAP configuration through this call.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `ldap_configuration` | object | Required | The LDAP configuration. |

```
{
  "operation": "setLdapConfig",
  "request_created": "2021-01-13T07:22:33.966Z",
  "request_id": 2,
  "ldap_configuration": {
    "enabled": true,
    "ldapAdminPassword": "p",
    "ldapAdminUser": "cn=admin,dc=homelab,dc=local",
    "ldapGroupSearchRoot": "dc=homelab,dc=local",
    "ldapServerUri": "ldap://ldap.homelab.local:389",
    "ldapUserSearchRoot": "dc=homelab,dc=local",
    "groupMappings": [ {
        "cmonGroupName": "ldapgroup",
        "ldapGroupId": "ldapgroup",
        "sectionName": "mapping1"
      }  ],
    "ldapSettings": {
      "ldapEmailAttributes": "mail",
      "ldapGroupClassName": null,
      "ldapGroupIdAttributes": null,
      "ldapGroupNameAttribute": null,
      "ldapMemberAttributes": "memberUid",
      "ldapNetworkTimeout": null,
      "ldapProtocolVersion": null,
      "ldapQueryTimeLimit": null,
      "ldapRealnameAttributes": "displayName,cn",
      "ldapUserClassName": null,
      "ldapUsernameAttributes": "cn"
    },
```

(continues on next page)

(continued from previous page)

```
    "security": {
      "caCertFile": null,
      "certFile": null,
      "keyFile": null
    }
  }
}
```

### Response

**Response example**

```
{
  "controller_id": "ae03400a-b1d1-42eb-98ad-3196acc1ec23",
  "reply_received": "2021-01-13T07:22:33.980Z",
  "request_created": "2021-01-13T07:22:33.966Z",
  "request_id": 2,
  "request_processed": "2021-01-13T07:22:33.979Z",
  "request_status": "Ok",
  "request_user_id": 1,
  "debug_messages": [ "RPC V2 authenticated user is 'system'."  ]
}
```

## listTemplates

!!! post "POST /v2/config" This call is to list the ClusterControl provided configuration templates.

### Request

```
{
  "operation": "listTemplates",
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 2
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `data` | object | Contains the configuration templates for the given cluster type, vendor, and version. |

**Response example**

```
{
  "controller_id": "ae03400a-b1d1-42eb-98ad-3196acc1ec23",
  "reply_received": "2021-01-13T07:22:33.878Z",
```

```
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 2,
  "request_processed": "2021-01-13T07:22:33.877Z",
  "request_status": "Ok",
  "data":
  {
      "cluster_type": "replication",
      "config_templates": [ "my.cnf.gtid_replication" ],
      "vendor": "percona",
      "version": "5.6",
      "requestStatus": "ok",
  }
}
```


### setLicense

!!! post "POST /v2/config" immediately.

Here is how a request should look like:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `licensedata` | string | Required | The base64-encoded license data string. |


```
{
  "operation": "setLicense",
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 2,
  "licensedata":
↪"ewogICAgImNvbXBhbnkiOiAiU2FudGEgQ2xhdXNlIEluYy4iLAogICAgImVtYWlsX2FkZHJlc3Mi\nOiAic2Fu
}
```

### Response

| Field | Type | Description |
|---|---|---|
| data | object | Object containing the license status and details. |
| data.hasLicense | boolean | Whether a valid license is present. |
| data.licenseExpires | integer | Number of days until the license expires. |
| data.licenseStatus | string | Human-readable license status message. |
| data.license | object | The license object with full details. |
| data.license.class_name | string | Class name of the license object. |
| data.license.company | string | Company name the license is issued to. |
| data.license.days_left | integer | Days remaining on the license. |
| data.license.email_address | string | Email address associated with the license. |
| data.license.expiration_date | string | ISO 8601 expiration date of the license. |
| data.license.licensed_nodes | integer | Number of nodes the license covers. |
| data.license.type | string | License type (e.g., Demo). |
| data.license.used_nodes | integer | Number of nodes currently in use. |
| data.license.valid_date | boolean | Whether the license date is valid. |

**Response example**


```
{
  "controller_id": "ae03400a-b1d1-42eb-98ad-3196acc1ec23",
  "reply_received": "2021-01-13T07:22:33.878Z",
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 2,
  "request_processed": "2021-01-13T07:22:33.877Z",
  "request_status": "Ok",
  "data":
  {
     "hasLicense":true,
     "licenseExpires":387,
     "licenseStatus":"License found.",
     "license":{
         "class_name":"CmonLicense",
         "company":"Santa Clause Inc.",
         "days_left":16,
         "email_address":"santa@hoho.ho",
         "expiration_date":"2021-12-24T00:00:00.001Z",
         "licensed_nodes":7,
         "type":"Demo",
         "used_nodes":89,
         "valid_date":true
     }
  }
}
```

### getLicense

!!! post "POST /v2/config" Returns the current license information registered with the controller, including the license type, expiration date, licensed node count, and current usage.

### Request

```
{
  "operation": "getLicense",
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 3,
}
```

### Response

| Field | Type | Description |
|---|---|---|
| data | object | Object containing the license status and details. |
| data.hasLicense | boolean | Whether a valid license is present. |
| data.licenseExpires | integer | Number of days until the license expires. |
| data.licenseStatus | string | Human-readable license status message. |
| data.license | object | The license object with full details. |
| data.license.class_name | string | Class name of the license object. |
| data.license.company | string | Company name the license is issued to. |
| data.license.days_left | integer | Days remaining on the license. |
| data.license.email_address | string | Email address associated with the license. |
| data.license.expiration_date | string | ISO 8601 expiration date of the license. |
| data.license.licensed_nodes | integer | Number of nodes the license covers. |
| data.license.type | string | License type (e.g., Demo). |
| data.license.used_nodes | integer | Number of nodes currently in use. |
| data.license.valid_date | boolean | Whether the license date is valid. |

**Response example**

```
{
  "controller_id": "ae03400a-b1d1-42eb-98ad-3196acc1ec23",
  "reply_received": "2021-01-13T07:22:33.878Z",
  "request_created": "2021-01-13T07:22:33.872Z",
  "request_id": 2,
  "request_processed": "2021-01-13T07:22:33.877Z",
  "request_status": "Ok",
  "data":
  {
     "hasLicense":true,
     "licenseExpires":387,
     "licenseStatus":"License found.",
     "license":{
         "class_name":"CmonLicense",
         "company":"Santa Clause Inc.",
         "days_left":16,
```

```
            "email_address":"santa@hoho.ho",
            "expiration_date":"2021-12-24T00:00:00.001Z",
            "licensed_nodes":7,
            "type":"Demo",
            "used_nodes":89,
            "valid_date":true
        }
    }
}
```

### setmailserver

!!! post "POST /v2/config" Sets the SMTP mail server configuration used by the controller for sending notifications.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| smtp_server | object | Required | Object containing SMTP server configuration. |
| smtp_server.hostname | string | Required | Hostname of the SMTP server. |
| smtp_server.password | string | Required | Password for SMTP authentication. |
| smtp_server.port | integer | Required | SMTP server port number. |
| smtp_server.sender | string | Required | Email address used as the sender. |
| smtp_server.use_tls | boolean | Required | Whether to use TLS for the SMTP connection. |
| smtp_server.username | string | Required | Username for SMTP authentication. |

```
{
  "operation": "setmailserver",
  "smtp_server":
  {
    "hostname": "my.smtp.server",
    "password": "password",
    "port": 587,
    "sender": "hullala@smtp.server.tld",
    "use_tls": false,
    "username": "hullala@smtp.server.tld"
  }
}
```

### Response

**Response example**

```
{
  "controller_id": "ae03400a-b1d1-42eb-98ad-3196acc1ec23",
  "reply_received": "2021-01-13T07:22:33.878Z",
  "request_created": "2021-01-13T07:22:33.872Z",
```

```
  "request_id": 2,
  "request_processed": "2021-01-13T07:22:33.877Z",
  "request_status": "Ok",
}
```

### getmailserver

!!! post "POST /v2/config" Returns the current SMTP mail server configuration registered with the controller.

### Request

```
{
  "operation": "getmailserver",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| smtp_server | object | Object containing the SMTP server configuration. |
| smtp_server.hostname | string | Hostname of the SMTP server. |
| smtp_server.password | string | Password for SMTP authentication. |
| smtp_server.port | integer | SMTP server port number. |
| smtp_server.sender | string | Email address used as the sender. |
| smtp_server.use_tls | boolean | Whether TLS is enabled for the SMTP connection. |
| smtp_server.username | string | Username for SMTP authentication. |

**Response example**

```
{
  "controller_id": "ae03400a-b1d1-42eb-98ad-3196acc1ec23",
  "reply_received": "2022-01-13T07:22:33.878Z",
  "request_created": "2022-01-13T07:22:33.872Z",
  "request_id": 3,
  "request_processed": "2022-01-13T07:22:33.877Z",
  "request_status": "Ok",
  "smtp_server":
  {
      "hostname": "my.smtp.server",
      "password": "password",
      "port": 587,
      "sender": "hullala@smtp.server.tld",
      "use_tls": false,
      "username": "hollala@smtp.server.tld"
  }
}
```

### setContent

!!! post "POST /v2/config" This call can be used to overwrite a configuration file content in a free way.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| content | string | Required | The new content to write to the configuration file. |
| cluster_id | integer | Required | The ID of the cluster. |
| hostname | string | Required | The hostname of the node whose configuration file is being updated. |
| filename | string | Required | The full path to the configuration file to overwrite. |
| port | integer | Required | The port of the node. |

```
{
    "content": "[mysql]\\nport=322\\nlisten_address=\"0.0.0.0\"\\n\\n",
    "cluster_id": 114,
    "hostname": "192.168.1.127",
    "operation": "setContent",
    "filename": "/etc/my.cnf",
    "port": 9555,
    "request_created": "2023-02-06T12:06:53.093Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| messages | array | This call can send multiple messages back. Changing the configuration is a complex step and may result in more than one messages. |

**Response example**

```
{
    "messages": [ "Node restart is required for change to take effect." ],
    "request_created": "2017-07-28T12:06:53.093Z",
    "request_id": 3,
    "request_processed": "2017-07-28T12:06:53.157Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### addIntegration

!!! post "POST /v2/config" This call can be used to add a cmon-events (clustercontrol-notifications package) external integration.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `integration` | object | Required | Object describing the integration to add. |
| `integration.name` | string | Required | Name of the integration. |
| `integration.service_id` | string | Required | Service identifier for the integration (e.g., `webhook`). |
| `integration.is_active` | boolean | Required | Whether the integration is active. |
| `integration.config` | string | Required | JSON-encoded configuration string for the integration. |

```
{
   "operation": "addintegration",
   "integration": {
        "name": "webhook",
        "service_id": "webhook",
        "is_active": true,
        "config": "{\
↪"clusters\":[-1],\"triggers\":[{\"trigger\":\"ALL\",\"severity\":\"ALL\"}],\"name\":\"w
   }
}
```

### updateIntegration

!!! post "POST /v2/config" This call can be used to update (by id) a cmon-events (clustercontrol-notifications package) external integration.

The other operation is 'updateIntegration' then you must supply an existing 'id' otherwise the request will fail.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `integration` | object | Required | Object describing the integration to update. |
| `integration.id` | integer | Required | The ID of the existing integration to update. |
| `integration.name` | string | Required | Name of the integration. |
| `integration.service_id` | string | Required | Service identifier for the integration (e.g., `webhook`). |
| `integration.is_active` | boolean | Required | Whether the integration is active. |
| `integration.config` | string | Required | JSON-encoded configuration string for the integration. |


```
{
   "operation": "updateintegration",
   "integration": {
       "id": 1,
       "name": "webhook",
       "service_id": "webhook",
       "is_active": true,
       "config": "{\
↪"clusters\":[-1],\"triggers\":[{\"trigger\":\"ALL\",\"severity\":\"ALL\"}],\"name\":\"w
   }
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `integration` | object | The updated integration object. |
| `integration.id` | integer | The ID of the integration. |
| `integration.name` | string | Name of the integration. |
| `integration.service_id` | string | Service identifier for the integration. |
| `integration.is_active` | boolean | Whether the integration is active. |
| `integration.config` | string | JSON-encoded configuration string for the integration. |

**Response example**


```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-05-05T12:16:31.637Z",
  "request_status": "Ok",
  "debug_messages":
  [
      "RPC V2 authenticated user is 'alex'."
  ],
```

```
  "integration":
  {
      "name": "webhook",
      "config": "{\
↪"clusters\":[-1],\"triggers\":[{\"trigger\":\"ALL\",\"severity\":\"ALL\"}],\"name\":\"
↪
      "id": 1,
      "is_active": true,
      "service_id": "webhook"
  }
}
```

### removeIntegration

!!! post "POST /v2/config" This call can be used to remove an external integration.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| id | integer | Required | The ID of the integration to remove. |

```
{
   "operation": "removeintegration",
   "id": 1
}
```

### Response

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-05-05T12:16:31.637Z",
  "request_status": "Ok",
  "debug_messages":
  [
      "RPC V2 authenticated user is 'alex'."
  ]
}
```

### listIntegrations

!!! post "POST /v2/config" This call can be used to list external integrations, you may use pagination (limit, offset) arguments here.

### Request

```
{
   "operation": "listintegrations"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| total | integer | Total number of integrations. |
| integrations | array | List of integration objects. |
| integrations[].id | integer | The ID of the integration. |
| integrations[].name | string | Name of the integration. |
| integrations[].service_id | string | Service identifier for the integration. |
| integrations[].is_active | boolean | Whether the integration is active. |
| integrations[].config | string | JSON-encoded configuration string for the integration. |

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-05-05T12:16:31.637Z",
  "request_status": "Ok",
  "total": 1,
  "debug_messages":
  [
      "RPC V2 authenticated user is 'alex'."
  ],
  "integrations":
  [
      {
          "name": "webhook",
          "config": "{\
↪"clusters\":[-1],\"triggers\":[{\"trigger\":\"ALL\",\"severity\":\"ALL\"}],\"name\":\"
↪
          "id": 1,
          "is_active": false,
          "service_id": "webhook"
      }
  ]
}
```

### getcomponentmeta

!!! post "POST /v2/config" A method to obtain human readable description of e-mail notification components.

### Request

```
{
   "operation": "getcomponentmeta"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| total | integer | Total number of component metadata entries. |
| data | array | List of component metadata objects. |
| data[].alarm_name | string | Human-readable name for alarms of this component. |
| data[].component | string | Internal component identifier. |
| data[].log_name | string | Human-readable name for logs of this component. |
| data[].message_name | string | Human-readable description of messages for this component. |

**Response example**


```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-05-05T12:16:31.637Z",
  "request_status": "Ok",
  "total": 1,
  "debug_messages":
  [
      "RPC V2 authenticated user is 'alex'."
  ],
  "data":
  [
    {
          "alarm_name": "Network alarms",
          "component": "Network",
          "log_name": "Network related logs",
          "message_name": "Network related messages, e.g host unreachable,␣
↪SSH issues."
    },
    {
          "alarm_name": "Cmon Internal alarms",
          "component": "CmonDatabase",
          "log_name": "Cmon database logs",
          "message_name": "Internal Cmon database related messages."
     }
```

```
    ]
}
```

### getrecipients

!!! post "POST /v2/config" Obtain the recipients list.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster to get recipients for. |

```
{
    "operation": "getrecipients",
    "cluster_id": 101
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of recipients. |
| `data` | array | List of recipient email address strings. |

**Response example**

```
{
  "request_processed": "2023-05-19T15:03:23.080Z",
  "request_status": "Ok",
  "total": 3,
  "data":
  [
      "testing1000@aaaa.bbb",
      "testing100@aaaa.bbb",
      "testing10@aaaa.bbb"
  ]
}
```

### setnotificationsettings

!!! post "POST /v2/config" Sets/updates the notification settings for the specified email.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `daily_limit` | integer | Required | Maximum number of emails per day. |
| `digest_hour` | integer | Required | Hour of the day to send digest emails. |
| `email` | string | Required | The recipient email address to configure. |
| `settings` | object | Required | Notification settings per component and severity. |
| `time_zone` | integer | Required | Timezone offset in hours. |

```
{
  "cluster_id": 200,
  "daily_limit": 1000,
  "digest_hour": 7,
  "email": "testing1000@aaaa.bbb",
  "operation": "setnotificationsettings",
  "settings":
  {
      "Cluster":
      {
          "CRITICAL": "Ignore",
          "INFO": "Ignore",
          "WARNING": "Ignore"
      },
      "Network":
      {
          "CRITICAL": "Digest",
          "INFO": "Digest",
          "WARNING": "Digest"
      }
  },
  "time_zone": 0
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of notification setting entries returned. |
| `data` | array | List of notification setting objects for the recipient. |
| `data[].daily_limit` | integer | Maximum number of emails per day. |
| `data[].digest_hour` | integer | Hour of the day to send digest emails. |
| `data[].recipient` | string | The recipient email address. |
| `data[].time_zone` | integer | Timezone offset in hours. |
| `data[].settings` | object | Notification settings per component and severity. |

**Response example**


```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-05-05T12:16:31.637Z",
  "request_status": "Ok",
  "total": 1,
  "data":
  [
      {
          "daily_limit": 1000,
          "digest_hour": 7,
          "recipient": "testing1000@aaaa.bbb",
          "time_zone": 0,
          "settings":
          {
              "Backup":
              {
                  "CRITICAL": "Digest",
                  "INFO": "Digest",
                  "WARNING": "Digest"
              },
              "Cluster":
              {
                  "CRITICAL": "Ignore",
                  "INFO": "Ignore",
                  "WARNING": "Ignore"
              },
              "ClusterConfiguration":
              {
                  "CRITICAL": "Digest",
                  "INFO": "Digest",
                  "WARNING": "Digest"
              },
              "ClusterRecovery":
              {
                  "CRITICAL": "Deliver",
                  "INFO": "Digest",
                  "WARNING": "Digest"
              },
              "CmonDatabase":
              {
                  "CRITICAL": "Digest",
                  "INFO": "Digest",
                  "WARNING": "Digest"
              },
              "DbHealth":
              {
                  "CRITICAL": "Digest",
                  "INFO": "Digest",
```

```
                "WARNING": "Digest"
            },
            "DbPerformance":
            {
                "CRITICAL": "Digest",
                "INFO": "Digest",
                "WARNING": "Digest"
            },
            "Host":
            {
                "CRITICAL": "Digest",
                "INFO": "Digest",
                "WARNING": "Digest"
            },
            "Mail":
            {
                "CRITICAL": "Digest",
                "INFO": "Digest",
                "WARNING": "Digest"
            },
            "Network":
            {
                "CRITICAL": "Digest",
                "INFO": "Digest",
                "WARNING": "Digest"
            },
            "Node":
            {
                "CRITICAL": "Deliver",
                "INFO": "Digest",
                "WARNING": "Digest"
            },
            "SoftwareInstallation":
            {
                "CRITICAL": "Digest",
                "INFO": "Deliver",
                "WARNING": "Digest"
            },
            "Unknown":
            {
                "CRITICAL": "Digest",
                "INFO": "Digest",
                "WARNING": "Digest"
            }
        }
    }
  ]
}
```

### getnotificationsettings

!!! post "POST /v2/config" Returns the notification settings of a specified recipient.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| email | string | Required | The recipient email address to retrieve settings for. |
| cluster_id | integer | Required | The ID of the cluster. |

```
{
   "operation": "getnotificationsettings",
   "email": "testing1000@aaaa.bbb",
   "cluster_id": 101
}
```

### Response

| Field | Type | Description |
|---|---|---|
| total | integer | Total number of notification setting entries returned. |
| data | array | List of notification setting objects for the recipient. |
| data[].daily_limit | integer | Maximum number of emails per day. |
| data[].digest_hour | integer | Hour of the day to send digest emails. |
| data[].recipient | string | The recipient email address. |
| data[].time_zone | integer | Timezone offset in hours. |
| data[].settings | object | Notification settings per component and severity. |

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-05-05T12:16:31.637Z",
  "request_status": "Ok",
  "total": 1,
  "data":
  [
      {
          "daily_limit": 1000,
          "digest_hour": 7,
          "recipient": "testing1000@aaaa.bbb",
          "time_zone": 0,
          "settings":
          {
              "Backup":
              {
                  "CRITICAL": "Digest",
```

```
            "INFO": "Digest",
            "WARNING": "Digest"
        },
        "Cluster":
        {
            "CRITICAL": "Ignore",
            "INFO": "Ignore",
            "WARNING": "Ignore"
        },
        "ClusterConfiguration":
        {
            "CRITICAL": "Digest",
            "INFO": "Digest",
            "WARNING": "Digest"
        },
        "ClusterRecovery":
        {
            "CRITICAL": "Deliver",
            "INFO": "Digest",
            "WARNING": "Digest"
        },
        "CmonDatabase":
        {
            "CRITICAL": "Digest",
            "INFO": "Digest",
            "WARNING": "Digest"
        },
        "DbHealth":
        {
            "CRITICAL": "Digest",
            "INFO": "Digest",
            "WARNING": "Digest"
        },
        "DbPerformance":
        {
            "CRITICAL": "Digest",
            "INFO": "Digest",
            "WARNING": "Digest"
        },
        "Host":
        {
            "CRITICAL": "Digest",
            "INFO": "Digest",
            "WARNING": "Digest"
        },
        "Mail":
        {
            "CRITICAL": "Digest",
            "INFO": "Digest",
            "WARNING": "Digest"
```

```
                },
                "Network":
                {
                    "CRITICAL": "Digest",
                    "INFO": "Digest",
                    "WARNING": "Digest"
                },
                "Node":
                {
                    "CRITICAL": "Deliver",
                    "INFO": "Digest",
                    "WARNING": "Digest"
                },
                "SoftwareInstallation":
                {
                    "CRITICAL": "Digest",
                    "INFO": "Deliver",
                    "WARNING": "Digest"
                },
                "Unknown":
                {
                    "CRITICAL": "Digest",
                    "INFO": "Digest",
                    "WARNING": "Digest"
                }
            }
        }
    ]
}
```

### removerecipient

!!! post "POST /v2/config" Removes an e-mail recipient.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| email | string | Required | The email address of the recipient to remove. |
| cluster_id | integer | Required | The ID of the cluster. |

```
{
    "operation": "removerecipient",
    "email": "someuser@mailhost.tld",
    "cluster_id": 101
}
```

### Response

**Response example**


```
{
  "request_processed": "2023-05-19T15:03:23.080Z",
  "request_status": "Ok"
}
```


## 4.12.11 Controllers and HA

The `/v2/controller` path is for managing controller and operate the Cmon HA subsystem.

| Operation | Description |
|---|---|
| *ping* | Check if a controller is reachable without locking the HA subsystem. |
| *JoinRequest* | Internal HA call for a follower to request joining the controller pool. |
| *HeartBeat* | Internal HA call exchanged between controllers to maintain cluster membership. |
| *appendEntries* | Internal Raft call to replicate log entries from leader to followers. |
| *requestVote* | Internal Raft call sent during leader election to request votes. |

### ping

!!! post "POST /v2/controller" This call is created so that the client application can check on a controller even when that controller is not the leader. This call should be as fast as possible and should not do any locking on the Cmon HA subsystem.

### Request


```
{
    "operation": "ping",
    "request_created": "2020-01-02T10:01:08.035Z",
    "request_id": 1
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `cmon_ha_supported` | boolean | Whether Cmon HA is supported on this controller. |
| `controller_status` | string | Current status of the controller (e.g. `CmonControllerLeader`). |
| `term` | integer | Current Raft term number. |
| `prev_log_index` | integer | Index of the last log entry. |
| `prev_log_term` | integer | Term of the last log entry. |

**Response example**

```
{
    "cmon_ha_supported": true,
    "controller_status": "CmonControllerLeader",
    "term": 0,
    "prev_log_index": 0,
    "prev_log_term": 0,
    "request_created": "2020-01-02T10:01:08.035Z",
    "request_id": 1,
    "request_processed": "2020-01-02T10:01:08.044Z",
    "request_status": "Ok"
}
```


### JoinRequest

!!! post "POST /v2/controller" An internal HA call used by a follower controller to request joining the controller pool managed by the leader. This is part of the CMON HA protocol and is not intended for direct use by external clients.

### HeartBeat

!!! post "POST /v2/controller" An internal HA call exchanged between controllers to maintain cluster membership and detect node failures. The payload includes the current leader, committed log index, and known controller membership. This is part of the CMON HA protocol and is not intended for direct use by external clients.

### Request


```
{
    "client_ip_address": "192.168.0.127",
    "controller_key": "6a03558f-5d7a-41ed-9af1-9e0ff162bcb8",
    "failed_controllers":
    {
    },
    "follower_controllers":
    {
        "192.168.0.127:10001":
        {
            "class_name": "CmonControllerHost",
            "hostname": "192.168.0.127",
            "ip": "192.168.0.127",
            "port": 10001
        },
        "192.168.0.127:20001":
        {
            "class_name": "CmonControllerHost",
            "hostname": "192.168.0.127",
            "ip": "192.168.0.127",
            "port": 20001
        }
    },
```


(continues on next page)

(continued from previous page)

```
    "committed_index": 6,
    "leader_controller":
    {
        "class_name": "CmonControllerHost",
        "hostname": "192.168.0.127",
        "ip": "192.168.0.127",
        "port": 9556
    },
    "operation": "heartBeat",
    "request_created": "2019-03-19T09:43:58.444Z",
    "term": 0
}
```

### Response

| Field | Type | Description |
|---|---|---|
| cmon_ha_supported | boolean | Whether Cmon HA is supported on this controller. |
| pid | integer | Process ID of the Cmon controller. |
| wallclock | integer | Unix timestamp when the response was generated. |

**Response example**

```
{
    "cmon_ha_supported": true,
    "pid": 41918,
    "request_created": "2019-03-19T09:43:58.444Z",
    "request_processed": "2019-03-19T09:43:58.489Z",
    "request_status": "Ok",
    "wallclock": 1552988638
}
```

### appendEntries

!!! post “POST /v2/controller” An internal Raft consensus call used to replicate log entries from the leader controller to its followers. Each entry carries a payload instruction — such as saving a configuration file — that the follower applies locally. This is part of the CMON HA replication protocol and is not intended for direct use by external clients.

### Request

```
{
    "client_ip_address": "192.168.0.127",
    "controller_key": "6a03558f-5d7a-41ed-9af1-9e0ff162bcb8",
    "log_entry":
    {
        "class_name": "CmonHaLogEntry",
```

(continues on next page)

(continued from previous page)


```
        "comment": "",
        "committed": false,
        "executed": false,
        "index": 11,
        "prev_log_index": 10,
        "prev_log_term": 0,
        "payload":
        {
            "files": [
            {
                "class_name": "CmonFile",
                "content":
↪"basedir=/usr\ncdt_path=/\ncluster_id=1\ncmon_db=ft_install_testdb\ncmon_user=root\ncre
↪'password'\
↪nmysql_bindir=/usr/bin/\nmysql_hostname=127.0.0.1\nmysql_password='p'\
↪nmysql_port=3306\nmysql_server_addresses=192.168.0.232:3306\nmysql_version=5.6\nname=
↪'ft_cmonhabasic_41388'\
↪nos=debian\nowner=1\npidfile=/var/run\nquery_monitor_use_local_settings=false\nssh_iden
↪-n 2>/dev/null\ntype=galera\nvendor=percona\n",
                "full_path": "/tmp/cmon_1.cnf"
            } ],
            "instruction": "SaveLocalFile"
        },
        "term": 0
    },
    "operation": "appendEntries",
    "request_created": "2019-03-19T09:48:32.864Z"
}
```


### Response

**Response example**


```
{
    "request_created": "2019-03-19T09:48:32.864Z",
    "request_processed": "2019-03-19T09:48:32.909Z",
    "request_status": "Ok"
}
```


### requestVote

!!! post "POST /v2/controller" An internal Raft consensus call sent during leader election. A candidate controller sends this to request votes from the other controllers in the pool. This is part of the CMON HA protocol and is not intended for direct use by external clients.

### Request


```
{
    "client_ip_address": "192.168.0.127",
    "prev_log_index": 23,
    "prev_log_term": 42,
    "operation": "requestVote",
    "request_created": "2019-02-28T07:32:22.095Z",
    "term": 43
}
```


## 4.12.12 Discovery

The `/v2/discovery` path provides calls for collecting real-time data about nodes and discovering cluster resources.

| Operation | Description |
|---|---|
| *checkClusterName* | Check whether a cluster name is acceptable for use when creating a new cluster. |
| *getSupportedClusterTypes* | Get metadata about all cluster types supported by the controller. |
| *getSshCredentials* | Get the SSH credentials the controller uses to authenticate on nodes of an existing cluster. |
| *checkHosts* | Run connectivity and hardware tests against a list of hosts. |

### checkClusterName

!!! post "POST /v2/discovery" This call can be used to check if a specific cluster name is acceptable to be used to create a new cluster.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| new_cluster_name | string | Required | The new cluster name the user plans to use to name a new cluster. |

```
{
    "new_cluster_name": "Cluster_01",
    "operation": "checkClusterName",
    "request_created": "2017-08-15T09:32:29.823Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| is_accep | boole | Shows if the cluster name is acceptable as a name for a new cluster. If the name is not valid for this purpose (e.g. the name already used for some other cluster) this value will be false. |
| mes-sage | string | Shows why the name is not acceptable. This string is human readable, it can be shown to the user. |
| re-quest_st | string | Please note that the request status will be "Ok" even if the reply is negative, when the new name is not acceptable. This field shows if the request is processed, it is not the reply. |

**Response example**

```
{
    "is_acceptable": true,
    "message": "The cluster name 'Cluster_01' is acceptable.",
    "request_created": "2017-08-15T09:32:29.823Z",
    "request_id": 3,
    "request_processed": "2017-08-15T09:32:29.868Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### getSupportedClusterTypes

!!! post "POST /v2/discovery" The getSupportedClusterTypes returns some metadata about the cluster types supported by the controller. This is a very important call, using it the GUI can fill up comboboxes, lists and can make sure that the cluster creation will not fail because the user requested some invalid combination. It is also worth mentioning that if the UI uses this call the UI source doesn't need to be changed when we start supporting a new version or vendor.

Here is how the call look like:

### Request

```
{
    "operation": "getSupportedClusterTypes",
    "request_created": "2017-08-15T08:37:30.150Z",
    "request_id": 3
}
```

### Response

And here is the reply. Please note that this call is under construction, we might add new fields if we find some other important information.

**Response example**

```
{
    "cluster_type_names": [ "galera", "replication", "groupreplication",
↪"myqlcluster", "postgresql", "mongodb" ],
    "cluster_type_properties":
    {
        "galera":
        {
            "intro_url": "http://galeracluster.com/products/",
            "long_name": "Galera Cluster for MySQL",
            "short_name": "Galera",
            "type_name": "galera",
            "vendors": [
            {
                "name": "mariadb",
                "versions": [ "5.5", "10.1", "10.2" ]
            },
            {
                "name": "codership",
                "versions": [ "5.5", "5.6" ]
            },
            {
                "name": "percona",
                "versions": [ "5.5", "5.6", "5.7" ]
            } ]
        },
        "groupreplication":
        {
            "intro_url":
↪"http://mysqlhighavailability.com/gr/doc/index.html",
            "long_name": "MySQL Group Replication",
            "short_name": "Group Replication",
            "type_name": "groupreplication",
            "vendors": [
            {
                "name": "oracle",
                "versions": [ "5.7" ]
```

```
            } ]
        },
        "mongodb":
        {
            "intro_url": "https://www.mongodb.com/what-is-mongodb",
            "long_name": "MongoDB Cluster",
            "short_name": "MongoDB",
            "type_name": "mongodb",
            "vendors": [
            {
                "name": "10gen",
                "versions": [ "3.2", "3.4" ]
            },
            {
                "name": "percona",
                "versions": [ "3.2" ]
            } ]
        },
        "myqlcluster":
        {
            "long_name": "MySQL NDB Cluster",
            "nick_name": "ndb",
            "short_name": "NDB Cluster",
            "type_name": "myqlcluster",
            "vendors": [
            {
                "name": "oracle",
                "versions": [ "7.4" ]
            } ]
        },
        "postgresql":
        {
            "intro_url": "https://www.postgresql.org/about/",
            "long_name": "PostgreSQL Cluster",
            "short_name": "PostgreSQL",
            "type_name": "postgresql",
            "vendors": [
            {
                "name": "postgresql",
                "versions": [ "9.4", "9.5", "9.6", "10" ]
            } ]
        },
        "replication":
        {
            "long_name": "MySQL Replication",
            "nick_name": "mysqlreplication",
            "short_name": "Replication",
            "type_name": "replication",
            "vendors": [
            {
```

```
                "name": "mariadb",
                "versions": [ "5.5", "10.1", "10.2" ]
            },
            {
                "name": "codership",
                "versions": [ "5.5", "5.6" ]
            },
            {
                "name": "percona",
                "versions": [ "5.5", "5.6", "5.7" ]
            } ]
        }
    },
    "request_created": "2017-08-15T08:37:30.150Z",
    "request_id": 3,
    "request_processed": "2017-08-15T08:37:30.200Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 6
}
```


### getSshCredentials

!!! post "POST /v2/discovery" Using the getSshCredentials the caller can get the SSH credentials used when the controller authenticates on the nodes of the cluster. This cluster is made for the UI that creates a new cluster: here the user can check and re-use the SSH credentials used on some existing cluster.

The privilege check provides access if the authenticated user is either the system user or the owner of the existing cluster. No ACLs are implemented and other than the owner can't access the sensitive information of the existing clusters.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Optional | The numerical ID of the cluster to read. The cluster name also can be used to identify the cluster. |
| `cluster_name` | string | Optional | The name of the cluster to read. The cluster ID can also be used to identify the cluster. |


```
{
    "cluster_id": 1,
    "operation": "getSshCredentials",
    "request_created": "2017-08-14T08:25:20.298Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| cluster_id | integer | The numerical ID of the cluster is always sent back even if the cluster was identified by its name in the request. |
| ssh_credenti | object | The credentials used to authenticate on the nodes of the cluster. |

**Response example**


```
{
    "cluster_id": 1,
    "request_created": "2017-08-14T08:25:20.298Z",
    "request_id": 3,
    "request_processed": "2017-08-14T08:25:20.344Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "ssh_credentials":
    {
        "class_name": "CmonSshCredentials",
        "password": "",
        "port": 22,
        "ssh_keyfile": "./configs/testing_id_rsa",
        "timeout": 30,
        "tty_for_sudo": true,
        "user_name": "pipas"
    }
}
```


### checkHosts

!!! post "POST /v2/discovery" The checkHosts request performs a set of tests on the given hosts and returns various hardware information about the hosts themselves. The call can specify what kind of tests it needs at the given stage, e.g. if the user already provided an ssh password the access to the host through SSH can be checked.

Here is how these checks can be performed using the s9s command line tool:

```
s9s cluster --check-hosts --nodes="192.168.1.113" --cluster-type=galera --
↪provider-version=5.7 --vendor=codership
```

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| check | boole | Optional | If true the controller will check if the hosts are already registered for other clusters. If the port is set the code will check if the same host with the same name is already registered in some other cluster. The same port on the same host can't be used twice and the test will check this. If the class_name is set the backend will perform another check — we do not support running two instances of the same daemon on one single host. Supported class names: CmonHost (generic host), CmonMySqlHost, CmonPostgreSqlHost, CmonProxySqlHost, CmonHaProxyHost. |
| check | boole | Optional | This check will see that the host is actually accessible through ssh as superuser. For this test either the SSH credentials or an existing cluster must be specified. |
| check | boole | Optional | When this property is true the caller can send a "job" field with the "create_cluster" job for a preliminary check. The job will not be executed, but some checks will be performed to see if the job with the passed properties is valid. |
| cluster_i | integer | Optional | The cluster can be specified through the numerical ID of the cluster. If the cluster is specified the SSH credentials can be accessed from the cluster configuration, so the user don't need to enter those. |
| cluster_n | string | Optional | Same as cluster_id, but here the cluster can be referenced by the cluster name. |
| ssh_c | object | Optional | Fields providing SSH access to the nodes. The SSH credentials can come from the existing cluster and from this field, but this field has precedence. |

Here is an example of the request:

```
{
    "check_if_already_registered": true,
    "nodes": [
    {
        "class_name": "CmonHost",
        "hostname": "192.168.1.212",
        "port": 8089
    } ],
    "operation": "checkHosts",
    "request_created": "2017-08-11T12:32:58.547Z",
    "request_id": 3
}
```

Here is another request that is slightly different. Note that this request will not check if the host is already registered simply because the "check_if_already_registered" field is not set to true. This request does not identify an existing cluster either, it contains the "ssh_credentials" field so that the controller can gain access to the nodes. Should the cluster be also referenced the "ssh_credentials" have priority over the credentials found in an existing cluster.

```
{
    "check_ssh_sudo": true,
    "nodes": [
    {
```

(continues on next page)

(continued from previous page)

```
        "class_name": "CmonHost",
        "hostname": "192.168.1.113",
        "port": 8089
    } ],
    "operation": "checkHosts",
    "request_created": "2017-08-14T10:48:50.978Z",
    "request_id": 4,
    "ssh_credentials":
    {
        "class_name": "CmonSshCredentials",
        "password": "",
        "ssh_keyfile": "./configs/testing_id_rsa",
        "user_name": "pipas"
    }
}
```

And here is a call that will also check the passed "create_cluster" job:

```
{
    "check_if_already_registered": true,
    "check_job": true,
    "check_ssh_sudo": true,
    "job":
    {
        "class_name": "CmonJobInstance",
        "job_spec":
        {
            "command": "create_cluster",
            "job_data":
            {
                "cluster_type": "group_replication",
                "mysql_version": "",
                "nodes": [
                {
                    "class_name": "CmonHost",
                    "hostname": "192.168.1.113"
                } ],
                "vendor": ""
            }
        }
    },
    "nodes": [
    {
        "class_name": "CmonHost",
        "hostname": "192.168.1.113"
    } ],
    "operation": "checkHosts",
    "request_created": "2017-10-10T07:55:59.653Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| checked_h | array | The detailed list with the hosts checked. Every host has the status showing what was the result of the check on that particular host. |
| total | integer | The total number of hosts checked. |
| total_faile | integer | The number of hosts that failed the checks. This can be used as a quick check, but the details of the failure is in the reply too. |

The `error_code` field of the `status` can be one of the following strings:

- `HostFoundInOtherCluster`: The host is already part of some other cluster.
- `SshConnectionFailed`: The SSH connection failed.
- `SshAuthenticationFailed`: The SSH connected, but the authentication failed.
- `SudoFailed`: Failed to gain root access.
- `InvalidRequest`: The request is invalid, check could not be performed.
- `JobWouldFail`: The job would fail if it is sent seriously.
- `HostIsOk`: The host passed all tests.

When the `check_ssh_sudo` test is performed the reply shows some information about the hosts checked. Please note that these tests are under development, later more information might be added.

The response fields `hardware/cpu`, `hardware/memory`, `hardware/mounted_partitions`, `hardware/network_interfaces`, `software/firewall`, and `software/os_version` provide hardware and software details about the checked hosts.

- `hardware/cpu`: Some information about the CPU(s) found in the host. This can be shown to the user to help recognize the hardware. A total can also be calculated for the cluster so that the user can see what resources the cluster will have.
- `hardware/memory`: Information about the memory configuration.
- `hardware/mounted_partitions`: Very important piece of information that shows the available storage capacity.
- `hardware/network_interfaces`: This actually could show more, like we could show the type of connection.
- `software/firewall`: Shows what firewall software is installed.
- `software/os_version`: The distribution and its version.

Well, it is clear, we need to send more info, but the basics are there. Takes a lot of work add new things here.

**Response example (simple)**

```
{
    "checked_hosts": [
    {
        "host":
        {
            "class_name": "CmonHost",
            "hostname": "192.168.1.112",
            "port": 8089
        },
        "status":
        {
            "error_code": "HostIsOk"
        }
    } ],
    "request_created": "2017-08-11T13:49:10.052Z",
    "request_id": 3,
    "request_processed": "2017-08-11T13:49:10.097Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 1,
    "total_failed": 0
}
```


**Response example (with hardware/software details)**


```
{
    "checked_hosts": [
    {
        "hardware":
        {
            "cpu":
            {
                "class_name": "CmonCpuSet",
                "cpus": [
                {
                    "cores": 4,
                    "cpu_max_ghz": 2.268,
                    "id": 0,
                    "model": "Intel(R) Xeon(R) CPU L5520 @ 2.27GHz",
                    "siblings": 8,
                    "vendor": "GenuineIntel"
                },
                {
                    "cores": 4,
                    "cpu_max_ghz": 2.268,
                    "id": 1,
                    "model": "Intel(R) Xeon(R) CPU L5520 @ 2.27GHz",
                    "siblings": 8,
                    "vendor": "GenuineIntel"
```

```
            } ],
            "total_cores": 8,
            "total_cpus": 2,
            "total_siblings": 16
        },
        "memory":
        {
            "class_name": "CmonMemoryInfo",
            "memory_available_mb": 16085,
            "memory_free_mb": 16085,
            "memory_total_mb": 16384,
            "swap_free_mb": 0,
            "swap_total_mb": 0
        },
        "mounted_partitions": [
        {
            "device": "/dev/mapper/core1--vg-root",
            "filesystem": "ext4",
            "free_gb": 142.631,
            "free_mb": 146053,
            "mountpoint": "/",
            "total_gb": 162.286,
            "total_mb": 166180
        } ],
        "network_interfaces":
        {
            "eth0": "192.168.1.113"
        }
    },
    "host":
    {
        "class_name": "CmonHost",
        "hostname": "192.168.1.113",
        "port": 8089
    },
    "software":
    {
        "firewall":
        {
            "firewalld_installed": false,
            "ip_tables_installed": false,
            "iptables_installed": false,
            "ufw_installed": false
        },
        "os_version":
        {
            "class_name": "CmonOsVersion",
            "codename": "xenial",
            "name": "ubuntu",
            "release": "16.04",
```

```
                "type": "debian"
            },
            "security":
            {
                "apparmor_installed": false,
                "selinux_installed": false,
                "sudo_installed": true
            }

        },
        "status":
        {
            "error_code": "HostIsOk"
        }
    } ],
    "request_created": "2017-08-16T10:21:19.122Z",
    "request_id": 4,
    "request_processed": "2017-08-16T10:21:20.571Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 1,
    "total_failed": 0
}
```


### 4.12.13 Hosts and Containers

The `/v2/host` path provides calls for managing hosts and containers monitored by the controller.

| Operation | Description |
|---|---|
| *startServers* | Start (boot) one or more servers. |
| *shutDownServers* | Shut down one or more servers. |
| *registerServers* | Register servers with the controller so they can be monitored. |
| *unregisterHost* | Unregister a single host from the controller. |
| *unregisterServers* | Unregister multiple servers from the controller. |
| *getContainers* | Get information about containers managed by the controller. |
| *getServers* | Get information about servers monitored by the controller. |

#### startServers

!!! post "POST /v2/host" This call is for starting up (or booting) servers.

#### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `servers` | array | Required | A list of server objects to start. Each object should include `class_name` and `hostname`. |

```
{
    "operation": "startServers",
    "request_created": "2017-10-16T07:48:21.779Z",
    "request_id": 3,
    "servers": [
    {
        "class_name": "CmonContainerServer",
        "hostname": "host04"
    } ]
}
```

### Response

**Response example**

```
{
    "messages": [ "Started server 'host04'." ],
    "request_created": "2017-10-16T07:48:21.779Z",
    "request_id": 3,
    "request_processed": "2017-10-16T07:48:21.943Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

## shutDownServers

!!! post "POST /v2/host" Shuts down (powers off) one or more container servers. The servers must be registered with the controller. This is the counterpart to `startServers`.

## registerServers

!!! post "POST /v2/host" This call can be used to register a container server in the Cmon Controller. This is the fast way, but there is a job called "create_server" that does a very similar registration. The job will actually install software, this call just registers an existing server.

Here is how a request will be sent:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `servers` | array | Required | A list of server objects to register. Each object should include `class_name`, `cdt_path`, and `hostname`. |

```
{
    "operation": "registerServers",
    "request_created": "2017-08-31T07:53:41.158Z",
```

(continues on next page)

(continued from previous page)

```
    "request_id": 3,
    "servers": [
    {
        "class_name": "CmonContainerServer",
        "cdt_path": "myservers",
        "hostname": "storage01"
    } ]
}
```

### Response

Here is a reply that shows the server was registered, but the server is actually turned off:

| Field | Type | Description |
|---|---|---|
| `servers` | array | A list of registered server objects with their current state. |

**Response example**

```
{
    "reply_received": "2017-10-16T11:33:24.501Z",
    "request_created": "2017-10-16T11:33:24.496Z",
    "request_id": 3,
    "request_processed": "2017-10-16T11:33:27.606Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "servers": [
    {
        "cdt_path": "/",
        "class_name": "CmonContainerServer",
        "clusterid": 0,
        "connected": false,
        "hostId": 6,
        "hostname": "storage01",
        "hoststatus": "CmonHostOffLine",
        "ip": "192.168.1.17",
        "maintenance_mode_active": false,
        "message": "SSH connection failed.",
        "owner_group_id": 2,
        "owner_group_name": "users",
        "owner_user_id": 3,
        "owner_user_name": "pipas",
        "protocol": "lxc",
        "timestamp": 1508153607,
        "unique_id": 6
    } ]
}
```

### unregisterHost

!!! post "POST /v2/host" This call will drop the given host from our database entirely. The computer the host represents will not be touched in any way, software will not be uninstalled, service will not be stopped, nada!

So there are multiple fields to identify the host, there is teh hostname, the port, the cluster ID, maybe the cluster name. The backend will use whatever it is provided and find the host. If multiple hosts are found with the given data the request will be rejected.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Optional | The numerical ID of the cluster that will execute the job. The cluster can also be referenced using the cluster name. For RPC v2 identifying the cluster is not mandatory, but it can be usefull if the same host is part of multiple clusters. |
| `cluster_na` | string | Optional | The name of the cluster that will execute the job. The cluster can also be referenced using the cluster ID. For RPC v2 identifying the cluster is not mandatory, but it can be usefull if the same host is part of multiple clusters. |
| `dry_ru` | boole | Optional | If this field is provided (with any value at all) the host will not be really unregistered. All the checks will be done, a success will be reported back, only the error message will show that this was just a drill. |
| `host` | object | Require | The host that shall be unregistered. |
| `host.class_` | string | Optional | The class name of the host will not be strictly checked (because the client might not know this), just send "CmonHost" that'll do. |
| `host.hostna` | string | Require | This is mandatory to identify the host. |
| `host.port` | integer | Optional | The port of the host. RPC v2 allows the usage of hosts without a port, but for most cases the port will be required. |

```
{
    "cluster_id": 1,
    "dry_run": true,
    "host":
    {
        "class_name": "CmonHost",
        "hostname": "192.168.1.127",
        "port": 9555
    },
    "operation": "unregisterHost",
    "request_created": "2017-09-08T09:42:49.137Z",
    "request_id": 3
}
```

### Response

Here is what we get for a dry run:

**Response example**


```
{
    "error_string": "Dry run was requested, not unregistering host.",
    "request_created": "2017-09-08T09:42:49.137Z",
    "request_id": 3,
    "request_processed": "2017-09-08T09:42:49.183Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


## unregisterServers

!!! post "POST /v2/host" This call is for dropping a container server from the scope of the controller. The server will not seize to exist, but the controller will forget everything about it.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `server` | array | Required | A list of CmonContainerServer class objects to unregister. In the objects only the class name and the host name has to be provided. |


```
{
    "operation": "unregisterServers",
    "request_created": "2017-09-01T08:42:22.756Z",
    "request_id": 3,
    "servers": [
    {
        "class_name": "CmonContainerServer",
        "hostname": "core1"
    } ]
}
```


### Response

**Response example**


```
{
    "messages": [ "Unregistered server 'core1'." ],
    "request_created": "2017-09-01T08:42:22.756Z",
    "request_id": 3,
    "request_processed": "2017-09-01T08:42:22.810Z",
```

```
    "request_status": "Ok",
    "request_user_id": 3
}
```

### getContainers

!!! post "POST /v2/host" Returns the list of containers known by the controller. The RPC v2 call also checks the rights of the authenticated user, if the user has no right to read the container object it will not appear in the list.

### Request

```
{
    "operation": "getContainers",
    "request_created": "2017-09-07T05:09:16.895Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| containers | array | A list of #CmonContainer objects. The following command can be used to print the properties of this class: `s9s metatype --list-properties --type=CmonContainer --long` |
| total | integer | The total number of containers known by the controller. |

**Response example**

```
{
    "containers": [
    {
        "alias": "debian",
        "class_name": "CmonContainer",
        "hostname": "debian",
        "owner_group_id": 4,
        "owner_group_name": "testgroup",
        "owner_user_id": 3,
        "owner_user_name": "pipas",
        "parent_server": "storage01",
        "status": "STOPPED",
        "type": "lxc"
    },
    . . .
    {
        "alias": "vnc_server",
```

```
        "class_name": "CmonContainer",
        "hostname": "192.168.1.210",
        "ip": "192.168.1.210",
        "ipv4_addresses": [ "192.168.1.210" ],
        "owner_group_id": 4,
        "owner_group_name": "testgroup",
        "owner_user_id": 3,
        "owner_user_name": "pipas",
        "parent_server": "storage01",
        "status": "RUNNING",
        "template": "ubuntu",
        "type": "lxc"
    } ],
    "request_created": "2017-09-07T05:09:16.895Z",
    "request_id": 3,
    "request_processed": "2017-09-07T05:09:16.947Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 5
}
```


### getServers

!!! post "POST /v2/host" Returns the container servers and their properties.

### Request


```
{
    "operation": "getServers",
    "request_created": "2017-10-03T08:19:58.769Z",
    "request_id": 3
}
```


### Response

And here is the reply. It is rather complex, but we have a lot of data.

| Field | Type | Description |
|---|---|---|
| `servers` | array | A list of container server objects with detailed properties including containers, disk devices, distribution info, memory, network interfaces, and processors. |
| `total` | integer | The total number of servers. |

**Response example**


```
{
    "request_created": "2017-10-03T08:19:58.769Z",
```

```
    "request_id": 3,
    "request_processed": "2017-10-03T08:19:58.824Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "servers": [
    {
        "cdt_path": "/",
        "class_name": "CmonContainerServer",
        "clusterid": 0,
        "connected": false,
        "containers": [
        {
            "acl": "user::rwx,user:nobody:r--,group::rw-,other::---",
            "alias": "bestw_controller",
            "cdt_path": "/core1/containers",
            "class_name": "CmonContainer",
            "container_id": 1,
            "hostname": "192.168.1.182",
            "ip": "192.168.1.182",
            "ipv4_addresses": [ "192.168.1.182" ],
            "owner_group_id": 2,
            "owner_group_name": "users",
            "owner_user_id": 3,
            "owner_user_name": "pipas",
            "parent_server": "core1",
            "status": "RUNNING",
            "type": "lxc",
            "version": 25
        },
        . . .
        {
            "acl": "user::rwx,group::rw-,other::---",
            "alias": "ubuntu",
            "cdt_path": "/core1/containers",
            "class_name": "CmonContainer",
            "container_id": 5,
            "hostname": "ubuntu",
            "owner_group_id": 2,
            "owner_group_name": "users",
            "owner_user_id": 3,
            "owner_user_name": "pipas",
            "parent_server": "core1",
            "status": "STOPPED",
            "type": "lxc",
            "version": 25
        } ],
        "disk_devices": [
        {
            "class_name": "CmonDiskDevice",
            "device": "/dev/mapper/core1--vg-root",
```

```
            "filesystem": "ext4",
            "free_mb": 141617,
            "mountpoint": "/",
            "total_mb": 166180
        },
        . . .
        {
            "device": "/dev/sdc",
            "is_hardware_storage": false,
            "model": "LSILOGIC Logical Volume",
            "total_mb": 170230,
            "volumes": [
            {
                "description": "Linux filesystem partition",
                "device": "/dev/sdc1",
                "filesystem": "ext2",
                "free_mb": 295,
                "mount_point": "/boot",
                "total_mb": 487
            },
            {
                "description": "Extended partition",
                "device": "/dev/sdc2",
                "filesystem": "",
                "mount_point": "",
                "total_mb": 169741,
                "volumes": [
                {
                    "description": "Linux LVM Physical Volume partition",
                    "device": "/dev/sdc5",
                    "filesystem": "",
                    "mount_point": "",
                    "total_mb": 169741
                } ]
            } ]
        },
        . . .
        {
            "device": "/dev/sda",
            "is_hardware_storage": false,
            "model": "SCSI Disk",
            "total_mb": 15272.1,
            "volumes": [
            {
                "description": "EXT4 volume",
                "device": "/dev/sda1",
                "filesystem": "ext4",
                "mount_point": "",
                "total_mb": 15271.1
            } ]
```

```
        } ],
        "distribution":
        {
            "codename": "xenial",
            "name": "ubuntu",
            "release": "16.04",
            "type": "debian"
        },
        "hostId": 1,
        "hostname": "core1",
        "hoststatus": "CmonHostOffLine",
        "ip": "192.168.1.4",
        "last_container_collect": 1506791691,
        "last_hw_collect": 1507017024,
        "lastseen": 1506791691,
        "maintenance_mode_active": false,
        "memory":
        {
            "banks": [
            {
                "bank": "0",
                "name": "DIMM 800 MHz (1.2 ns)",
                "size": 4294967296
            },
            . . .
            {
                "bank": "7",
                "name": "DIMM 800 MHz (1.2 ns)",
                "size": 4294967296
            } ],
            "class_name": "CmonMemoryInfo",
            "memory_available_mb": 54091,
            "memory_free_mb": 41919,
            "memory_total_mb": 64421,
            "swap_free_mb": 0,
            "swap_total_mb": 0
        },
        "message": "SSH connection failed.",
        "model": "SUN FIRE X4170 SERVER (4583256-1)",
        "network_interfaces": [
        {
            "gbits": 1,
            "link": true,
            "mac": "00:21:28:76:06:2a",
            "model": "82575EB Gigabit Network Connection",
            "name": "enp1s0f0"
        },
        . . .
        {
            "gbits": 0,
```

```
            "ip": "192.168.122.1",
            "mac": "",
            "model": "",
            "name": "virbr0"
        } ],
        "owner_group_id": 2,
        "owner_group_name": "users",
        "owner_user_id": 3,
        "owner_user_name": "pipas",
        "processors": [
        {
            "cores": 4,
            "cpu_max_ghz": 1.6,
            "id": 5,
            "model": "Intel(R) Xeon(R) CPU L5520 @ 2.27GHz",
            "siblings": 8,
            "vendor": "Intel Corp."
        },
        {
            "cores": 4,
            "cpu_max_ghz": 1.6,
            "id": 9,
            "model": "Intel(R) Xeon(R) CPU L5520 @ 2.27GHz",
            "siblings": 8,
            "vendor": "Intel Corp."
        } ],
        "timestamp": 1507017024,
        "unique_id": 1,
        "version": "2.17"
    },
    . . .
    ],
    "total": 8
}
```


### 4.12.14 Job Information and Manipulation

Creating and manipulating jobs, reading information about jobs is available on the `/v2/jobs` path.

| Operation | Description |
|---|---|
| *createJobInstance* | Create and start a new job instance on the controller |
| *updateJobInstance* | Update an existing non-closed job instance |
| *deleteJobInstance* | Delete a closed job instance record from the controller |
| *killJobInstance* | Send a signal to abort a running job |
| *cloneJobInstance* | Clone an existing job instance to re-execute it |
| *getJobInstance* | Get information about a single job instance |
| *getJobInstances* | Get a list of job instances from the controller |
| *getJobLog* | Read job status and messages from the controller |

### createJobInstance

!!! post "POST /v2/jobs" Arguably the most important call of the whole RPC 2.0 is the "createJobInstance" call where new jobs can be started. Jobs are very powerfull and complex and so this RPC call is very important and very often used.

The request contains a job description that will be created as a new job and then the new job will be described in detail returned in the reply. This way the caller immediately know what properties the controller added to the new job.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_` | integer | Optional | The numerical ID of the cluster that will execute the job. The cluster can also be referenced using the cluster name. |
| `cluster_` | string | Optional | The name of the cluster that will execute the job. The cluster can also be referenced using the cluster ID. |
| `job` | object | Required | The job that will be created. Here the caller can set various properties while other properties will be set by the controller automatically. The most important properties the caller can send are the following: `class_name` (should always be "CmonJobInstance"), `job_spec` (specifies what the job will do), `recurrence` (a crontab string for recurring jobs), `scheduled` (a date/time for deferred execution), `title` (human readable title). |

To get a comprehensive list of the properties the controller supports one can run the following command (or check the jobclasses page):

```
s9s metatype --list-properties --type=CmonJobInstance --long
```

```
{
    "cluster_id": 1,
    "job":
    {
        "class_name": "CmonJobInstance",
        "job_spec":
        {
            "command": "rolling_restart"
        },
        "title": "Rolling Restart",
    },
    "operation": "createJobInstance",
    "request_created": "2017-06-22T12:46:05.054Z",
    "request_id": 3
}
```

Here is an example that registers a recurring job:

```
{
    "cluster_id": 1,
    "job":
    {
        "class_name": "CmonJobInstance",
        "job_spec":
        {
            "command": "success",
            "job_data":
            {
            }
        },
        "recurrence": "2 * * * *",
        "tags": [ "testRecurringJob" ],
        "title": "Simulated Success"
    },
    "operation": "createJobInstance",
    "request_created": "2019-11-06T06:16:23.859Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| job | object | The newly created job instance with all properties set by the controller. |

**Response example**


```
{
    "job":
    {
        "can_be_aborted": false,
        "can_be_deleted": true,
        "class_name": "CmonJobInstance",
        "cluster_id": 1,
        "created": "2017-06-22T12:46:05.000Z",
        "exit_code": 0,
        "group_id": 2,
        "group_name": "users",
        "job_id": 2,
        "job_spec":
        {
            "command": "rolling_restart"
        },
        "status": "DEFINED",
        "status_text": "Waiting",
        "title": "Rolling Restart",
        "user_id": 3,
```

```
        "user_name": "pipas"
    },
    "request_created": "2017-06-22T12:46:05.054Z",
    "request_id": 3,
    "request_processed": "2017-06-22T12:46:05.105Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### updateJobInstance

!!! post "POST /v2/jobs" An RPC 2.0 method call to update an existing (non-closed) job instance.

The request contains the updated job description and status then the updated job will be described in detail returned in the reply. This way the caller immediately know what properties the controller has updated added to the existing job.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| `cluster_` | integer | Optional | The numerical ID of the cluster that will execute the job. The cluster can also be referenced using the cluster name. |
| `cluster_` | string | Optional | The name of the cluster that will execute the job. The cluster can also be referenced using the cluster ID. |
| `job` | object | Require | The job that will be updated. The most important properties the caller can send are the following: `class_name` (should always be "CmonJobInstance"), `job_spec` (specifies what the job will do), `status` (a new job status, e.g. PAUSED or SCHEDULED), `recurrence` (a crontab string for recurring jobs), `scheduled` (a date/time for deferred execution), `title` (human readable title). |

To get a comprehensive list of the properties the controller supports one can run the following command (or check the jobclasses page):

```
s9s metatype --list-properties --type=CmonJobInstance --long
```

```
{
    "cluster_id": 1,
    "job":
    {
        "class_name": "CmonJobInstance",
        "status": "PAUSED",
        "title": "Rolling Restart",
        "job_spec":
        {
```

(continues on next page)

(continued from previous page)

```
            "command": "rolling_restart"
        },
    },
    "operation": "updateJobInstance",
    "request_created": "2020-05-05T12:46:05.054Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| job | object | The updated job instance with all properties as set by the controller. |

**Response example**

```
{
    "job":
    {
        "can_be_aborted": false,
        "can_be_deleted": true,
        "class_name": "CmonJobInstance",
        "cluster_id": 1,
        "created": "2020-05-05T12:46:05.000Z",
        "exit_code": 0,
        "group_id": 2,
        "group_name": "users",
        "job_id": 2,
        "job_spec":
        {
            "command": "rolling_restart"
        },
        "status": "DEFINED",
        "status_text": "Waiting",
        "title": "Rolling Restart",
        "user_id": 3,
        "user_name": "pipas"
    },
    "request_created": "2020-05-05T12:46:05.054Z",
    "request_id": 3,
    "request_processed": "2020-05-05T12:46:05.105Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### deleteJobInstance

!!! post "POST /v2/jobs" Deletes an existing job instance record from the controller. Only closed jobs — those with a status of `FINISHED`, `FAILED`, or `ABORTED` — can be deleted. Use `killJobInstance` to abort a running job before deleting it.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `job_id` | integer | Required | The numerical ID of the job instance to delete. |

```
{
    "job_id": 4,
    "operation": "deleteJobInstance",
    "request_created": "2017-10-20T12:55:08.277Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
    "request_created": "2017-10-20T12:55:08.277Z",
    "request_id": 3,
    "request_processed": "2017-10-20T12:55:08.335Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### killJobInstance

!!! post "POST /v2/jobs" This call was made for killing/aborting jobs running on the controller. To be more precise killing a Cmon job works like killing a UNIX process (it is not a real UNIX process though): we deliver a signal, the job code recognizes the job signal and aborts itself. In short we could say this is not preemptive, jobs that don't want to be aborted will not be aborted.

Here is a request to abort a job:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `job_i` | integer | Required | The numerical ID of the job that should receive the signal. |
| `signal` | integer | Required | The signal itself as a single integer number. In the current implementation there is very little difference between the various signals, it is a good practice to send 15 now. |

```
{
    "job_id": 1,
    "operation": "killJobInstance",
    "request_created": "2019-05-22T06:41:18.944Z",
    "request_id": 3,
    "signal": 15
}
```

### Response

The reply is just a notification about the signal being delivered. The job will be aborted later (if the job supports aborting at all).

**Response example**

```
{
    "request_created": "2019-05-22T06:41:18.944Z",
    "request_id": 3,
    "request_processed": "2019-05-22T06:41:18.953Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```

### cloneJobInstance

!!! post "POST /v2/jobs" This call will clone (copy) an existing job instance in order to execute it again. All the properties are going to be inherited from the existing job and the newly created copy will be executed the same way new jobs are always executed.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `job_id` | integer | Required | The ID of the job that will be cloned. The original job. |
| `cluster_id` | integer | Optional | If this field is provided the job will be executed by this cluster. If this field is not provided the job will be executed by the same cluster that executed the original job. |

```
{
    "cluster_id": 1,
    "job_id": 2,
    "operation": "cloneJobInstance",
    "request_created": "2018-09-19T12:01:57.142Z",
    "request_id": 3
}
```

### getJobInstance

!!! post "POST /v2/jobs" This call is very similar to the "getJobInstances" call, but this one returns information about one job.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `job_id` | integer | Required | The numerical ID of the job that should be returned. |

```
{
    "job_id": 1,
    "operation": "getJobInstance",
    "request_created": "2017-06-23T08:29:47.266Z",
    "request_id": 4
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `job` | object | The job instance itself. |

**Response example**

```
{
    "job":
    {
        "can_be_aborted": false,
        "can_be_deleted": false,
        "class_name": "CmonJobInstance",
        "cluster_id": 0,
        "created": "2017-06-23T08:29:19.000Z",
        "exit_code": 0,
        "group_id": 2,
        "group_name": "users",
        "has_progress": true,
        "job_id": 1,
        "job_spec":
        {
            "command": "create_cluster",
            "job_data":
            {
                "cluster_name": "ft_postgresql_63712",
                "cluster_type": "postgresql_single",
                "enable_uninstall": true,
                "hostname": "192.168.1.192",
                "port": 8089,
                "postgre_password": "passwd12",
                "postgre_user": "postmaster",
                "ssh_user": "pipas",
                "type": "postgresql"
            }
        },
        "progress_percent": 20,
        "started": "2017-06-23T08:29:19.000Z",
        "status": "RUNNING",
        "status_text": "Installing software",
        "title": "Setup PostgreSQL Server",
        "user_id": 3,
        "user_name": "pipas"
    },
    "request_created": "2017-06-23T08:31:23.490Z",
    "request_id": 3,
    "request_processed": "2017-06-23T08:31:23.537Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### getJobInstances

!!! post "POST /v2/jobs" This call can be used to get the job instances from the controller.

If no `cluster_id`, `cluster_ids` or `cluster_name` is sent the controller will send the list of jobs of all the clusters (considering the limit and offset filters of course). The access rights of the authenticated user should also filter the records sent back, but that is unfortunately not yet implemented.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| ascending | boole | Optional | Normally the records will be sent in descending order of time, the newest first. If this boolean is true the order will be descending. |
| cluster_id | integer | Optional | The ID of the cluster from which the jobs will be received. The cluster name can also be used to identify the cluster, but neither of them are mandatory. It is possible to receive a mixed job list with multiple clusters. |
| cluster_id | array | Optional | A list of integers, the IDs of the clusters from which we will get the jobs. No cluster ID or cluster name is mandatory, this call works without any identification of the clusters. |
| cluster_na | string | Optional | The name of the cluster from which the job list will be received. The cluster ID can also be used to identify the cluster, but neither of them are mandatory. |
| show_d | boole | Optional | Turn on state filtering and add jobs in defined state. |
| show_r | boole | Optional | Turn on state filtering and add jobs that are running. |
| show_s | boole | Optional | Turn on state filtering and add jobs that are scheduled. |
| show_a | boole | Optional | Turn on state filtering and add jobs that are aborted. |
| show_f | boole | Optional | Turn on state filtering and add jobs that are finished. |
| show_f | boole | Optional | Turn on state filtering and add jobs that are failed. |
| tags | array | Optional | A list of short strings (tags) to filter the job list. If the list is empty no tags will be considered. |
| limit | integer | Optional | Limits the number of records the same way the SQL LIMIT statement does. |
| offset | integer | Optional | The offset of the first record returned. This works the same way the SQL OFFSET statement works. |

```
{
  "operation": "getJobInstances",
  "request_created": "2017-06-23T05:25:17.891Z",
  "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `jobs` | array | The list of job instances found. The elements in the list are CmonJobInstance class objects. To get a comprehensive list of the properties the controller supports in the CmonJobInstance class please run: `s9s metatype --list-properties --type=CmonJobInstance --long` (or check the jobclasses page). |
| `total` | integer | The total number of records found in the database. The caller can implement a pager using this "total", the "limit" and the "offset" properties. |

**Response example**


```
{
    "jobs": [
    {
        "can_be_aborted": false,
        "can_be_deleted": true,
        "class_name": "CmonJobInstance",
        "cluster_id": 1,
        "created": "2017-06-23T01:00:00.000Z",
        "ended": "2017-06-23T01:00:00.000Z",
        "exit_code": 0,
        "group_id": 1,
        "group_name": "admins",
        "ip_address": "127.0.0.1",
        "job_id": 2,
        "job_spec": "Removing Old Backups",
        "started": "2017-06-23T01:00:00.000Z",
        "status": "FINISHED",
        "status_text": "Command ok",
        "title": "Removing Old Backups",
        "user_id": 0,
        "user_name": "system"
    },
    {
        "can_be_aborted": false,
        "can_be_deleted": true,
        "class_name": "CmonJobInstance",
        "cluster_id": 0,
        "created": "2017-06-22T12:56:33.000Z",
        "ended": "2017-06-22T12:58:18.000Z",
        "exit_code": 0,
        "group_id": 2,
        "group_name": "users",
        "has_progress": true,
        "job_id": 1,
        "job_spec":
        {
```

```
            "command": "create_cluster",
            "job_data":
            {
                "cluster_name": "ft_postgresql_4357",
                "cluster_type": "postgresql_single",
                "enable_uninstall": true,
                "hostname": "192.168.1.191",
                "port": 8089,
                "postgre_password": "passwd12",
                "postgre_user": "postmaster",
                "ssh_user": "pipas",
                "type": "postgresql"
            }
        },
        "progress_percent": 100,
        "started": "2017-06-22T12:56:33.000Z",
        "status": "FINISHED",
        "status_text": "Job finished.",
        "title": "Setup PostgreSQL Server",
        "user_id": 3,
        "user_name": "pipas"
    } ],
    "request_created": "2017-06-23T05:25:17.891Z",
    "request_id": 3,
    "request_processed": "2017-06-23T05:25:17.939Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 2
}
```


### getJobLog

!!! post “POST /v2/jobs” This RPC call is for reading the job status and job messages from the controller. The reply will have the job and the job message in it, so the caller can show the job together with all the messages.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| job_ | integer | Required | The numerical ID of the job to return. |
| ascending | boole | Optional | Normally the job messages returned so that the first is the newest. When this property is set to true in the request the first message will be the oldest, the last will be the newest. |
| limi | integer | Optional | Setting the limit for the log messages sent back. |
| log_ | string | Optional | Controls the severity of the messages that will be sent back. Valid values are "DEBUG", "SUCCESS", "WARNING" or "FAILED" (or as an alternative "JOB_DEBUG", "JOB_SUCCESS", "JOB_WARNING" or "JOB_FAILED"). The "DEBUG" will send back every message, "SUCCESS" will not send debug messages, "WARNING" will send warning and failure messages while "FAILURE" will send only failure messages. |
| offset | integer | Optional | Setting the offset of the first log message to be sent back. |

```
{
    "ascending": true,
    "job_id": 1,
    "limit": 1,
    "log_level": "DEBUG",
    "offset": 100,
    "operation": "getJobLog",
    "request_created": "2019-07-17T05:23:50.262Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| job | object | The info about the job instance as it is described in the jobclasses page. |
| messages | array | The job messages of the given job. This is a list of cmonjobmessage objects. |

**Response example**

```
{
    "job":
    {
```

```
        "can_be_aborted": false,
        "can_be_deleted": true,
        "class_name": "CmonJobInstance",
        "cluster_id": 0,
        "created": "2017-06-23T08:29:19.000Z",
        "ended": "2017-06-23T08:32:31.000Z",
        "exit_code": 0,
        "group_id": 2,
        "group_name": "users",
        "has_progress": true,
        "job_id": 1,
        "job_spec":
        {
            "command": "create_cluster",
            "job_data":
            {
                "cluster_name": "ft_postgresql_63712",
                "cluster_type": "postgresql_single",
                "enable_uninstall": true,
                "hostname": "192.168.1.192",
                "port": 8089,
                "postgre_password": "passwd12",
                "postgre_user": "postmaster",
                "ssh_user": "pipas",
                "type": "postgresql"
            }
        },
        "progress_percent": 100,
        "started": "2017-06-23T08:29:19.000Z",
        "status": "FINISHED",
        "status_text": "Job finished.",
        "title": "Setup PostgreSQL Server",
        "user_id": 3,
        "user_name": "pipas"
    },
    "messages": [
    {
        "class_name": "CmonJobMessage",
        "created": "2017-06-23T08:29:19.000Z",
        "file_name": "CmonCommandHandlerWorker.cpp",
        "job_id": 1,
        "line_number": 2820,
        "message_id": 1,
        "message_status": "JOB_SUCCESS",
        "message_text": "Started create cluster job."
    },

    . . .

    {
```

```
        "class_name": "CmonJobMessage",
        "created": "2017-06-23T08:32:31.000Z",
        "file_name": "CmonCommandHandlerWorker.cpp",
        "job_id": 1,
        "line_number": 945,
        "message_id": 79,
        "message_status": "JOB_WARNING",
        "message_text": "Try Settings -> Cluster Registrations and press
↪'Synchronize Clusters'."
    } ],
    "reply_received": "2017-06-23T08:38:01.052Z",
    "request_created": "2017-06-23T08:38:01.046Z",
    "request_id": 3,
    "request_processed": "2017-06-23T08:38:01.102Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### 4.12.15 Log Related Calls

The `/v2/log` provides calls to access the log handled by the Cmon controller. This is not for the job messages, this is about the logs that are stored in ASCII files.

| Operation | Description |
|---|---|
| *getLogEntries* | Get log messages from the controller. |
| *getLogEntry* | Retrieve one log message by its message ID. |
| *getLogStatistics* | Get statistical information about the log subsystem. |
| *listCollectedLogs* | List logs collected from all hosts stored in the local file system. |
| *getCollectedLog* | Retrieve the details of a selected collected log file. |

#### getLogEntries

!!! post “POST /v2/log” This call can be used to get the log messages from the controller.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| ascending | boolean | Optional | Boolean value controlling the order of the log messages. |
| cluster_name | string | Optional | Identifies the cluster from which the log entries will be returned. The cluster ID can also be used to identify the cluster. |
| cluster_id | integer | Optional | Identifies the cluster from which the log entries will be returned. The cluster name can also be used. |
| component | string | Optional | Identifies the component that sent the log message. Currently this string can be 'Network', 'CmonDatabase', 'Mail', 'Cluster', 'ClusterConfiguration', 'ClusterRecovery', 'Node', 'Host', 'DbHealth', 'DbPerformance', 'SoftwareInstallation', 'Backup' or 'Unknown'. |
| created_b | string | Optional | If this date&time string is provided it will filter the messages by their creation time. |
| created_a | string | Optional | If this date&time string is provided it fill filter the messages by their creation time. |
| limit | integer | Optional | If this provided it limits the number of log entries sent back. |
| offset | integer | Optional | The index of the first item. If this value is provided together with the limit they can be used to implement a pager. |
| severity | string | Optional | This string can be one of the "LOG_EMERG", "LOG_ALERT", "LOG_CRIT", "LOG_ERR", "LOG_WARNING", "LOG_NOTICE", "LOG_INFO", "LOG_DEBUG" to filter the messages by their severity. |


```
{
    "ascending": true,
    "cluster_name": "ft_postgresql_31001",
    "limit": 3,
    "operation": "getLogEntries",
    "request_created": "2017-07-26T07:22:25.767Z",
    "request_id": 200
}
```

### Response

| Field | Type | Description |
|---|---|---|
| log_entries | array | A list that holds the log entries returned as result. The log entries returned as CmonLogMessage objects. |
| log_entry_c | object | A simple statistics showing how many messages are in the reply with various properties. |
| total | integer | The total number of records with the given filters. This with the limit and offset request parameters can be used to implement a pager. |

**Response example**


```
{
    "log_entries": [
    {
        "class_name": "CmonLogMessage",
        "component": "ClusterConfiguration",
        "created": "2017-07-26T08:35:40.704Z",
        "log_class": "LogMessage",
        "log_id": 110,
        "log_origins":
        {
            "sender_binary": "cmon",
            "sender_file": "../../src/cmonhostmanager.cpp",
            "sender_line": 594,
            "sender_pid": 40818,
            "tv_nsec": 704637455,
            "tv_sec": 1501058140
        },
        "log_specifics":
        {
            "cluster_id": 1,
            "message_text": "Registering CmonPostgreSqlHost:␣
↪192.168.1.167:8089"
        },
        "severity": "LOG_DEBUG"
    },

    . . .

    {
        "class_name": "CmonLogMessage",
        "created": "2017-07-26T08:38:11.697Z",
        "log_class": "LogMessage",
        "log_id": 351,
        "log_origins":
        {
            "sender_binary": "cmon",
```

```
            "sender_file": "../../src/cmonalarmdb.cpp",
            "sender_line": 1467,
            "sender_pid": 40818,
            "tv_nsec": 697060177,
            "tv_sec": 1501058291
        },
        "log_specifics":
        {
            "cluster_id": 1,
            "message_text": "hostId: 1, nodeId: 0, alarm: Host is not␣
↪responding"
        },
        "severity": "LOG_DEBUG"
    } ],
    "log_entry_counts":
    {
        "component":
        {
            "Cluster": 1,
            "ClusterConfiguration": 2,
            "Node": 3,
            "Unknown": 194
        },
        "hostname":
        {
            "": 199,
            "192.168.1.167": 1
        },
        "sender_binary":
        {
            "cmon": 200
        },
        "severity":
        {
            "LOG_CRIT": 2,
            "LOG_DEBUG": 192,
            "LOG_ERR": 1,
            "LOG_INFO": 2,
            "LOG_WARNING": 3
        }
    },
    "reply_received": "2017-07-26T08:39:53.936Z",
    "request_created": "2017-07-26T08:39:53.931Z",
    "request_id": 3,
    "request_processed": "2017-07-26T08:39:54.019Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 258
}
```

### getLogEntry

!!! post "POST /v2/log" This call is made for retrieving one message by its message ID.

Here is how a request looks like:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The cluster ID. Use -1 for controller-level scope. |
| `message_id` | integer | Required | The ID of the log message to retrieve. |

```
{
    "cluster_id": -1,
    "message_id": 32,
    "operation": "getLogEntry",
    "request_created": "2019-08-13T05:22:18.882Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `log_entry` | object | The log entry returned as a CmonLogMessage object. |

**Response example**


```
{
    "debug_messages": [ "RPC V2 authenticated user is 'pipas'." ],
    "log_entry":
    {
        "class_name": "CmonLogMessage",
        "created": "2019-08-13T04:00:38.375Z",
        "log_class": "JobStarted",
        "log_id": 32,
        "log_origins":
        {
            "sender_binary": "cmon",
            "sender_file": "CmonCommandHandlerWorker.cpp",
            "sender_line": 332,
            "sender_pid": 49080,
            "tv_nsec": 375158422,
            "tv_sec": 1565668838
        },
        "log_specifics":
        {
            "cluster_id": 0,
            "message_text": "Simulated Failure",
```

```
            "reason": "Simulated Failure"
        },
        "severity": "LOG_INFO"
    },
    "request_created": "2019-08-13T05:22:18.882Z",
    "request_id": 3,
    "request_processed": "2019-08-13T05:22:18.891Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```

### getLogStatistics

!!! post "POST /v2/log" This call can be used to get some statistical information about the subsystem that handles log messages. This is mostly for debugging purposes, but it also can be used to get some information that is shown to the end-user.

### Request

```
{
    "operation": "getLogStatistics",
    "request_created": "2017-07-26T10:58:35.447Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| log_statist | object | Statistical information about the log subsystem, including per-cluster log statistics, entry counts, and writer thread status. |

**Response example**

```
{
    "log_statistics":
    {
        "cluster_log_statistics": [
        {
            "cluster_id": -1,
            "disabled": false,
            "entries_received": 3,
            "format_string": "%C : (%S) %M",
            "last_error_message": "",
            "lines_written": 0,
            "log_file_name": "",
            "max_log_file_size": 5242880,
            "messages_per_sec": 0.05,
```

```
            "syslog_enabled": false,
            "write_cycle_counter": 1
        },
        {
            "cluster_id": 0,
            "disabled": false,
            "entries_received": 93,
            "format_string": "%C : (%S) %M",
            "last_error_message": "Success.",
            "lines_written": 93,
            "log_file_name": "./cmon-ft-install.log",
            "max_log_file_size": 5242880,
            "messages_per_sec": 0.05,
            "syslog_enabled": false,
            "write_cycle_counter": 3
        },
        {
            "cluster_id": 1,
            "disabled": false,
            "entries_received": 218,
            "format_string": "%C : (%S) %M",
            "last_error_message": "Success.",
            "lines_written": 218,
            "log_file_name": "/tmp/cmon_1.log",
            "max_log_file_size": 5242880,
            "messages_per_sec": 0.85,
            "syslog_enabled": false,
            "write_cycle_counter": 11
        } ],
        "current_time": "2017-07-26T11:01:08.884Z",
        "entries_statistics":
        {
            "entries_received": 319,
            "entries_written_to_cmondb": 314
        },
        "has_cmondb": true,
        "last_error_message": "Success.",
        "last_flush_time": "2017-07-26T11:00:59.543Z",
        "log_debug_enabled": false,
        "writer_thread_running": true,
        "writer_thread_started": "2017-07-26T10:55:29.410Z"
    },
    "request_created": "2017-07-26T11:01:08.840Z",
    "request_id": 3,
    "request_processed": "2017-07-26T11:01:08.884Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### listCollectedLogs

!!! post "POST /v2/log" This call retrieves logs from all hosts then store in the local file system. The default folder is /var/lib/cmon/host_logs. Each subfolder represents the cluster which contains some nodes. Nodes/hosts are named using the format: hostname_port.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster whose collected logs to list. |

```
{
    "cluster_id": 10,
    "operation": "listCollectedLogs",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `files` | array | A list of collected log file objects, each containing metadata about a log file stored on the local file system. |

**Response example**

```
{
    "files": [
    {
        "created": "1662022801",
        "filename": "postgresql-2022-09-01_064920.log",
        "hostname": "10.0.8.11",
        "length": "2030",
        "path": "/
↪var/lib/cmon/host_logs/10/10.0.8.11_5432/postgresql-2022-09-01_064920.log"
    },
    {
        "created": "1663655401",
        "filename": "prometheus.log",
        "hostname": "178.128.85.233",
        "length": "313625",
        "path": "/
↪var/lib/cmon/host_logs/10/178.128.85.233_9090/prometheus.log"
    },
    {
        "created": "1663655400",
        "filename": "agent_internal_logs.log",
        "hostname": "10.0.8.11",
        "length": "3568",
```

```
        "path": "/
↪var/lib/cmon/host_logs/10/10.0.8.11_4433/agent_internal_logs.log"
    },
    {
        "created": "1663632001",
        "filename": "cmnd.log",
        "hostname": "10.0.8.11",
        "length": "105914",
        "path": "/var/lib/cmon/host_logs/10/10.0.8.11_4433/cmnd.log"
    }
]
}
```

### getCollectedLog

!!! post "POST /v2/log" This call retrieves the details of a selected log from the UI. Logs are stored in the local file system and 'read' when user clicks on the particular log.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster the log belongs to. |
| `filename` | string | Required | The filename of the log to retrieve. |
| `hostname` | string | Required | The hostname of the node whose log is being retrieved. |

```
{
    "cluster_id": 10,
    "filename": "postgresql-2022-08-31_035553.log",
    "hostname": "10.0.8.11",
    "operation": "getCollectedLog",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `data` | object | An object containing the content and metadata of the retrieved log file. |

**Response example**

```
{
    "data": {
        "content": "2022-06-02 05:16:44.319 UTC [499] LOG:  starting␣
↪PostgreSQL 14.2 (Ubuntu 14.2-1.pgdg20.04+1+b1) on x86_64-pc-linux-gnu,␣
↪compiled by gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0,␣
↪64-bit\r\n2022-06-02 05:16:44.319 UTC [499] LOG:  listening on IPv4␣
```

```
↪address \"0.0.0.0\", port 5432\r\n2022-06-02 05:16:44.319 UTC [499]␣
↪LOG:  listening on IPv6 address \"::\", port 5432\r\n2022-06-02␣
↪05:16:44.339 UTC [499] LOG:  listening on Unix socket \"/
↪var/run/postgresql/.s.PGSQL.5432\"\r\n2022-06-02 05:16:44.451 UTC [508]␣
↪LOG:  database system was shut down at 2022-06-02 05:14:41␣
↪UTC\r\n2022-06-02 05:16:44.653 UTC [499] LOG:  database system is ready␣
↪to accept connections\r\n",
        "created": "1661841000",
        "filename": "postgresql-2022-06-02_051644.log",
        "hostname": "10.0.8.11",
        "length": "642"
    }
}
```

### 4.12.16 PostgreSQL Logical Replication

The `/v2/logical_replication` is for managing logical replication clusters.

| Operation | Description |
|---|---|
| *createPublication* | Create a publication for logical replication. |
| *dropPublication* | Remove an existing publication from logical replication. |
| *modifyPublication* | Modify an existing publication in logical replication. |
| *getPublications* | Retrieve list of all publications configured for logical replication on the specified host. |
| *createSubscription* | Create a subscription for logical replication. |
| *dropSubscription* | Remove an existing subscription from logical replication. |
| *modifySubscription* | Modify an existing subscription in logical replication. |
| *setSubscriptionState* | Set the state (enabled/disabled) of a subscription. |
| *getSubscriptions* | Retrieve list of all subscriptions configured for logical replication on the specified host. |

#### createPublication

!!! post “POST /v2/logical_replication” This call is for creating a publication for logical replication.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Optional | The sub-cluster ID to create the publication on. |
| `subcluster_name` | string | Optional | The sub-cluster name to create the publication on. |
| `pub_name` | string | Required | The name of the new publication. |
| `{pub_db_name\|db_` | string | Required | The database to create the publication on. |
| `include_all_tables` | boolean | Optional | Whether to replicate changes for all tables in the database (including future tables). |
| `db_tables` | array | Optional | List of specific tables to include in the publication. |

**Remarks:** Either `subcluster_id` or `subcluster_name` must be provided. Either `include_all_tables` must be `true` or `db_tables` must contain at least one table. The user must have the necessary privileges to manage replication.

```
{
    "operation": "createPublication",
    "cluster_id": 1,
    "subcluster_id": 3,
    "subcluster_name": "p1",
    "pub_name": "pub1",
    "pub_db_name": "postgres",
    "include_all_tables": true,
    "db_tables": ["t1", "t2"]
}
```

### Response

**Response example**


```
{
    "request_status": "Ok",
    "message": "Created publication 'pub1' in database 'postgres' on␣
↪sub-cluster '3'"
}
```


### dropPublication

!!! post “POST /v2/logical_replication” Removes an existing publication from logical replication.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Optional | The sub-cluster ID containing the publication. |
| `subcluster_name` | string | Optional | The sub-cluster name containing the publication. |
| `pub_name` | string | Required | The name of the publication to drop. |
| `{pub_db_name\|db_name}` | string | Required | The database containing the publication. |

**Remarks:** Either `subcluster_id` or `subcluster_name` must be provided. The user must have the necessary privileges to manage replication.


```
{
    "operation": "dropPublication",
    "cluster_id": 1,
    "subcluster_id": 3,
    "subcluster_name": "p1",
    "pub_name": "pub1",
    "pub_db_name": "postgres"
}
```


### Response

**Response example**


```
{
    "request_status": "Ok",
    "message": "Dropped publication 'pub1' from database 'postgres' on␣
↪sub-cluster 3"
}
```

### modifyPublication

!!! post "POST /v2/logical_replication" Modifies an existing publication in logical replication.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Required | The sub-cluster ID containing the publication. |
| `pub_name` | string | Required | The current name of the publication. |
| `{pub_db_name\|db_name}` | string | Required | The database containing the publication. |
| `new_pub_name` | string | Optional | Optional new name for the publication. |
| `include_all_tables` | boolean | Optional | Whether to replicate all tables (including future ones). |
| `db_tables` | array | Optional | List of specific tables to include in the publication. |

```
{
    "operation": "modifyPublication",
    "cluster_id": 1,
    "subcluster_id": 3,
    "pub_name": "old_pub_name",
    "pub_db_name": "postgres",
    "new_pub_name": "new_pub_name",
    "include_all_tables": true,
    "db_tables": ["t1", "t2"]
}
```

### Response

**Response example**

```
{
    "request_status": "Ok",
    "message": "Modified publication 'old_pub_name' to 'new_pub_name' in␣
↪database 'postgres'"
}
```

### getPublications

!!! post "POST /v2/logical_replication" Retrieves list of all publications configured for logical replication on the specified host.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Optional | The sub-cluster ID to query publications from. |
| `subcluster_name` | string | Optional | The sub-cluster name to query publications from. |

**Remarks:** Either `subcluster_id` or `subcluster_name` must be provided.

```
{
    "operation": "getPublications",
    "cluster_id": 1,
    "subcluster_id": 3,
    "subcluster_name": "p1",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `publications` | array | List of publication objects. |
| `publications[].publication_name` | string | The name of the publication. |
| `publications[].database` | string | The database containing the publication. |
| `publications[].all_tables` | boolean | Whether the publication includes all tables. |
| `publications[].tables` | array | List of specific tables included in the publication. |

**Response example**

```
{
    "request_status": "Ok",
    "publications": [
        {
            "publication_name": "pub1",
            "database": "postgres",
            "all_tables": true,
            "tables": []
        },
        {
            "publication_name": "pub2",
            "database": "mydb",
            "all_tables": false,
            "tables": ["table1", "table2"]
        }
    ]
}
```

### createSubscription

!!! post "POST /v2/logical_replication" Creates a subscription for logical replication. A subscription represents a replication connection to the publisher.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Optional | The sub-cluster ID to create the subscription on. |
| `subcluster_name` | string | Optional | The sub-cluster name to create the subscription on. |
| `sub_name` | string | Required | The name of the new subscription. |
| `pub_name` | string | Required | The name of the publication to subscribe to. |
| `pub_db_name` | string | Optional | The database name of the publication. |
| `pub_origin` | string | Optional | Specifies whether to request changes with no origin ("none") or all changes regardless of origin ("any"). |
| `sub_db_name` | string | Required | The database to create the subscription in (preferred field name). |
| `db_name` | string | Optional | Deprecated: use `sub_db_name` instead. The database to create the subscription in. |
| `copy_data` | boolea | Optional | Specifies whether to copy pre-existing data in the publications when replication starts. |

**Remarks:** Either `subcluster_id` or `subcluster_name` must be provided. The subscription name must be distinct from existing subscriptions in the database. The `pub_db_name` is optional and can be ignored, yet with multiple publications having the same name an error will be reported. In this case, it must be provided. The default value for `pub_origin` is `"any"`, and for `copy_data` is `true`. When `copy_data=true` and `origin=none`, initial sync copies data directly from publisher, which may include data from upstream sources. The user must have the necessary privileges to manage replication.

```
{
    "operation": "createSubscription",
    "cluster_id": 1,
    "subcluster_id": 4,
    "subcluster_name": "p2",
    "sub_name": "sub1",
    "pub_name": "pub1",
    "pub_db_name": "mydb",
    "pub_origin": "any",
    "sub_db_name": "postgres",
    "copy_data": false
}
```

### Response

**Response example**

```
{
    "request_status": "Ok",
    "message": "Created subscription 'sub1' for publication 'pub1' in␣
↪database 'postgres' on sub-cluster 4"
}
```

### dropSubscription

!!! post “POST /v2/logical_replication” Removes an existing subscription from logical replication.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Optional | The sub-cluster ID containing the subscription. |
| `subcluster_name` | string | Optional | The sub-cluster name containing the subscription. |
| `sub_name` | string | Required | The name of the subscription to drop. |
| `{sub_db_name\|db_name}` | string | Required | The database containing the subscription. |

**Remarks:** Either `subcluster_id` or `subcluster_name` must be provided. The user must have the necessary privileges to manage replication.

```
{
    "operation": "dropSubscription",
    "cluster_id": 1,
    "subcluster_id": 4,
    "subcluster_name": "p2",
    "sub_name": "sub1",
    "sub_db_name": "postgres"
}
```

### Response

**Response example**

```
{
    "request_status": "Ok",
    "message": "Dropped subscription 'sub1' from database 'postgres' on␣
↪sub-cluster 4"
}
```

### modifySubscription

!!! post "POST /v2/logical_replication" Modifies an existing subscription in logical replication.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Required | The sub-cluster ID containing the subscription. |
| `sub_name` | string | Required | The current name of the subscription. |
| `sub_db_name` | string | Required | The database containing the subscription (preferred field name). |
| `db_name` | string | Optional | Deprecated: use `sub_db_name` instead. The database containing the subscription. |
| `new_sub_name` | string | Optional | Optional new name for the subscription. |
| `pub_origin` | string | Optional | Optional new publication origin setting ('any' or 'none') for the subscription. |
| `enabled` | boolean | Optional | Optional flag to enable/disable the subscription. |

```
{
    "operation": "modifySubscription",
    "cluster_id": 1,
    "subcluster_id": 4,
    "sub_name": "old_sub_name",
    "sub_db_name": "postgres",
    "new_sub_name": "new_sub_name",
    "pub_origin": "none",
    "enabled": true
}
```

### Response

**Response example**


```
{
    "request_status": "Ok",
    "message": "Modified subscription 'old_sub_name' to 'new_sub_name' in␣
↪database 'postgres'"
}
```


### setSubscriptionState

!!! post "POST /v2/logical_replication" Sets the state (enabled/disabled) of a subscription.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Required | The sub-cluster ID containing the subscription. |
| `sub_name` | string | Required | The name of the subscription to modify. |
| `{sub_db_name\|db_name` | string | Required | The database containing the subscription. |
| `enabled` | boolean | Required | Whether to enable (true) or disable (false) the subscription. |

**Remarks:** The user must have the necessary privileges to manage replication.


```
{
    "operation": "setSubscriptionState",
    "cluster_id": 1,
    "subcluster_id": 4,
    "sub_name": "sub1",
    "sub_db_name": "postgres",
    "enabled": true
}
```


### Response

**Response example**


```
{
    "request_status": "Ok",
    "message": "Subscription 'sub1' in database 'postgres' on sub-cluster␣
↪4 has been enabled"
}
```

### getSubscriptions

!!! post "POST /v2/logical_replication" Retrieves list of all subscriptions configured for logical replication on the specified host.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The logical replication cluster ID. |
| `subcluster_id` | integer | Optional | The sub-cluster ID to query subscriptions from. |
| `subcluster_name` | string | Optional | The sub-cluster name to query subscriptions from. |

**Remarks:** Either `subcluster_id` or `subcluster_name` must be provided.

```
{
    "operation": "getSubscriptions",
    "cluster_id": 1,
    "subcluster_id": 4,
    "subcluster_name": "p2"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `subscriptions` | array | List of subscription objects. |
| `subscriptions[].subscription_name` | string | The name of the subscription. |
| `subscriptions[].database` | string | The database containing the subscription. |
| `subscriptions[].publication_name` | string | The name of the publication being subscribed to. |
| `subscriptions[].publication_database` | string | The database of the publication. |
| `subscriptions[].publication_origin` | string | The publication origin setting ('any' or 'none'). |
| `subscriptions[].enabled` | boolean | Whether the subscription is enabled. |

**Response example**

```
{
    "request_status": "Ok",
    "subscriptions": [
        {
            "subscription_name": "sub1",
            "database": "postgres",
            "publication_name": "pub1",
            "publication_database": "mydb",
            "publication_origin": "any",
            "enabled": true
```

```
        },
        {
            "subscription_name": "sub2",
            "database": "mydb",
            "publication_name": "pub2",
            "publication_database": "postgres",
            "publication_origin": "none",
            "enabled": false
        }
    ]
}
```

### 4.12.17 Maintenance

The "v2/maintenance" is for creating, deleting and reading maintenance periods.

| Operation | Description |
|---|---|
| *addMaintenance* | Add a new maintenance period for a cluster or host. |
| *removeMaintenance* | Delete a maintenance period before it expires. |
| *getMaintenance* | Retrieve maintenance periods for clusters and/or hosts. |
| *getCurrentMaintenance* | Get the currently active maintenance period for a cluster or host. |
| *getNextMaintenance* | Get the next upcoming maintenance period for a cluster or host. |

#### addMaintenance

!!! post "POST /v2/maintenance" The addMaintenance call is for adding new maintenance periods for clusters and hosts. A maintenance period is either belong to an entire cluster, in which case the cluster must be identified or it belongs to a host, so that the host name must be provided.

#### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `deadline` | string | Required | Marks the end of the maintenance period. At this time the maintenance period will automatically expire. |
| `cluster_id` | integer | Optional | The numerical ID of the cluster if this is a cluster maintenance period. |
| `hostname` | string | Optional | The name of the host if this is a host maintenance. |
| `initiate` | string | Required | The start of the maintenance period. |
| `reason` | string | Required | A human readable string which the users will see. |

```
{
    "deadline": "2017-06-23T10:46:37.000Z",
```

(continues on next page)

(continued from previous page)


```
    "hostname": "192.168.1.195",
    "initiate": "2017-06-23T10:46:27.000Z",
    "operation": "addMaintenance",
    "reason": "test_31643_maintenance",
    "request_created": "2017-06-23T10:46:27.754Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| UUID | string | A unique string which can be used to identify the maintenance period. |

**Response example**


```
{
    "UUID": "366372d3-2633-f126-bf8b-c69418274d69",
    "request_created": "2017-06-23T10:46:27.754Z",
    "request_id": 3,
    "request_processed": "2017-06-23T10:46:27.800Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### removeMaintenance

!!! post "POST /v2/maintenance" This RPC call can be used to delete maintenance periods before their time. Deleting a maintenance period will make them expire immediately.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| UUID | string | Required | Indentifies which maintenance period is going to be deleted. |


```
{
    "UUID": "a1f2f274-b151-e286-0eee-199efaa5b444",
    "operation": "removeMaintenance",
    "request_created": "2017-06-23T13:23:28.770Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
    "request_created": "2017-06-23T13:23:28.770Z",
    "request_id": 3,
    "request_processed": "2017-06-23T13:23:28.816Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

## getMaintenance

!!! post "POST /v2/maintenance" This call either sends back cluster maintenance periods when it received a cluster ID or host maintenance periods when it received one or more hostnames. When no cluster ID provided and no host names are sent either, both cluster maintenance and host maintenance periods are sent back.

The idea behind this call is that when the UI shows a host, it gets the maintenance periods of that host to show the details, when it shows a cluster it gets the maintenance periods of that cluster to show the details. If however the UI needs to know what nodes are in a cluster, that's an other call, that's not a "getMaintenance" call at all.

### Request

```
{
    "operation": "getMaintenance",
    "request_created": "2017-06-23T10:46:27.913Z",
    "request_id": 3
}
```

### Response

Here is an example of an s9s run. It shows two maintenance records, one for a host and one for a cluster:

```
$ s9s maintenance --list --long
ST UUID    OWNER GROUP      START               END                 ␣
↪HOST/CLUSTER  *          REASON
-h 1da4955 pipas testgroup 2019-11-10 13:05:40 2019-11-10 15:05:40 192.
↪168.0.58          testMaintenanceJob4
Ac 2358876 pipas testgroup 13:05:37            14:05:37            ␣
↪ft_maintenance_36279 testMaintenanceJob1
-c d89319c pipas testgroup 2019-11-09 13:05:39 2019-11-09 15:05:39␣
↪ft_maintenance_36279 testMaintenanceJob3
Total: 2
```

But the two records has three maintenance periods in it, because the same cluster has two maintenance periods. Now these looks like this in a reply:

**Response example**


```
{
  "debug_messages": [ "RPC V2 authenticated user is 'pipas'."  ],
  "maintenance_records": [
    {
      "class_name": "CmonMaintenanceInfo",
      "hostname": "192.168.0.58",
      "is_active": false,
      "maintenance_periods": [ {
          "UUID": "1da49552-dd68-4604-ab16-a340c312cd9c",
          "deadline": "2019-11-10T14:05:40.000Z",
          "groupid": 4,
          "groupname": "testgroup",
          "initiate": "2019-11-10T12:05:40.000Z",
          "is_active": false,
          "reason": "testMaintenanceJob4",
          "userid": 4,
          "username": "pipas"
        }  ]
    },
    {
      "class_name": "CmonMaintenanceInfo",
      "cluster": {
        "acl": "user::rwx,group::rwx,other::---",
        "alarm_statistics": {
          "class_name": "CmonAlarmStatistics"
        },
        "cdt_path": "/",
        "class_name": "CmonClusterInfo",
        "cluster_auto_recovery": true,
        "cluster_id": 1,
        "cluster_name": "ft_maintenance_36279",
        "cluster_type": "POSTGRESQL_SINGLE",
        "configuration_file": "/tmp/cmon_1.cnf",
        "effective_privileges": "",
        "group_owner": {
          "acl": "user::rwx,group::rwx,other::---",
          "cdt_path": "/groups",
          "class_name": "CmonGroup",
          "created": "2019-11-08T12:02:37.983Z",
          "group_id": 4,
          "group_name": "testgroup",
          "owner_group_id": 1,
          "owner_group_name": "admins",
          "owner_user_id": 1,
          "owner_user_name": "system"
        },
        "job_statistics": {
          "class_name": "CmonJobStatistics"
        },
```

```
        "log_file": "/tmp/cmon_1.log",
        "maintenance_mode_active": true,
        "managed": true,
        "node_auto_recovery": true,
        "owner": {
          "acl": "user::rwx,group::r--,other::r--",
          "cdt_path": "/",
          "class_name": "CmonUser",
          "created": "2019-11-08T12:02:37.995Z",
          "disabled": false,
          "email_address": "laszlo@severalnines.com",
          "first_name": "Laszlo",
          "groups": [ {
              "acl": "user::rwx,group::rwx,other::---",
              "cdt_path": "/groups",
              "class_name": "CmonGroup",
              "created": "2019-11-08T12:02:37.983Z",
              "group_id": 4,
              "group_name": "testgroup",
              "owner_group_id": 1,
              "owner_group_name": "admins",
              "owner_user_id": 1,
              "owner_user_name": "system"
            } ],
          "last_failed_login": "",
          "last_login": "2019-11-08T12:24:56.456Z",
          "last_name": "Pere",
          "n_failed_logins": 0,
          "origin": "CmonDb",
          "owner_group_id": 1,
          "owner_group_name": "admins",
          "owner_user_id": 4,
          "owner_user_name": "pipas",
          "suspended": false,
          "timezone": {
            "abbreviation": "CET",
            "class_name": "CmonTimeZone",
            "name": "Central European Time",
            "offset": -3600,
            "has_dst_rules": true
          },
          "user_id": 4,
          "user_name": "pipas"
        },
        "state": "STARTED",
        "status_text": "All nodes are operational.",
        "vendor": "postgres",
        "version": "9.6"
      },
      "cluster_id": 1,
```

```
      "is_active": true,
      "maintenance_periods": [
        {
          "UUID": "2358876c-8eec-42c8-b149-69e722044b3f",
          "deadline": "2019-11-08T13:05:37.178Z",
          "groupid": 4,
          "groupname": "testgroup",
          "initiate": "2019-11-08T12:05:37.178Z",
          "is_active": true,
          "reason": "testMaintenanceJob1",
          "userid": 4,
          "username": "pipas"
        },
        {
          "UUID": "d89319c9-3e7c-462e-9f54-4052682d2daa",
          "deadline": "2019-11-09T14:05:39.000Z",
          "groupid": 4,
          "groupname": "testgroup",
          "initiate": "2019-11-09T12:05:39.000Z",
          "is_active": false,
          "reason": "testMaintenanceJob3",
          "userid": 4,
          "username": "pipas"
        }
      ]
    }
  ],
  "reply_received": "2019-11-08T12:24:56.475Z",
  "request_created": "2019-11-08T12:24:56.464Z",
  "request_id": 3,
  "request_processed": "2019-11-08T12:24:56.475Z",
  "request_status": "Ok",
  "request_user_id": 4,
  "total": 2
}
```


### getCurrentMaintenance

!!! post “POST /v2/maintenance” This RPC call was made so that the client program has a quick and easy way to get information about the active maintanence for a cluster or a host.

If this call finds a host maintenance that is active it will return that, if it finds a cluster maintenance that will be returned.

### Request

Here is a call to search for a cluster wide maintenance (no host maintenance is returned ever).


```
{
    "cluster_id": 1,
```


(continues on next page)

(continued from previous page)

```
    "operation": "getCurrentMaintenance",
    "request_created": "2019-11-11T15:07:37.539Z",
    "request_id": 3
}
```

Here is a call that returns either the cluster maintenance or the host maintenance or no maintenance if neither of those found:

```
{
    "cluster_id": 1,
    "hostname": "10.10.1.8",
    "operation": "getCurrentMaintenance",
    "request_created": "2019-11-12T08:55:58.507Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
  "debug_messages": [ "RPC V2 authenticated user is 'pipas'."  ],
  "found_maintenance": true,
  "maintenance_period": {
    "UUID": "9dec403d-0940-4603-b93f-3c58fa773ce1",
    "class_name": "CmonMaintenanceDate",
    "deadline": "2019-11-11T16:07:12.969Z",
    "groupid": 4,
    "groupname": "testgroup",
    "initiate": "2019-11-11T15:07:12.969Z",
    "reason": "testMaintenanceJob1",
    "userid": 4,
    "username": "pipas"
  },
  "reply_received": "2019-11-11T15:07:37.546Z",
  "request_created": "2019-11-11T15:07:37.539Z",
  "request_id": 3,
  "request_processed": "2019-11-11T15:07:37.546Z",
  "request_status": "Ok",
  "request_user_id": 4,
  "ui_string": "Cluster 1 is under maintenance: testMaintenanceJob1"
}
```

### getNextMaintenance

!!! post “POST /v2/maintenance” This RPC call was made so that the client program has a quick and easy way to get information about the next coming maintenance periods.

If this call finds a host maintenance that is active it will return that, if it finds a cluster maintenance that

will be returned.

### Request

Here is a call to search for a cluster wide maintenance (no host maintenance is returned ever).

```
{
    "cluster_id": 1,
    "operation": "getNextMaintenance",
    "request_created": "2019-11-12T08:07:29.966Z",
    "request_id": 3
}
```

Here is a call that returns either the cluster maintenance or the host maintenance or no maintenance if neither of those found:

```
{
    "cluster_id": 1,
    "hostname": "10.10.1.8",
    "operation": "getNextMaintenance",
    "request_created": "2019-11-12T08:55:15.628Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
  "debug_messages": [ "RPC V2 authenticated user is 'pipas'."  ],
  "found_maintenance": true,
  "maintenance_period": {
    "UUID": "8a812d58-18a2-4d55-9702-2e5e05de1a41",
    "class_name": "CmonMaintenanceDate",
    "deadline": "2019-11-13T09:07:15.000Z",
    "groupid": 4,
    "groupname": "testgroup",
    "initiate": "2019-11-13T08:07:15.000Z",
    "reason": "testMaintenanceJob1",
    "userid": 4,
    "username": "pipas"
  },
  "reply_received": "2019-11-12T08:07:29.973Z",
  "request_created": "2019-11-12T08:07:29.966Z",
  "request_id": 3,
  "request_processed": "2019-11-12T08:07:29.973Z",
  "request_status": "Ok",
  "request_user_id": 4,
  "ui_string": "Cluster 1 maintenance starts Nov 13 08:07:15:␣
↪testMaintenanceJob1"
}
```

### 4.12.18 Mysql databaseperformance

| Operation | Description |
|---|---|
| *noPrimaryKeys* | Returns a list of tables that do not have primary keys. |
| *myIsamTables* | Returns a list of tables that are using MyISAM engine. |
| *redundantIndexes* | Returns a list of tables that have redundant indexes. |

#### noPrimaryKeys

!!! post "POST /v2/performance" An RPC call that returns a list of tables that do not have primary keys.

#### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |

```
{
    "cluster_id": 1,
    "operation": "noPrimaryKeys",
}
```

#### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of tables without primary keys. |
| `data` | array | List of tables without primary keys. |
| `data[].database` | string | Database name. |
| `data[].engine` | string | Storage engine used by the table. |
| `data[].table` | string | Table name. |

**Response example**

```
{
    "controller_id": "5bfb8a65-8bbf-4762-9c48-e6603377b26e",
    "request_processed": "2024-08-01T12:16:59.926Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "total": 2,
    "data":
    [
        {
            "database": "mysql",
            "engine": "myEngine",
            "table": "AAA"
        },
```

```
            {
                "database": "mysql",
                "engine": "myEngine",
                "table": "BBB"
            }
        ],
        "debug_messages":
        [
            "RPC V2 authenticated user is 'Adam'."
        ]
}
```

### myIsamTables

!!! post "POST /v2/performance" An RPC call that returns a list of tables that are using MyISAM engine.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |

```
{
    "cluster_id": 1,
    "operation": "myisamTables",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of MyISAM tables. |
| `data` | array | List of MyISAM tables. |
| `data[].database` | string | Database name. |
| `data[].engine` | string | Storage engine used by the table. |
| `data[].table` | string | Table name. |

**Response example**

```
{
    "controller_id": "5bfb8a65-8bbf-4762-9c48-e6603377b26e",
    "request_processed": "2024-08-01T12:16:59.926Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "total": 1,
    "data":
    [
```

```
        {
            "database": "mysql",
            "engine": "myIsam",
            "table": "BBB"
        }
    ],
    "debug_messages":
    [
        "RPC V2 authenticated user is 'Adam'."
    ]
}
```

### redundantIndexes

!!! post "POST /v2/performance" An RPC call that returns a list of tables that have redundant indexes.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |

```
{
    "cluster_id": 1,
    "operation": "redundantIndexes",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of tables with redundant indexes. |
| `data` | array | List of tables with redundant indexes. |
| `data[].advise` | string | Advisory note about the redundant index. |
| `data[].database` | string | Database name. |
| `data[].index` | string | Index identifier. |
| `data[].indexColumns` | string | Columns in the index. |
| `data[].redidxColumns` | string | Columns in the redundant index. |
| `data[].redundantIndex` | string | Name of the redundant index. |
| `data[].table` | string | Table name. |

**Response example**

```
{
    "controller_id": "5bfb8a65-8bbf-4762-9c48-e6603377b26e",
    "request_processed": "2024-08-01T12:16:59.926Z",
    "request_status": "Ok",
```

```
    "request_user_id": 4,
    "total": 1,
    "data":
    [
        {
            "advise": "Do not touch",
            "database": "myXdb",
            "index": "66",
            "indexColumns": "88",
            "redidxColumns": "55",
            "redundantIndex": "111",
            "table": "myTable"
        }
    ],
    "debug_messages":
    [
        "RPC V2 authenticated user is 'Adam'."
    ]
}
```


### 4.12.19 Reports

The `/v2/reports` provides access to reports generated by the Cmon controller.

| Operation | Description |
|---|---|
| *Download report* | Download a generated report file by basename and cluster ID. |
| *generateReport* | Generate a named report for a cluster and return its content inline. |
| *getReport* | Retrieve a specific report by ID. |
| *getReportTemplates* | Return report templates describing available report types. |
| *deleteReport* | Delete an existing report from the controller. |
| *getReports* | Get existing reports from the controller. |
| *listErrorReports* | List generated error reports. |
| *removeErrorReport* | Remove a specific error report. |
| *downloadErrorReport* | Download a specific error report as a tar.gz file. |
| *addSchedule* | Schedule an operational report. |
| *removeSchedule* | Remove an operational report schedule. |
| *listSchedules* | List currently active operational report schedules. |

#### Download report

!!! post “POST /v2/reports” Pass the basename of the report after path `/v2/reports/`, and you must specify the cluster-id as well via the ?cluster_id= query argument.

#### Request

```
curl -k -b cookies.jar
↪'https://127.0.0.1:9501/v2/reports/default_2023-01-31_071953.html?cluster_id=103
↪'
```

### Response

```
<!DOCTYPE html>
<html><body>
...
<p class="text"><br/><b>How do you like this report?</b><br/>What else do␣
↪you like to see or not see? We'd <a href=
↪"mailto:info@severalnines.com?subject=Operation Reports Feedback">
↪appreciate your feedback</a>. Thanks!<br/></p>

</body>
</html>
```

### generateReport

!!! post "POST /v2/reports" Generates a named report for a cluster and returns its content inline. The `report_type` field specifies which report to generate, and `text_format` controls the output format (`AnsiTerminal`, `Html`, etc.). The generated report content is returned in `report.content` and `report.document`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster to generate the report for. |
| `report` | object | Required | A CmonReport class object specifying the report configuration. |
| `report.class_name` | string | Required | Must be `CmonReport`. |
| `report.recipients` | string | Optional | Email recipients for the report. |
| `report.report_type` | string | Required | The type of report to generate. |
| `report.text_format` | string | Required | The output format (e.g., `AnsiTerminal`, `Html`). |

```
{
    "cluster_id": 1,
    "operation": "generateReport",
    "report":
    {
        "class_name": "CmonReport",
        "recipients": "laszlo@severalnines.com",
        "report_type": "testreport",
        "text_format": "AnsiTerminal"
    },
    "request_created": "2017-08-04T11:06:32.148Z",
```

(continues on next page)

(continued from previous page)

```
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| report | object | The report that has been created as a CmonReport class object. |
| report.class_name | string | The class name, CmonReport. |
| report.content | string | The rendered text content of the report. |
| report.created | string | Timestamp when the report was created. |
| report.document | object | The structured document representation of the report. |
| report.owner | string | The owner of the report. |
| report.path | string | The file path where the report was saved. |
| report.recipients | string | Email recipients for the report. |
| report.report_id | integer | The unique ID of the generated report. |
| report.report_type | string | The type of the report. |
| report.text_format | string | The output format used. |
| report.title | string | The title of the report. |

To get a comprehensive list of the properties the controller supports in the CmonReport class one can run the following command:

```
s9s metatype --list-properties --type=CmonReport --long
```

**Response example**


```
{
    "report":
    {
        "class_name": "CmonReport",
        "content": "  Simple Test Report\n\n    This report was created␣
↪to test the report subsystem. The document was\n  created in the␣
↪CmonReporter::buildTestReport() method.\n\n",
        "created": "2017-08-04T11:06:32.000Z",
        "document":
        {
            "className": "CmonDocument",
            "elements": [
            {
                "className": "CmonTitle",
                "text": "Simple Test Report",
                "titleLevel": 1
            },
            {
                "className": "CmonParagraph",
                "text": "This report was created to test the report␣
↪subsystem. The document was created in the␣
```

```
↪CmonReporter::buildTestReport() method."
            } ]
        },
        "owner": "pipas",
        "path": "/
↪home/pipas/s9s_tmp/1/postgresql_single/cmon-reports/testreport_2017-08-04_130632.html"
↪
        "recipients": "laszlo@severalnines.com",
        "report_id": 1,
        "report_type": "testreport",
        "text_format": "AnsiTerminal",
        "title": "Simple Test Report"
    },
    "request_created": "2017-08-04T11:06:32.148Z",
    "request_id": 3,
    "request_processed": "2017-08-04T11:06:32.200Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### getReport

!!! post "POST /v2/reports" The request format for this call has recently been changed. Instead of passing just a single ID we need to pass a CmonReport class object.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `report` | object | Required | A CmonReport class object identifying the report to retrieve. |
| `report.class_name` | string | Required | Must be `CmonReport`. |
| `report.report_id` | integer | Required | The ID of the report to retrieve. |
| `report.text_format` | string | Optional | The output format (e.g., `AnsiTerminal`). |

```
{
    "operation": "getReport",
    "report":
    {
        "class_name": "CmonReport",
        "report_id": 1,
        "text_format": "AnsiTerminal"
    },
    "request_created": "2019-10-04T07:39:40.138Z",
```

(continues on next page)

(continued from previous page)

```
    "request_id": 3
}
```

### getReportTemplates

!!! post "POST /v2/reports" This call is to return the report templates describing what kind of reports are available for the user when generating reports. Report templates are just simple CmonReport objects with a few properties set to reflect the properties that the report will have when a report with the same report type is used.

So template name is just an other way of saying `report_type`.

### Request

```
{
    "operation": "getReportTemplates",
    "request_created": "2019-07-31T06:45:30.083Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `reports` | array | List of report template objects. |
| `reports[].class_name` | string | The class name, `CmonReport`. |
| `reports[].report_type` | string | The report type identifier (template name). |
| `reports[].title` | string | The human-readable title of the report template. |

**Response example**

```
{
    "reports": [
    {
        "class_name": "CmonReport",
        "report_type": "default",
        "title": "System Report"
    },

    . . .

    {
        "class_name": "CmonReport",
        "report_type": "capacity",
        "title": "Database Growth Report"
    } ],
    "request_created": "2019-07-31T06:45:30.083Z",
    "request_id": 3,
```

```
    "request_processed": "2019-07-31T06:45:30.089Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```

### deleteReport

!!! post "POST /v2/reports" A call to delete an existing report from the controller.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster the report belongs to. |
| `report` | object | Required | A CmonReport class object identifying the report to delete. |
| `report.class_name` | string | Required | Must be `CmonReport`. |
| `report.report_id` | integer | Required | The ID of the report to delete. |

```
{
    "cluster_id": 1,
    "operation": "deleteReport",
    "report":
    {
        "class_name": "CmonReport",
        "report_id": 1
    },
    "request_created": "2017-08-04T07:11:38.745Z",
    "request_id": 4
}
```

### Response

**Response example**

```
{
    "request_created": "2017-08-04T07:11:38.745Z",
    "request_id": 4,
    "request_processed": "2017-08-04T07:11:38.793Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### getReports

!!! post "POST /v2/reports" A call to get existing reports from the controller.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster to retrieve reports for. |

```
{
    "cluster_id": 1,
    "operation": "getReports",
    "request_created": "2017-08-02T12:34:59.892Z",
    "request_id": 4
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `reports` | array | List of report info objects. |
| `reports[].class_name` | string | The class name, `CmonReportInfo`. |
| `reports[].cluster_id` | integer | The cluster ID the report belongs to. |
| `reports[].created` | string | Timestamp when the report was created. |
| `reports[].name` | string | The filename of the report. |
| `reports[].owner` | string | The owner of the report. |
| `reports[].path` | string | The full file path of the report. |
| `reports[].recipients` | string | Email recipients for the report. |
| `reports[].report_id` | integer | The unique ID of the report. |
| `reports[].report_type` | string | The type of the report. |
| `total` | integer | Total number of reports returned. |

**Response example**

```
{
    "reports": [
    {
        "class_name": "CmonReportInfo",
        "cluster_id": 1,
        "created": "2017-08-02T13:01:44.000Z",
        "name": "availability_2017-08-02_143459.html",
        "owner": "pipas",
        "path": "/
↪home/pipas/s9s_tmp/1/postgresql_single/cmon-reports/availability_2017-08-02_143459.htm
↪
        "recipients": "",
        "report_id": 1,
        "report_type": "availability"
```

```
    } ],
    "request_created": "2017-08-02T12:34:59.892Z",
    "request_id": 4,
    "request_processed": "2017-08-02T12:34:59.936Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 1
}
```

### listErrorReports

!!! post "POST /v2/reports" A call to list the generated error reports.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster to list error reports for. |

```
{
    "cluster_id": 90,
    "operation": "listErrorReports",
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `is_superuser` | boolean | Whether the authenticated user is a superuser. |
| `total` | integer | Total number of error reports. |
| `data` | array | List of error report objects. |
| `data[].created` | string | Timestamp when the error report was created. |
| `data[].id` | integer | The unique ID of the error report. |
| `data[].path` | string | The file path of the error report archive. |
| `data[].size` | string | The size of the error report file. |
| `data[].www` | boolean | Whether the report is accessible via the web. |

**Response example**

```
{
 "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
 "is_superuser": true,
 "request_processed": "2022-09-26T08:21:43.405Z",
 "request_status": "Ok",
 "total": 1,
 "data":
 [
```

```
        {
            "created": "2022-09-26T08:21:18.000Z",
            "id": 13,
            "path": "/
↪home/cmon_user/s9s_tmp/error-report-2022-09-26_082116-cluster90.tar.gz",
            "size": "7.63 MiB",
            "www": false
        }
    ]
}
```

### removeErrorReport

!!! post "POST /v2/reports" A call to list the generated error reports.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster the error report belongs to. |
| `id` | integer | Required | The ID of the error report to remove. |

```
{
    "cluster_id": 90,
    "operation": "removeErrorReport",
    "id": 13
}
```

### Response

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2022-09-26T08:58:28.004Z",
  "request_status": "Ok"
}
```

### downloadErrorReport

!!! post "POST /v2/reports" A method to download a specific error report. Please NOTE this method will return the generated `.tar.gz` file directly (so don't expect a JSON reply).

The return value is an `application/octet-stream` of the `tar.gz` file directly. Or in case of error a normal JSON RPC reply including the error reason.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| cluster_id | integer | Required | The ID of the cluster the error report belongs to. |
| id | integer | Required | The ID of the error report to download. |

```
{
    "cluster_id": 90,
    "operation": "downloadErrorReport",
    "id": 13
}
```

### addSchedule

!!! post “POST /v2/reports” A call to schedule an operational report.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| cluster_id | integer | Required | The ID of the cluster to schedule the report for. |
| name | string | Required | The name/type of the report to schedule. |
| schedule | string | Required | The cron expression defining the schedule. |
| recipients | string | Optional | Email recipients for the scheduled report. |

```
{
    "cluster_id": 90,
    "operation": "addSchedule",
    "name": "default",
    "schedule": "*0 10 * * *",
    "recipients": "kedazo@severalnines.com"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| is_superuser | boolean | Whether the authenticated user is a superuser. |
| id | integer | The ID of the newly created schedule. |

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2022-09-26T08:58:28.004Z",
  "request_status": "Ok",
```

```
  "id": 103
}
```

### removeSchedule

!!! post "POST /v2/reports" A call to remove an operational report schedule.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| id | integer | Required | The ID of the schedule to remove. |

```
{
    "id": 112,
    "operation": "removeSchedule"
}
```

### Response

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2022-09-26T08:58:28.004Z",
  "request_status": "Ok"
}
```

### listSchedules

!!! post "POST /v2/reports" Lists the currently active operational reports schedules.

### Request

```
{
    "operation": "listSchedules"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| is_superuser | boolean | Whether the authenticated user is a superuser. |
| total | integer | Total number of schedules. |
| data | array | List of schedule objects. |
| data[].arguments | string | Arguments passed to the scheduled report. |
| data[].command | string | The report command/type to run. |
| data[].id | integer | The unique ID of the schedule. |
| data[].recipients | string | Email recipients for the scheduled report. |
| data[].schedule | string | The cron expression for the schedule. |
| user | object | The authenticated user object. |

**Response example**


```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2023-01-26T14:16:16.603Z",
  "request_status": "Ok",
  "total": 2,
  "data":
  [
      {
          "arguments": "days=7",
          "command": "default",
          "id": 1,
          "recipients": "",
          "schedule": "1 6 * * *"
      },
      {
          "arguments": "days=7",
          "command": "upgrade",
          "id": 2,
          "recipients": "kedazo@severalnines.com",
          "schedule": "1 22 10 * *"
      }
  ],
  "debug_messages":
  [
      "RPC V2 authenticated user is 'alex'."
  ],
  "user":
  {
      "class_name": "CmonUser",
      "cdt_path": "/",
      "owner_user_id": 68,
      "owner_user_name": "alex",
      "owner_group_id": 1,
```

```
        "owner_group_name": "admins",
        "acl": "user::rwx,group::r--,other::r--",
        "created": "2022-12-16T15:21:36.542Z",
        "disabled": false,
        "distinguished_name": "uid=alex,ou=People,dc=severalnines,dc=org",
        "email_address": "alex@severalnines.com",
        "first_name": "alex",
        "last_failed_login": "",
        "last_login": "2023-01-26T14:16:16.601Z",
        "n_failed_logins": 0,
        "origin": "LDAP",
        "suspended": false,
        "user_id": 68,
        "user_name": "alex",
        "groups":
        [
            {
                "class_name": "CmonGroup",
                "cdt_path": "/groups",
                "owner_user_id": 1,
                "owner_user_name": "system",
                "owner_group_id": 1,
                "owner_group_name": "admins",
                "acl": "user::rwx,group::rwx,other::---",
                "created": "2022-02-23T11:01:14.628Z",
                "group_id": 1,
                "group_name": "admins"
            }
        ],
        "timezone":
        {
            "class_name": "CmonTimeZone",
            "name": "Central European Time",
            "abbreviation": "CET",
            "offset": -3600,
            "has_dst_rules": false
        }
    }
}
```


### 4.12.20 Repositories

The `/v2/repositories` path provides calls to create and manage local software repositories and get information about supported software.

| Operation | Description |
|---|---|
| *getRepositories* | Returns available local repositories. |
| *deleteRepository* | Removes a software repository registration and deletes the repository directory from disk. |
| *getRepository-Setup* | Prints out the generated repository filename and contents for manual use. |
| *getSupportedSe-tups* | Returns what possible setups can be used to create/install a new node. |

### getRepositories

!!! post "POST /v2/repositories" Returns available local repositories.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The cluster ID. |
| `cluster_type` | string | Optional | Filter by cluster-type (galera, mongodb, postgresql, …). |
| `vendor` | string | Optional | Filter by vendor (percona, mariadb, …). |
| `db_version` | string | Optional | Filter by database (major.minor) version (5.6, 10.1, …). |

```
{
    "cluster_id": 1,
    "operation": "getRepositories",
    "request_created": "2017-07-31T07:56:02.214Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `repositories` | array | List of repository objects. |
| `total` | integer | Total number of repositories returned. |

**Response example**

```
{
    "repositories": [
    {
        "class_name": "CmonRepository",
        "cluster_type": "galera",
        "db_version": "5.6",
        "modified": "2017-08-01T08:25:29.000Z",
        "name": "percona-5.6-apt-precise",
        "os":
```

```
        {
            "distribution/codename": "unknown",
            "distribution/name": "unknown",
            "distribution/release": "unknown",
            "distribution/type": "debian"
        },
        "path": "/var/tmp/unammed-cmon-repo",
        "used_by_cluster": "",
        "vendor": "percona"
    } ],
    "request_created": "2017-08-01T08:25:35.414Z",
    "request_id": 4,
    "request_processed": "2017-08-01T08:25:35.462Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 1
}
```

### deleteRepository

!!! post "POST /v2/repositories" Removes a software repository registration from the controller and deletes the repository directory from disk. This operation is irreversible.

### getRepositorySetup

!!! post "POST /v2/repositories" Prints out the (semi-)generated repository filename and contents to use the repository manually. You have to substitute the CMON-HOSTNAME string with the controllers IP address before u use it.

### getSupportedSetups

!!! post "POST /v2/repositories" This call will return what possible setups can be used to create/install a new node.

The request does not need a cluster ID or cluster name, the only field that is needed is the "operation" field:

### Request

```
{
    "operation": "getSupportedSetups",
    "request_created": "2017-07-31T09:57:07.145Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `supported_setups` | array | List of supported setup objects, each with cluster_type, db_version, os_version, and vendor. |
| `total` | integer | Total number of supported setups returned. |

**Response example**


```
{
    "request_created": "2017-07-31T09:57:07.145Z",
    "request_id": 3,
    "request_processed": "2017-07-31T09:57:07.188Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "supported_setups": [
    {
        "cluster_type": "mongodb",
        "db_version": "3.2",
        "os_version": "5",
        "vendor": "10gen"
    },

    . . .

    {
        "cluster_type": "replication",
        "db_version": "5.7",
        "os_version": "xenial",
        "vendor": "oracle"
    } ],
    "total": 206
}
```


## 4.12.21 Cmon Events

The `/v2/events` path holds proxied requests to cmon-events.

| Operation | Description |
|---|---|
| *test* | Proxies to the `/test` endpoint in cmon-events. |
| *getSpreadsheetNames* | Returns the list of spreadsheet names available for a cluster. |
| *getSpreadsheet* | Returns the contents of a named spreadsheet for a cluster. |

### test

!!! post "POST /v2/events" This request proxies to the `/test` endpoint in cmon-events.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `name` | string | Required | Name value for the test request. |
| `credentials` | object | Required | Credentials object (can be empty). |

```
{
    "operation": "test",
    "request_created": "2017-06-20T06:19:58.451Z",
    "name": "test",
    "credentials": {
    }
}
```

### getSpreadsheetNames

!!! post "POST /v2/events" Returns the list of spreadsheet names available for a cluster. Spreadsheets are used for capacity planning and other analytical reports. Use the names returned here with `getSpreadsheet` to retrieve the actual data.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |

```
{
    "cluster_id": 1,
    "operation": "getSpreadsheetNames",
    "request_created": "2018-10-01T10:25:27.120Z",
    "request_id": 3
}
```

### getSpreadsheet

!!! post "POST /v2/events" Returns the contents of a named spreadsheet for a cluster. Use `getSpreadsheetNames` first to discover the spreadsheet names available for a given cluster.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `spreadsheet_name` | string | Required | The name of the spreadsheet to retrieve. |

```
{
    "cluster_id": 1,
    "operation": "getSpreadsheet",
    "request_created": "2018-10-02T07:12:36.353Z",
    "request_id": 3,
    "spreadsheet_name": "Capacity Planning"
}
```

### 4.12.22 Statistical Data Calls

The `/v2/stat` provides calls to access various statistical information about the cluster.

| Operation | Description |
|---|---|
| *getInfo* | Get data collected by the sheet manager from the system. |
| *statByName* | Return time-series statistics samples for a named metric across all nodes in a cluster. |
| *getCpuPhysical-Info* | Return physical CPU hardware information for each node in a cluster. |

#### getInfo

!!! post “POST /v2/stat” This call is for getting the data that the sheet manager collected from the system.

#### Request

```
{
    "operation": "getInfo",
    "request_created": "2017-07-27T12:58:00.283Z",
    "request_id": 3
}
```

#### Response

**Response example**

```
{
    "collected_info": [
    {
        "cluster_id": 1,
        "info":
        {
            "cluster.status": 2,
            "cluster.statustext": "Cluster started.",
            "cmon.domainname": "",
            "cmon.hostname": "t7500",
            "cmon.running": true,
            "cmon.starttime": 1501144642,
            "cmon.uptime": 15636,
            "conf.backup_retention": 31,
```

```
            "conf.clusterid": 1,
            "conf.clustername": "ft_postgresql_48154",
            "conf.clustertype": 5,
            "conf.configfile": "/tmp/cmon_1.cnf",
            "conf.hostname": "192.168.1.127",
            "conf.os": "debian",
            "conf.statustext": "Configuration loaded.",
            "host.1.connected": true,
            "host.1.cpu_io_wait_percent": 0.353357,
            "host.1.cpu_steal_percent": 0,
            "host.1.cpu_usage_percent": 15.5331,
            "host.1.cpucores": 16,
            "host.1.cpuinfo": [
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 4,
                "cpumaxmhz": 2.268e+06,
                "cpumhz": 1600,
                "cpumodel": "Intel(R) Xeon(R) CPU           L5520  @␣
↪2.27GHz",
                "cputemp": 55.5,
                "hostid": 1,
                "physical_cpu_id": 0,
                "siblings": 8,
                "vendor": "GenuineIntel"
            },
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 4,
                "cpumaxmhz": 2.268e+06,
                "cpumhz": 1733,
                "cpumodel": "Intel(R) Xeon(R) CPU           L5520  @␣
↪2.27GHz",
                "cputemp": 55.5,
                "hostid": 1,
                "physical_cpu_id": 1,
                "siblings": 8,
                "vendor": "GenuineIntel"
            } ],
            "host.1.cpumaxmhz": 2268,
            "host.1.cpumhz": 1733,
            "host.1.cpumodel": "Intel(R) Xeon(R) CPU           L5520  @␣
↪2.27GHz",
            "host.1.cputemp": 55.5,
            "host.1.devices": [ "/dev/mapper/core1--vg-root" ],
            "host.1.free_disk_bytes": 155814936576,
            "host.1.hostname": "192.168.1.174",
            "host.1.interfaces": [ "eth0" ],
            "host.1.ip": "192.168.1.174",
            "host.1.membuffer": 0,
```

```
            "host.1.memcached": 1,
            "host.1.memfree": 16031900,
            "host.1.memtotal": 16777216,
            "host.1.network_interfaces": [
            {
                "interface_name": "eth0",
                "rx_bytes_per_sec": 11775.8,
                "tx_bytes_per_sec": 17652.3
            } ],
            "host.1.pingdelay": -1,
            "host.1.pingstatustext": "Creating ICMP socket (to ping
↪'192.168.1.174') failed: Operation not permitted.",
            "host.1.port": 8089,
            "host.1.rx_bytes_per_second": 11775.8,
            "host.1.swapfree": 0,
            "host.1.swaptotal": 0,
            "host.1.total_disk_bytes": 208033853440,
            "host.1.tx_bytes_per_second": 17652.3,
            "host.1.uptime": 15779,
            "host.1.wallclock": 1501160299,
            "host.1.wallclocksampled": 1501160235,
            "host.2.class_name": "controller",
            "host.2.connected": true,
            "host.2.cpu_io_wait_percent": 2.89087,
            "host.2.cpu_steal_percent": 0,
            "host.2.cpu_usage_percent": 204.378,
            "host.2.cpucores": 24,
            "host.2.cpuinfo": [
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 6,
                "cpumaxmhz": 2.661e+06,
                "cpumhz": 1596,
                "cpumodel": "Intel(R) Xeon(R) CPU           X5650  @␣
↪2.67GHz",
                "cputemp": 0,
                "hostid": 2,
                "physical_cpu_id": 0,
                "siblings": 12,
                "vendor": "GenuineIntel"
            },
            {
                "class_name": "CmonCpuInfo",
                "cpucores": 6,
                "cpumaxmhz": 2.661e+06,
                "cpumhz": 1596,
                "cpumodel": "Intel(R) Xeon(R) CPU           X5650  @␣
↪2.67GHz",
                "cputemp": 0,
                "hostid": 2,
```

```
                "physical_cpu_id": 1,
                "siblings": 12,
                "vendor": "GenuineIntel"
            } ],
            "host.2.cpumaxmhz": 2661,
            "host.2.cpumhz": 1596,
            "host.2.cpumodel": "Intel(R) Xeon(R) CPU           X5650  @␣
↪2.67GHz",
            "host.2.cputemp": 0,
            "host.2.devices": [ "/dev/sda1" ],
            "host.2.free_disk_bytes": 1409452683264,
            "host.2.hostname": "192.168.1.127",
            "host.2.interfaces": [ "eth0" ],
            "host.2.ip": "192.168.1.127",
            "host.2.membuffer": 318624,
            "host.2.memcached": 19415480,
            "host.2.memfree": 9243460,
            "host.2.memtotal": 49453276,
            "host.2.network_interfaces": [
            {
                "interface_name": "eth0",
                "rx_bytes_per_sec": 29150.4,
                "tx_bytes_per_sec": 18520.8
            } ],
            "host.2.pingdelay": -1,
            "host.2.pingstatustext": "Creating ICMP socket (to ping
↪'192.168.1.127') failed: Operation not permitted.",
            "host.2.port": 9555,
            "host.2.rx_bytes_per_second": 29150.4,
            "host.2.swapfree": 0,
            "host.2.swaptotal": 0,
            "host.2.total_disk_bytes": 2125259440128,
            "host.2.tx_bytes_per_second": 18520.8,
            "host.2.uptime": 2.59852e+06,
            "host.2.version": "1.4.3",
            "host.2.wallclock": 1501160235,
            "host.2.wallclocksampled": 1501160235,
            "license.expires": -1,
            "license.status": false,
            "license.statustext": "No license found.",
            "mail.statustext": "Created.",
            "netStat.1.eth0.rxBytes": 287358005,
            "netStat.1.eth0.txBytes": 205682361,
            "netStat.2.eth0.rxBytes": 531084878595,
            "netStat.2.eth0.txBytes": 503628558465
        }
    } ],
    "request_created": "2017-07-27T12:58:00.283Z",
    "request_id": 3,
    "request_processed": "2017-07-27T12:58:00.328Z",
```

```
    "request_status": "Ok",
    "request_user_id": 3
}
```

### statByName

!!! post "POST /v2/stat" Returns time-series statistics samples for a named metric across all nodes in a cluster. Common metric names include `cpustat`, `memstat`, and `diskstat`.

Filter results to a specific time window using `start_datetime` and `end_datetime`. Set `with_hosts` to `true` to include host metadata alongside each sample, and `calculate_per_sec` to `true` to normalise counter-type values to per-second rates.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `name` | string | Required | The metric name to retrieve (e.g. `cpustat`, `memstat`, `diskstat`). |
| `start_datetime` | string | Optional | Start of the time window to filter results. |
| `end_datetime` | string | Optional | End of the time window to filter results. |
| `with_hosts` | boolean | Optional | Set to `true` to include host metadata alongside each sample. |
| `calculate_per_sec` | boolean | Optional | Set to `true` to normalise counter-type values to per-second rates. |

```
{
    "cluster_id": 1,
    "end_datetime": "15:21",
    "name": "cpustat",
    "operation": "statByName",
    "request_created": "2017-07-27T13:30:41.866Z",
    "request_id": 3,
    "start_datetime": "15:20",
    "with_hosts": true,
    "calculate_per_sec": true
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `data` | array | Array of time-series stat sample objects for the requested metric. |
| `hosts` | array | Array of host objects with metadata for each node (present when `with_hosts` is `true`). |
| `total` | integer | Total number of samples returned. |

**Response example**


```
{
    "data": [
    {
        "busy": 0.00555556,
        "cpuid": 0,
        "cpumhz": 1600,
        "cpumodelname": "Intel(R) Xeon(R) CPU           L5520  @ 2.27GHz",
        "cpuphysicalid": 0,
        "cputemp": 55.5,
        "created": 1501161605,
        "hostid": 1,
        "idle": 0.994444,
        "interval": 5473,
        "iowait": 0,
        "loadavg1": 0.14,
        "loadavg15": 0.09,
        "loadavg5": 0.11,
        "sampleends": 1501161608,
        "samplekey": "CmonCpuStats-1-0",
        "steal": 0,
        "sys": 0.0037037,
        "uptime": 17107,
        "user": 0.00185185
    },

    . . .

    {
        "busy": 0.0980228,
        "cpuid": 23,
        "cpumhz": 1596,
        "cpumodelname": "Intel(R) Xeon(R) CPU           X5650  @ 2.67GHz",
        "cpuphysicalid": 0,
        "cputemp": 0,
        "created": 1501161658,
        "hostid": 2,
        "idle": 0.901977,
        "interval": 5379,
```

```
        "iowait": 0,
        "loadavg1": 2.35,
        "loadavg15": 2.23,
        "loadavg5": 2.33,
        "sampleends": 1501161661,
        "samplekey": "CmonCpuStats-2-23",
        "steal": 0,
        "sys": 0.0247148,
        "uptime": 2.5999e+06,
        "user": 0.073308
    } ],
    "hosts": [
    {
        "class_name": "CmonPostgreSqlHost",
        "clusterid": 1,
        "configfile": [ "/etc/postgresql/9.6/main/postgresql.conf" ],
        "connected": true,
        "data_directory": "/var/lib/postgresql/9.6/main",
        "datadir": "/var/lib/postgresql/9.6/main",
        "description": "",
        "distribution":
        {
            "codename": "xenial",
            "name": "ubuntu",
            "release": "16.04",
            "type": "debian"
        },
        "hostId": 1,
        "hostname": "192.168.1.174",
        "hoststatus": "CmonHostOnline",
        "hot_standby": true,
        "ip": "192.168.1.174",
        "lastseen": 1501162240,
        "logfile": "/var/log/postgresql/postgresql-9.6-main.log[0m",
        "maintenance_mode_active": false,
        "message": "Up and running",
        "nodetype": "postgres",
        "pid": 5247,
        "pidfile": "/var/run/postgresql/9.6-main.pid",
        "pingstatus": 0,
        "pingtime": 0,
        "port": 8089,
        "readonly": false,
        "received_location": "0/1523B10",
        "replay_location": "0/1523B10",
        "role": "master",
        "sshfailcount": 0,
        "ssl_certs":
        {
            "server":
```

```
            {
                "ca": "",
                "id": 0,
                "key": "/etc/ssl/private/ssl-cert-snakeoil.key",
                "path": "/etc/ssl/certs/ssl-cert-snakeoil.pem"
            }
        },
        "timestamp": 1501162240,
        "unique_id": 1,
        "uptime": 17592,
        "version": "9.6.3",
        "wallclock": 1501162260,
        "wallclocktimestamp": 1501162197
    },
    {
        "class_name": "CmonHost",
        "clusterid": 1,
        "configfile": "/tmp/cmon_1.cnf",
        "connected": true,
        "distribution":
        {
            "codename": "trusty",
            "name": "ubuntu",
            "release": "14.04",
            "type": "debian"
        },
        "hostId": 2,
        "hostname": "192.168.1.127",
        "hoststatus": "CmonHostOnline",
        "ip": "192.168.1.127",
        "lastseen": 1501162240,
        "logfile": "/tmp/cmon_1.log",
        "maintenance_mode_active": false,
        "message": "Up and running",
        "nodetype": "controller",
        "pid": 48111,
        "pingstatus": 0,
        "pingtime": 0,
        "port": 9555,
        "role": "controller",
        "timestamp": 1501162240,
        "unique_id": 2,
        "uptime": 17759,
        "version": "1.4.3",
        "wallclock": 1501162197,
        "wallclocktimestamp": 1501162197
    } ],
    "reply_received": "2017-07-27T13:30:41.871Z",
    "request_created": "2017-07-27T13:30:41.866Z",
    "request_id": 3,
```

```
    "request_processed": "2017-07-27T13:30:41.929Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 384
}
```


### getCpuPhysicalInfo

!!! post "POST /v2/stat" Returns physical CPU hardware information for each node in a cluster. Each entry in the `data` array represents one physical CPU socket on one host, and includes the CPU model name, core count, clock speed, socket topology, and temperature reading.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |


```
{
    "cluster_id": 1,
    "operation": "getCpuPhysicalInfo",
    "request_created": "2017-07-27T13:43:47.802Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `data` | array | Array of physical CPU info objects, one per CPU socket per host. |
| `total` | integer | Total number of CPU socket entries returned. |

**Response example**


```
{
    "data": [
    {
        "class_name": "CmonCpuInfo",
        "cpucores": 4,
        "cpumaxmhz": 2.268e+06,
        "cpumhz": 1600,
        "cpumodel": "Intel(R) Xeon(R) CPU           L5520  @ 2.27GHz",
        "cputemp": 55.5,
        "hostid": 1,
        "physical_cpu_id": 0,
        "siblings": 8,
        "vendor": "GenuineIntel"
```

```
    },
    {
        "class_name": "CmonCpuInfo",
        "cpucores": 4,
        "cpumaxmhz": 2.268e+06,
        "cpumhz": 1600,
        "cpumodel": "Intel(R) Xeon(R) CPU           L5520  @ 2.27GHz",
        "cputemp": 55.5,
        "hostid": 1,
        "physical_cpu_id": 1,
        "siblings": 8,
        "vendor": "GenuineIntel"
    } ],
    "request_created": "2017-07-27T13:43:47.802Z",
    "request_id": 3,
    "request_processed": "2017-07-27T13:43:47.848Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 2
}
```


The caller should specify the hostname and cluster id where cmonagent is deployed; Along with limit (maximum number of data to retrieve), origin e.g. “timeline”, and begin and end date - range of queries to retrieve.

Example:

```
curl -k 'https://127.0.0.1:9501/v2/stat' -XPOST -d '{"hostname":
↪"10.10.10.10", "cluster_id": 123, "operation": "gettopqueries", "limit":␣
↪10, "begin": "2021-06-01T09:48:31.970Z", "end" :
↪"2021-06-07T09:48:31.970Z", "origin" : "timeline"}' -b cookies.jar
```

```
{"hostname": "10.10.10.10", "cluster_id": 123, "operation": "getTopQueries",
↪ "limit": 10, "begin": "2021-06-01T09:48:31.970Z", "end" :
↪"2021-06-07T09:48:31.970Z", "origin" : "timeline"}
```

### getTopQueries

!!! post “POST /v2/stat” Returns top query statistics for a host over a time range.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `hostname` | string | Required | The database host to query. |
| `requestType` | string | Required | Request type, e.g. `RequestRead`. |
| `requestData` | object | Required | Read parameters object. |
| `requestData.begin` | string | Required | Start of the time range (ISO 8601). |
| `requestData.end` | string | Required | End of the time range (ISO 8601). |
| `requestData.limit` | integer | Required | Maximum number of results to return. |
| `requestData.mimeType` | string | Required | Response MIME type, e.g. `application/json`. |
| `requestData.objectName` | string | Required | Object path, e.g. `/topQueryMonitor`. |
| `requestData.origin` | string | Required | Query origin, e.g. `timeline`. |

```
{
    "operation": "getTopQueries",
    "hostname": "dbhost1",
    "className": "CmnRequest",
    "requestCreated": "2021-06-07T09:41:43.636Z",
    "requestType": "RequestRead",
    "requestData":
    {
        "begin": "021-06-01T09:48:31.970Z",
        "className": "CmnReadParam",
        "end": "021-06-07T09:48:31.970Z",
        "limit": 10,
        "mimeType": "application/json",
        "objectName": "/topQueryMonitor",
        "origin": "timeline"
    }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `result` | object | Result container. |
| `result.readParam` | object | Echo of the read parameters used. |
| `result.resultData` | object | SQL result object. |
| `result.resultData.columnNames` | array | List of column names in the result set. |
| `result.resultData.records` | array | Array of result rows, each matching the column order. |
| `result.resultData.created` | string | Timestamp when the result was created. |
| `result.resultData.databaseName` | string | Database connection name used. |
| `result.resultData.sqlServerName` | string | SQL server name. |

**Response example**


```
{
    "className": "CmnReply",
    "requestCreated": "2021-06-07T09:51:44.255Z",
    "requestProcessed": "2021-06-07T09:51:44.243+00:00",
    "requestReceived": "2021-06-07T09:51:44.232+00:00",
    "requestStatus": "OK",
    "result":
    {
        "className": "CmnReadResult",
        "readParam":
        {
            "begin": "2021-06-01T09:48:31.970Z",
            "className": "CmnReadParam",
            "end": "2021-06-07T09:48:31.970Z",
            "limit": 1,
            "mimeType": "application/json",
            "objectName": "/topQueryMonitor",
            "origin": "timeline"
        },
        "resultData":
        {
            "className": "CmnSqlResult",
            "created": "2021-06-07T09:51:44.233+00:00",
            "databaseName": ".storageConnection",
            "sqlServerName": "localhost",
            "columnNames":
            [
                "created",
                "server",
```

```
                "schema",
                "count",
                "sumTime",
                "sumIoTime",
                "shrdBlcks",
                "localBlcks",
                "tempBlcks"
            ],
            "records":
            [
                [
                    "2021-06-07T09:41:12.915Z",
                    "127.0.0.1",
                    "sbtest",
                    1,
                    773514,
                    0,
                    0,
                    0,
                    0
                ]
            ]
        }
    }
}
```


Returns historical database size statistics for a cluster, which returns the growth trend of a database in order to assist with capacity planning.

### getDbGrowth

!!! post "POST /v2/stat" Returns historical database size statistics for a cluster, allowing you to track data growth over time. Results include per-host datadir size samples with timestamps and an optional per-database breakdown. Filter the time window using `startdate` and `enddate`, or request a specific day with `dayofyear` and `year`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `startdate` | string | Optional | Start of the time window to filter results. |
| `enddate` | string | Optional | End of the time window to filter results. |
| `dayofyear` | integer | Optional | Specific day of year to request data for. |
| `year` | integer | Optional | Year corresponding to `dayofyear`. |
| `include_tables` | boolean | Optional | Whether to include per-table breakdown. |
| `db` | string | Optional | Filter results to a specific database name. |

```
{"operation":"getDbGrowth", "cluster_id": 1}
```

### Response

| Field | Type | Description |
|---|---|---|
| total | integer | Total number of records returned. |
| data | array | List of database growth data objects. |
| data[].created | string | Timestamp of the sample. |
| data[].database_count | integer | Number of databases on the host. |
| data[].datadir | string | Path to the data directory. |
| data[].free_datadir_size | integer | Free space in the data directory in bytes. |
| data[].hostname | string | Hostname of the database node. |
| data[].port | integer | Port of the database node. |
| data[].total_datadir_size | integer | Total size of the data directory in bytes. |
| data[].year | integer | Year of the sample. |
| data[].yearday | integer | Day of year of the sample. |
| data[].dbs | array | Per-database breakdown (if requested). |

**Response example**


```
{
    "controller_id": "a19271f7-3dd7-4f6b-89a4-7b84b6fb0bf2",
    "request_processed": "2021-09-27T18:44:32.327Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 1,
    "data":
    [
        {
            "class_name": "CmonDbStats",
            "created": "Sep 18 15:13:36",
            "database_count": 0,
            "datadir": "/var/lib/mysql/",
            "free_datadir_size": 406810349568,
            "hostname": "10.63.228.75",
            "port": 3306,
            "total_datadir_size": 458595631104,
            "year": 2021,
            "yearday": 261,
            "dbs": []
        }
    ],
    "debug_messages":
    [
        "RPC V2 authenticated user is 'admin'."
    ]
}
```

### prometheus

!!! post "POST /v2/stat" The following URL path (`/v2/stat/prometheus`) will be proxied to the prometheus daemon, you have to pass additional URI paths like `/query`.

```
Mandatory argument: `cluster_id` or `cluster_name` to identify the cluster.
```

The caller could specify/select here a Prometheus monitor instance to be used for processing the query, if not specified the code will takes the first Online CmonPrometheusHost instance to process the query.

**Note**

Note Only the following arguments will be sent to Prometheus: "query", "time", "timeout", "start", "end", "step", "match[]"

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster (or use `cluster_name`). |
| `cluster_name` | string | Optional | The name of the cluster (alternative to `cluster_id`). |
| `queries` | array | Optional | Array of query objects with `query` and `step` fields. |
| `start` | integer | Optional | Start of the time range (Unix epoch). |
| `end` | integer | Optional | End of the time range (Unix epoch). |
| `step` | integer | Optional | Step interval in seconds. |
| `returnfrom` | integer | Optional | Offset for result pagination. |

```
{
"operation": "/prometheus",
"queries":[
   {
      "query":
↪"mysql_global_status_wsrep_cert_deps_distance{instance=~\"10.117.12.165(:[0-9]+)?\"}",
↪
      "step":10
   }
],
"start":1622105617,
"end":1622108317,
"step":10,
"returnfrom":0,
"cluster_id":1
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of results returned. |
| `data` | array | Array of Prometheus response objects. |

**Response example**


```
{
 "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
 "request_processed": "2021-05-27T09:51:45.311Z",
 "request_status": "Ok",
 "request_user_id": 6,
 "total": 1,
 "data":
 [
     {"status":"success","data":{"resultType":"vector","result":[{
↪"metric":{"__name__":"mysql_global_status_wsrep_cert_deps_distance",
↪"instance":"10.117.12.165:9104","job":"mysqld","monitors":
↪"10.117.12.165:3306"},"value":[1622109105.311,"0"]}]}}
 ],
 "debug_messages":
 [
     "RPC V2 authenticated user is 'kedz'."
 ]
}
```


### qm_topqueries

!!! post "POST /v2/stat" Returns the top queries captured by the Query Monitor for a cluster, ranked by total execution time. Supports filtering by `username` and `db_name`, and pagination via `limit` and `offset`. Pass `-1` for `limit` or `offset` to return all results without pagination.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `limit` | integer | Required | Maximum number of results; pass `-1` for all. |
| `offset` | integer | Required | Pagination offset; pass `-1` for all. |
| `username` | string | Optional | Filter results to a specific database user. |
| `db_name` | string | Optional | Filter results to a specific database name. |


```
{
  "operation": "qm_topqueries",
  "cluster_id": 107,
  "limit": -1,
  "offset": -1,
  "username": "root",
  "db_name": "test"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| total | integer | Total number of query records returned. |
| available_databases | array | List of database names available for filtering. |
| available_users | array | List of usernames available for filtering. |
| data | object | Container for query results. |
| data.total_query_exec_time_us | integer | Total execution time in microseconds across all queries. |
| data.queries | array | List of top query records. |

**Response example**


```
{
    "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
    "is_superuser": true,
    "request_processed": "2022-12-05T14:48:48.548Z",
    "request_status": "Ok",
    "total": 2,
    "available_databases": ["aaa", "bbb"],
    "available_users": ["root", "admin"],
    "controllers": {},
    "data": {
        "total_query_exec_time_us": 3587694,
        "queries": [
            {
                "class_name": "CmonQueryMonRecord",
                "avg_query_time": 620268,
                "canonical": "COMMIT",
                "command": "",
                "count": 3,
                "db": "dcps",
                "host": "localhost",
                "host_id": 1101,
                "hostid": 1101,
                "info": "COMMIT",
                "last_seen": 1634133907,
                "lock_time": 0,
                "max_query_time": 685060,
                "min_query_time": 0,
                "query_id": 10201897797591768868,
                "query_time": 685060,
                "rows_examined": 0,
                "rows_sent": 0,
                "state": "",
                "stddev": 48187,
                "sum_created_tmp_disk_tables": 0,
                "sum_created_tmp_tables": 0,
```

```
                "sum_lock_time": 0,
                "sum_no_good_index_used": 0,
                "sum_no_index_used": 0,
                "sum_query_time": 1860804,
                "sum_rows_examined": 0,
                "sum_rows_sent": 0,
                "user": "cmon[cmon]",
                "variance": 2322006804
            },
            {
                "class_name": "CmonQueryMonRecord",
                "avg_query_time": 575630,
                "canonical": "UPDATE `dcps`.`cake_sessions` SET `id` = ?,␣
↪`data` = ?, `expires` = ? ",
                "command": "",
                "count": 3,
                "db": "dcps",
                "host": "localhost",
                "host_id": 1101,
                "hostid": 1101,
                "info": "UPDATE `dcps`.`cake_sessions` SET `id` =
↪'mqivh99fpnav209eomfouo4mi1', `data`",
                "last_seen": 1634133910,
                "lock_time": 160,
                "max_query_time": 602311,
                "min_query_time": 0,
                "query_id": 9449288033917105119,
                "query_time": 602311,
                "rows_examined": 1,
                "rows_sent": 0,
                "state": "",
                "stddev": 10766,
                "sum_created_tmp_disk_tables": 0,
                "sum_created_tmp_tables": 0,
                "sum_lock_time": 160,
                "sum_no_good_index_used": 0,
                "sum_no_index_used": 0,
                "sum_query_time": 1726890,
                "sum_rows_examined": 2,
                "sum_rows_sent": 0,
                "user": "cmon[cmon]",
                "variance": 115915538
            }
        ]
    }
}
```

### qm_queryoutliers

!!! post "POST /v2/stat" Returns individual query executions that exceeded the outlier threshold configured in the Query Monitor. Filter results by time range using `starttime` and `endttime` (Unix epoch timestamps), and paginate using `limit` and `offset`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `starttime` | integer | Optional | Start of the time range (Unix epoch timestamp). |
| `endttime` | integer | Optional | End of the time range (Unix epoch timestamp). |
| `limit` | integer | Required | Maximum number of results; pass `-1` for all. |
| `offset` | integer | Required | Pagination offset; pass `-1` for all. |

```
{
  "operation": "qm_queryoutliers",
  "cluster_id": 107,
  "starttime": 1670000000,
  "endttime": 1670252422,
  "limit": -1,
  "offset": -1
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of outlier query records returned. |
| `data` | object | Container for query results. |
| `data.queries` | array | List of outlier query records. |

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2022-12-05T15:00:58.305Z",
  "request_status": "Ok",
  "total": 0,
  "controllers": { },
  "data":
  {
      "queries": []
  }
}
```

### qm_purge

!!! post "POST /v2/stat" Purges all slow-query data collected by the Query Monitor for a cluster. This clears the accumulated query history. Use with caution — the data cannot be recovered after purging.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster whose query monitor data should be purged. |

```
{
  "operation": "qm_purge",
  "cluster_id": 107,
}
```

### Response

**Response example**

```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2022-12-05T14:46:24.323Z",
  "request_status": "Ok"
}
```

### qm_killprocess

!!! post "POST /v2/stat" Terminates a specific database backend process identified by its `pid` on the node given by `host_port`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `pid` | integer | Required | Process ID of the database backend process to terminate. |
| `host_port` | array | Required | Array containing the host:port string of the target node. |

```
{
  "operation": "qm_killprocess",
  "cluster_id": 107,
  "pid": 916805,
  "host_port": [ "10.228.47.220:5432" ]
}
```

### Response

**Response example**


```
{
  "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
  "is_superuser": true,
  "request_processed": "2022-12-05T14:46:24.323Z",
  "request_status": "Ok"
}
```


## qm_processes

!!! post "POST /v2/stat" Returns the list of currently active database backend processes for a cluster as seen by the Query Monitor. Supports pagination via `limit` and `offset`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `limit` | integer | Required | Maximum number of results to return. |
| `offset` | integer | Required | Pagination offset. |


```
{"operation":"qm_processes","cluster_id": 107,"limit":1,"offset":0}
```


### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of active processes. |
| `data` | array | List of database backend process objects. |
| `data[].appName` | string | Application name connected to the backend. |
| `data[].backendPid` | integer | Backend process ID. |
| `data[].backendStart` | string | Timestamp when the backend started. |
| `data[].client` | string | Client address and port. |
| `data[].databaseName` | string | Name of the database the backend is connected to. |
| `data[].elapsedTime` | string | Elapsed time since the current query started. |
| `data[].hostId` | integer | Internal host ID of the node. |
| `data[].hostname` | string | Hostname of the node. |
| `data[].node` | string | Node address in host:port format. |
| `data[].port` | integer | Port of the node. |
| `data[].query` | string | Currently executing query text. |
| `data[].queryStart` | string | Timestamp when the current query started. |
| `data[].reportTs` | integer | Unix timestamp of the report. |
| `data[].state` | string | Backend state. |
| `data[].userName` | string | Database user name. |
| `data[].waiting` | string | Lock or wait condition the backend is waiting on. |
| `data[].xactStart` | string | Timestamp when the current transaction started. |

**Response example**


```
{
      "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
      "is_superuser": true,
      "request_processed": "2022-12-05T14:37:36.259Z",
      "request_status": "Ok",
      "total": 6,
      "controllers": { },
      "data":
      [
          {
              "class_name": "CmonPostgreSqlDbProcess",
              "appName": "",
              "backendPid": 440,
              "backendStart": "2022-11-17 15:22:04.460071+01",
              "client": "localhost:0",
              "databaseName": "",
              "elapsedTime": "",
              "hostId": 7006,
              "hostname": "10.228.47.220",
              "node": "10.228.47.220:5432",
              "port": 5432,
              "query": "",
              "queryStart": "",
              "reportTs": 1670251052,
              "state": "",
              "userName": "postgres",
              "waiting": "LogicalLauncherMain",
              "xactStart": ""
          }
    ]
}
```


### gettxdeadlock

!!! post "POST /v2/stat" Returns recent transaction deadlock events detected across the cluster nodes. Each entry describes the SQL statement involved, the host and instance where the deadlock occurred, and the IDs of the conflicting transactions.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |
| `limit` | integer | Required | Maximum number of results to return. |
| `offset` | integer | Required | Pagination offset. |

```
{"operation":"gettxdeadlock","cluster_id": 107,"limit":1,"offset":0}
```

### Response

| Field | Type | Description |
|---|---|---|
| total | integer | Total number of deadlock records returned. |
| data | array | List of deadlock event objects. |
| data[].blocked_by_trx_id | string | Transaction ID that blocked this transaction. |
| data[].db | string | Database name where the deadlock occurred. |
| data[].duration | string | Duration of the transaction in seconds. |
| data[].host | string | Host where the deadlock was detected. |
| data[].hostId | integer | Internal host ID of the node. |
| data[].info | string | SQL statement involved in the deadlock. |
| data[].innodb_trx_id | string | InnoDB transaction ID. |
| data[].instance | string | Node instance in host:port format. |
| data[].internal_trx_id | string | Internal transaction ID. |
| data[].lastseen | integer | Unix timestamp when the deadlock was last seen. |
| data[].message | string | Deadlock message. |
| data[].mysql_trx_id | string | MySQL transaction ID. |
| data[].port | integer | Port of the node. |
| data[].sql | string | SQL statement that caused the deadlock. |
| data[].state | string | Transaction state at the time of the deadlock. |
| data[].user | string | Database user that executed the statement. |

**Response example**


```
{
    "controller_id": "7f399ff3-7484-4a93-a47a-a5c0bba42fca",
     "is_superuser": true,
     "request_processed": "2022-12-05T14:37:36.259Z",
     "request_status": "Ok",
     "total": 6,
     "controllers": { },
     "data": [
         {
             "blocked_by_trx_id": "",
             "db": "classicmodels",
             "duration": "45",
             "host": "localhost",
             "hostId": 2,
             "info": "DELETE FROM t WHERE i = 1",
             "innodb_trx_id": "",
             "instance": "10.0.8.12:3306",
             "internal_trx_id": "",
             "lastseen": 1696620029,
             "message": "DELETE FROM t WHERE i = 1",
             "mysql_trx_id": "23",
             "port": 3306,
```

```
            "sql": "DELETE FROM t WHERE i = 1",
            "state": "updating",
            "user": "root"
        }
    ]
}
```

### nodevariables

!!! post "POST /v2/stat" Returns the current runtime variables for each node in a cluster (equivalent to `SHOW VARIABLES` for MySQL/MariaDB nodes). Nodes that have not yet reported their variables appear in the list with a `null` `report_ts`.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |

```
{"operation": "nodevariables", "cluster_id": 399}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of node entries returned. |
| `data` | array | List of per-node variable objects. |
| `data[].hostId` | integer | Internal host ID of the node. |
| `data[].hostname` | string | Hostname of the node. |
| `data[].port` | integer | Port of the node. |
| `data[].report_ts` | integer | Unix timestamp of the last variable report, or `null` if not yet reported. |
| `data[].unique_id` | integer | Unique identifier for the node entry. |
| `data[].variables` | object | Key-value map of runtime variable names and their current values (present when variables have been reported). |

**Response example**

```
{
    "controller_id": "42e4370a-cae2-4b51-aa29-63d7fbfeacb2",
    "request_processed": "2023-10-16T15:56:58.857Z",
```

```
    "request_status": "Ok",
    "request_user_id": 5,
    "total": 14,
    "data":
    [
        {
            "hostId": 2611,
            "hostname": "10.238.5.169",
            "port": 27017,
            "report_ts": null,
            "unique_id": 1251
        },
        {
            "hostId": 2612,
            "hostname": "10.238.5.25",
            "port": 27017,
            "report_ts": null,
            "unique_id": 1252
        },
        {
            "hostId": 2613,
            "hostname": "10.238.5.5",
            "port": 27017,
            "report_ts": null,
            "unique_id": 1253
        },
        {
            "hostId": 2675,
            "hostname": "10.238.5.17",
            "port": 3306,
            "report_ts": 1697471809,
            "unique_id": 1275,
            "variables":
            {
                "allow_suspicious_udfs": "OFF",
                "alter_algorithm": "DEFAULT",
                "analyze_sample_percentage": "100.000000",
                ".......": ".........",
                ".......": ".........",
                "wsrep_trx_fragment_size": "0",
                "wsrep_trx_fragment_unit": "bytes"
            }
        },
        {
            "hostId": 2676,
            "hostname": "10.238.5.112",
            "port": 3306,
            "report_ts": 1697471809,
            "unique_id": 1276,
            "variables":
```

```
            {
                "allow_suspicious_udfs": "OFF",
                "alter_algorithm": "DEFAULT",
                "analyze_sample_percentage": "100.000000",
                ".......": ".........",
                ".......": ".........",
                "wsrep_trx_fragment_size": "0",
                "wsrep_trx_fragment_unit": "bytes"
            }
        },
        {
            "hostId": 2677,
            "hostname": "10.238.5.23",
            "port": 3306,
            "report_ts": 1697471809,
            "unique_id": 1277,
            "variables":
            {
                "allow_suspicious_udfs": "OFF",
                "alter_algorithm": "DEFAULT",
                "analyze_sample_percentage": "100.000000",
                ".......": ".........",
                ".......": ".........",
                "wsrep_trx_fragment_size": "0",
                "wsrep_trx_fragment_unit": "bytes"
            }
        }
    ],
    "debug_messages":
    [
        "RPC V2 authenticated user is 's9s'."
    ]
}
```


### getdbstatus

!!! post "POST /v2/stat" Returns the current runtime status variables for each node in a cluster (equivalent to `SHOW STATUS` for MySQL/MariaDB nodes). Useful for monitoring engine-level counters, replication state, and cache hit rates.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_id` | integer | Required | The ID of the cluster. |

```
{"operation": "getdbstatus", "cluster_id": 399}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | Total number of node entries returned. |
| `data` | array | List of per-node status objects. |
| `data[].hostId` | integer | Internal host ID of the node. |
| `data[].hostname` | string | Hostname of the node. |
| `data[].statuses` | object | Key-value map of status variable names and their current values, plus a `report_ts` timestamp field. |

**Response example**


```
{
    "controller_id": "42e4370a-cae2-4b51-aa29-63d7fbfeacb2",
    "request_processed": "2023-10-16T15:57:20.493Z",
    "request_status": "Ok",
    "request_user_id": 5,
    "total": 3,
    "data":
    [
        {
            "hostId": 2675,
            "hostname": "10.238.5.17",
            "statuses":
            {
                "class_name": "CmonSqlVariables",
                "ACL_DATABASE_GRANTS": "2",
                ".......": ".......",
                ".......": ".......",
                "WSREP_LOCAL_INDEX": "18446744073709551615",
                "report_ts": 1697471840
            }
        },
        {
            "hostId": 2676,
            "hostname": "10.238.5.112",
            "statuses":
            {
                "class_name": "CmonSqlVariables",
                "ACL_DATABASE_GRANTS": "2",
                ".......": ".......",
                ".......": ".......",
                "WSREP_LOCAL_INDEX": "18446744073709551615",
                "report_ts": 1697471840
            }
        },
```

```
        {
            "hostId": 2677,
            "hostname": "10.238.5.23",
            "statuses":
            {
                "class_name": "CmonSqlVariables",
                "ACL_DATABASE_GRANTS": "2",
                ".......": ".......",
                ".......": ".......",
                "WSREP_LOCAL_INDEX": "18446744073709551615",
                "report_ts": 1697471840
            }
        }
    ],
    "debug_messages":
    [
        "RPC V2 authenticated user is 's9s'."
    ]
}
```


### 4.12.23 CMON Directory Tree

The `/v2/tree` path is for managing objects in the CMON Directory Tree.

| Operation | Description |
|---|---|
| *getTree* | Get the CDT (CMON Directory Tree) or a sub-tree. |
| *move* | Move an object into a new location in the tree. |
| *rename* | Change the name of a CDT entry. |
| *getAcl* | Get the ACL of an object from the CMON Directory Tree. |
| *cat* | Read the content of a CDT file. |
| *getObject* | Get an object using its full path as a reference. |
| *addAcl* | Add an ACL entry to an object. |
| *removeAcl* | Remove an ACL entry from the ACL of an object. |
| *checkAccess* | Check if the authenticated user has access to a given object. |
| *chown* | Change the ownership of object(s). |
| *mkdir* | Create folders in the CMON Directory Tree. |
| *rmdir* | Remove empty folders from the CMON Directory Tree. |
| *mkfile* | Create a file type CDT entry. |
| *setContent* | Change the content string of a CDT entry. |
| *delete* | Delete an object from the tree. |

### getTree

!!! post "POST /v2/imperative" This call can be used to get the CDT (CMON Directory Tree).

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `path` | string | Optional | The path of the sub-tree to get. This is not mandatory, if it is not provided the entire tree will be returned (or at least the part that the user allowed to see). |

```
{
    "operation": "getTree",
    "path": "/galera_001",
    "filters": [ "Cluster", "User", "/some/path" ],
    "request_created": "2017-10-03T07:12:13.682Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `cdt` | object | The CMON Directory Tree organized into a tree of objects. This is either the root node of the tree or the node representing the path that was requested. |

**Response example**

```
{
    "cdt":
    {
        "item_acl": "user::rwx,group::rw-,other::---",
        "item_name": "galera_001",
        "item_path": "/",
        "item_type": "Cluster",
        "object_id": 1,
        "owner_group_id": 2,
        "owner_group_name": "users",
        "owner_user_id": 3,
        "owner_user_name": "pipas",
        "sub_items": [
        {
            "item_acl": "user::rwx,group::rw-,other::---",
            "item_name": "192.168.1.212:3306",
            "item_path": "/galera_001",
            "item_type": "Node",
            "object_id": 2,
```

```
            "owner_group_id": 1,
            "owner_group_name": "admins",
            "owner_user_id": 1,
            "owner_user_name": "system"
        },
        {
            "item_acl": "user::rwx,group::rw-,other::---",
            "item_name": "192.168.1.127:9555",
            "item_path": "/galera_001",
            "item_spec": "controller",
            "item_type": "Node",
            "object_id": 3,
            "owner_group_id": 1,
            "owner_group_name": "admins",
            "owner_user_id": 1,
            "owner_user_name": "system"
        } ]
    },
    "request_created": "2017-10-03T07:12:13.682Z",
    "request_id": 3,
    "request_processed": "2017-10-03T07:12:13.750Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### move

!!! post "POST /v2/imperative" This call can be used to move an object into a new location in the tree.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| source_path | string | Required | The full path of the object to move. |
| target_path | string | Required | The new location where the object will be moved. |


```
{
    "operation": "move",
    "request_created": "2017-10-03T07:51:53.285Z",
    "request_id": 3,
    "source_path": "/storage01",
    "target_path": "/servers"
}
```

### Response

The reply of this request is quite simple, it just states the request was processed.

**Response example**


```
{
    "request_created": "2017-10-03T07:51:53.285Z",
    "request_id": 3,
    "request_processed": "2017-10-03T07:51:53.400Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### rename

!!! post "POST /v2/imperative" Changes the name of a CDT entry.

So for example if one wants to change the name of a user group one needs to get the group. Then the "cdt_path" and the "group_name" property of the group has to be accessed to build the path of the group object and the "rename" request has to be sent.

### Request


```
{
    "operation": "rename",
    "request_created": "2019-01-08T09:18:04.315Z",
    "request_id": 62,
    "source_path": "/groups/mygroup",
    "target_name": "newgroup"
}
```


### Response

Here is a reply given for the "rename" request if the operation was successful.

**Response example**


```
{
    "request_created": "2019-01-08T09:18:04.315Z",
    "request_id": 62,
    "request_processed": "2019-01-08T09:18:04.343Z",
    "request_status": "Ok",
    "request_user_id": 1
}
```


### getAcl

!!! post "POST /v2/imperative" This call can be used to get the ACL of an object from the CMON Directory Tree.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| path | string | Required | The full path of the object. |


```
{
    "operation": "getAcl",
    "path": "/core1/containers/bestw_controller",
    "request_created": "2017-10-03T06:39:59.289Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| acl | string | The ACL for the requested object in string format. |
| object_name | string | The name of the object. |
| object_path | string | The path in the CDT where the object can be found. |
| owner_group_name | string | The group name of the group owner for the object. |
| owner_user_name | string | The user name of the owner of the group. |

**Response example**


```
{
    "acl": "user::rwx,group::rw-,other::---",
    "object_name": "bestw_controller",
    "object_path": "/core1/containers",
    "owner_group_name": "users",
    "owner_user_name": "pipas",
    "request_created": "2017-10-03T06:39:59.289Z",
```

```
    "request_id": 3,
    "request_processed": "2017-10-03T06:40:02.358Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### cat

!!! post "POST /v2/imperative" This call reads the content of a CDT file.

### Request

```
{
    "operation": "cat",
    "path": "/.issue",
    "request_created": "2017-12-18T13:22:45.664Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
    "file_content": "Severalnines Clustercontrol Server.\n",
    "request_created": "2017-12-18T13:22:45.664Z",
    "request_id": 3,
    "request_processed": "2017-12-18T13:22:45.681Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### getObject

!!! post "POST /v2/imperative" This call can be used to get the object using its full path as a reference.

### Request

```
{
    "operation": "getObject",
    "path": "/.runtime/cluster_manager",
    "request_created": "2018-11-29T09:09:22.836Z",
    "request_id": 2
}
```

### Response

**Response example**


```
{
    "object":
    {
        "acl": "user::r--,group::---,other::---",
        "cdt_path": "/.runtime",
        "class_name": "CmonCdtEntry",
        "content": "        cluster_manager_instance :␣
↪0x7fad58001380\n\n",
        "dirty": false,
        "major_device_number": 10,
        "minor_device_number": 1,
        "name": "cluster_manager",
        "owner_group_id": 1,
        "owner_group_name": "admins",
        "owner_user_id": 1,
        "owner_user_name": "system",
        "type": "CmonCdtFile"
    },
    "reply_received": "2018-11-29T09:09:22.841Z",
    "request_created": "2018-11-29T09:09:22.836Z",
    "request_id": 2,
    "request_processed": "2018-11-29T09:09:22.864Z",
    "request_status": "Ok",
    "request_user_id": 1
}
```


### addAcl

!!! post "POST /v2/imperative" This call can be used to add an ACL entry to an object. Please note that if the given entry is already added to the ACL it will be overwritten. This is the intuitive thing to do.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `acl` | string | Required | The ACL to add. This should be the text format of one ACL entry. |
| `path` | string | Required | The full path of the object to change. |

```
{
    "acl": "user:nobody:r--",
    "operation": "addAcl",
    "path": "/core1/containers/bestw_controller",
    "request_created": "2017-10-03T06:43:10.458Z",
```

(continues on next page)

(continued from previous page)

```
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| acl | string | The ACL of the object after the change. |
| object_name | string | The name of the object. |
| object_path | string | The path in the CDT where the object can be found. |
| owner_group_name | string | The group name of the group owner for the object. |
| owner_user_name | string | The user name of the owner of the group. |

**Response example**

```
{
    "acl": "user::rwx,user:nobody:r--,group::rw-,other::---",
    "object_name": "bestw_controller",
    "object_path": "/core1/containers",
    "owner_group_name": "users",
    "owner_user_name": "pipas",
    "request_created": "2017-10-03T06:46:42.233Z",
    "request_id": 3,
    "request_processed": "2017-10-03T06:46:42.322Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### removeAcl

!!! post "POST /v2/imperative" This call can be used to remove an ACL entry from the ACL of an object. Currently only one single ACL entry can be removed at a a time.

Please note that certain ACL entries are mandatory and can not be removed only changed. If the call tries to remove such a mandatory entry an error will be reported.

### Request

```
{
    "acl": "user:pipas:rwx",
    "operation": "removeAcl",
    "path": "/",
    "request_created": "2017-10-04T15:07:50.427Z",
    "request_id": 3
}
```

### Response

Here is a reply notifying the caller about the successful removal of the entry.

**Response example**


```
{
    "acl": "user::rwx,group::rwx,other::rwx",
    "object_name": "",
    "object_path": "/",
    "owner_group_name": "admins",
    "owner_user_name": "system",
    "request_created": "2017-10-04T15:07:50.427Z",
    "request_id": 3,
    "request_processed": "2017-10-04T15:07:50.504Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


## checkAccess

!!! post "POST /v2/imperative" This RPC call can be used to check if the authenticated user has access to the given object.

The reply will simply be an "Ok" reply if the user has the access or an "AccessDenied" if the the user is not provided with the given access right(s). Other errors also might be possible, for example the request might be mallformad of the object might be missing.

### Request


```
{
    "operation": "checkAccess",
    "path": "/pipas",
    "privileges": "rw-",
    "request_created": "2017-12-06T12:08:53.704Z",
    "request_id": 3
}
```


### Response

Here is the reply showing the user has the given rights.

**Response example**


```
{
    "request_created": "2017-12-06T12:08:53.704Z",
    "request_id": 3,
    "request_processed": "2017-12-06T12:08:53.717Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### chown

!!! post "POST /v2/imperative" Changes the ownership of object(s).

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| owner_gr | string | Required | The group name of the new group owner. One of the owner_user_name and owner_group_name is mandatory. |
| owner_us | string | Required | The username of the new owner. One of the owner_user_name and owner_group_name is mandatory. |
| path | string | Required | The full path of the object to change. |
| recursive | boole | Optional | If this propertyis true all the subitems of the tree will be changed too. If any of the changes fail with access denied no change will be done on any of the tree items. |

```
{
    "operation": "chown",
    "owner_group_name": "users",
    "owner_user_name": "pipas",
    "path": "/core2/containers",
    "recursive": true,
    "request_created": "2017-10-05T07:51:05.133Z",
    "request_id": 3
}
```

### Response

The reply to a successful chown request looks like this.

**Response example**

```
{
    "acl": "user::rwx,group::rwx,other::rwx",
    "object_name": "containers",
    "object_path": "/core2/",
    "owner_group_name": "users",
    "owner_user_name": "pipas",
    "request_created": "2017-10-05T07:51:05.133Z",
    "request_id": 3,
    "request_processed": "2017-10-05T07:51:05.208Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### mkdir

!!! post "POST /v2/imperative" This call is for creating folders in the CMON Directory Tree.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `path` | string | Required | The full path of the folder to create. |

```
{
    "operation": "mkdir",
    "path": "/foldername",
    "request_created": "2017-10-09T07:42:17.091Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
    "request_created": "2017-10-09T07:42:17.091Z",
    "request_id": 3,
    "request_processed": "2017-10-09T07:42:17.143Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### rmdir

!!! post "POST /v2/imperative" This call is for removing empty folders. This call will be returning with an error if the object at the given path is not an empty folder. To remove objects other than a folder please consider using the `delete` call. The `delete` call is also able to remove empty folders.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `path` | string | Required | The full path of the folder to remove. |

```
{
    "operation": "rmdir",
    "path": "/foldername",
    "request_created": "2017-10-09T07:13:54.410Z",
    "request_id": 3
}
```

### Response

**Response example**


```
{
    "request_created": "2017-10-09T07:13:54.410Z",
    "request_id": 3,
    "request_processed": "2017-10-09T07:13:54.470Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### mkfile

!!! post "POST /v2/imperative" This call can be used to create a file type CDT entry.

### Request


```
{
    "operation": "mkfile",
    "path": "/tmp/filename.txt",
    "request_created": "2019-01-04T09:05:27.349Z",
    "request_id": 17
}
```


### Response

Here is a reply showing a successful file operation.

**Response example**


```
{
    "request_created": "2019-01-04T09:05:27.349Z",
    "request_id": 17,
    "request_processed": "2019-01-04T09:05:27.364Z",
    "request_status": "Ok",
    "request_user_id": 1
}
```


### setContent

!!! post "POST /v2/imperative" This call can be used to change the content string of a CDT entry. Currently only the file type CDT entries can have contents handled in this call.

### Request


```
{
    "content": "This is something.\n\n",
    "operation": "setcontent",
    "path": "/tmp/ak.txt",
    "request_created": "2019-01-04T09:03:34.657Z",
    "request_id": 13
}
```


### Response

Here is a reply reporting a successful operation.

**Response example**


```
{
    "request_created": "2019-01-04T09:03:34.657Z",
    "request_id": 13,
    "request_processed": "2019-01-04T09:03:34.670Z",
    "request_status": "Ok",
    "request_user_id": 1
}
```


### delete

!!! post "POST /v2/imperative" This call is for deleting object from the tree. Once an object is deleted, it is deleted for good. If for example one performs a delete operation on a container it is deleted from the server. It is gone, it is nonexisting, it is a late container.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `path` | string | Required | The full path of the object to delete. |


```
{
    "operation": "delete",
    "path": "/host04/containers/temporary",
    "request_created": "2017-10-16T13:58:53.186Z",
    "request_id": 3
}
```


### Response

**Response example**


```
{
    "request_created": "2017-10-16T13:58:53.186Z",
```

```
    "request_id": 3,
    "request_processed": "2017-10-16T13:58:56.179Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```

### 4.12.24 User Database manipulation

The `/v2/users` path is to manipulate Cmon Users, users that are maintained by the Cmon controller. These users are used while setting up encrypted communication for the RPC v2 interface, so without this interface it is not possible to use the RPC v2 itself.

| Operation | Description |
|---|---|
| *createUser* | Create a new Cmon user account. |
| *whoAmI* | Return information about the currently authenticated user. |
| *getUsers* | Read the user database managed by the controller. |
| *setUser* | Modify certain properties of an existing Cmon user. |
| *disable* | Set the disabled flag for a user to prevent login. |
| *deleteUser* | Permanently delete an existing Cmon user. |
| *enable* | Clear the disabled flag so a user can log in again. |
| *changePassword* | Change the password of an existing user. |
| *getKeys* | Return the public keys registered for a user. |
| *addKey* | Add a new public key for a user. |
| *deleteKey* | Delete a public key by its name. |

#### createUser

!!! post “POST /v2/users” The createUser RPC call can be used to create a new Cmon User.

In the call both the user’s password and public kay can be specified so that the user can log in after the account is created.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `create_g` | boole | Optional | If the groups of the user does not exist the controller will try to create the group on the fly. If the group could be created the user will be added to the group, if not the request will be failed. |
| `new_p` | string | Optional | The plain text password that the user can actually use to log in. This is not mandatory, if the new password is not provided no password will be set and the user won't be able to log in with a password until a valid password is set. Please note that the `new_password` is not a property of the cmonuser. The plain password will not be stored, only the encoded version of the password is a property of the CmonUser object, but even that is a hidden property, not seen from outside of the controller code. |
| `user` | object | Required | The user object to create. |
| `user.class` | string | Required | Must be `"CmonUser"`. |
| `user.group` | array | Optional | The user groups the user belongs to. This is not a mandatory argument, if the `create_group` is set to true, but no group is provided the controller will try to create a group that has the same name as the user and add the new user to this new group. It is important to note that every user must belong to at least one group the so called "primary group". If the user belongs to multiple groups the first group is the primary group. Theoretically this call can accept multiple groups, but this feature is not yet implemented. |
| `user.publi` | array | Optional | Zero or more keys the user can use to log in. This is not a mandatory argument, if no key is provided the user will not able to log in with a key, but might still be able to use the system with a password. The `public_keys` is actually a property of the cmonuser, but it is a hidden property, not readable only using special calls like the getKeys call. |
| `user.publi key` | string | Optional | The actual public key string. |
| `user.publi name` | string | Optional | The name of the key. The name is just a human readable string the user provided as a reminder. |
| `user.user_` | string | Required | The username for the new user. |

```
{
    "create_group": true,
    "new_password": "p",
    "operation": "createUser",
    "user":
    {
        "class_name": "CmonUser",
        "groups": [
        {
```

(continues on next page)

(continued from previous page)

```
            "class_name": "CmonGroup",
            "group_name": "testgroup"
        } ],
        "public_keys": [
        {
            "key": "-----BEGIN RSA PUBLIC␣
↪KEY-----\nMIIBCgKCAQEAxupGXCUCtoytVYBD1NKeAB+oWauxJzGYvunMGnVIqhjF/o8gokNf\nV0mcad022SJ
↪RSA PUBLIC KEY-----\n",
            "name": "No Name"
        } ],
        "user_name": "pipas"
    }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| user | object | All the details of the user that has been created. |
| user.user_id | integer | The user ID of the user that just has been created. The only inmutable property of the user that can not be ever changed. |

Please note that there is no password or public key in the reply. We handle those as a secret.

Here is an example showing a stored user with the internal properties that are not visible from the outside (but might be set like the "password_encrypted"), for example:

```
{
    "class_name": "CmonUser",
    "email_address": "laszlo@severalnines.com",
    "first_name": "Laszlo",
    "groups": [
    {
        "class_name": "CmonGroup",
        "group_id": 4,
        "group_name": "testgroup"
    } ],
    "last_login": "2017-09-06T08:45:55.588Z",
    "last_name": "Pere",
    "password_encrypted":
↪"3babe525f514e13a421b1e2870824d8d3e08531c5db461b088aa8c4298a1eca8",
    "password_format": "sha256",
    "password_salt": "7cc254f8-1be8-e78d-765a-2e63339fc99a",
    "public_keys": [
    {
        "key": "-----BEGIN RSA PUBLIC␣
↪KEY-----\nMIIBCgKCAQEAxupGXCUCtoytVYBD1NKeAB+oWauxJzGYvunMGnVIqhjF/o8gokNf\nV0mcad022SJ
↪RSA PUBLIC KEY-----\n",
```

(continues on next page)

(continued from previous page)

```
        "name": "No Name"
    } ],
    "user_id": 3,
    "user_name": "pipas"
}
```

It is also important to know what the backend checks to see if the authenticated user actually has the proper privileges to create the new user in the given situation.

1. The controller checks if the `/.runtime/user_manager` entry exists and the authenticated user has execute access on it. If not the request will be rejected.
2. The backend checks if a user with the same name already exists. If so the creation of the user with the same name will be denied.
3. The controller checks if the passed new user has a group set. If so and the group does not exist the controller checks if the request asks for a new group to be created. If so the controller tries to create a new group.
4. Then the controller checks if the authenticated user has write access on the group. If the group was just created the authenticated user will be able to write it, but if it is pre-existing the authenticated user might not have write access to it. If this happens the request will be rejected.
5. Then the controller checks if there is already an object (not a cmon user) at the same path and name. If so the request will be rejected.
6. Then the new user is saved and an OK reply is sent.

**Response example**

```
{
    "request_created": "2017-07-04T12:49:50.896Z",
    "request_id": 2,
    "request_processed": "2017-07-04T12:49:50.954Z",
    "request_status": "Ok",
    "request_user_id": 1,
    "user":
    {
        "class_name": "CmonUser",
        "email_address": "rpc@email.com",
        "groups": [
        {
            "class_name": "CmonGroup",
            "group_id": 6,
            "group_name": "testgroup"
        } ],
        "user_id": 11,
        "user_name": "rpc_user"
    }
}
```

### whoAmI

!!! post "POST /v2/users" This call returns information about the authenticated user. The client application can send this call periodically to update the information about the user and the privileges the user has.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| with_extended_pr | boolea | Optional | If the request has this field set to true the reply will contain some extra information about the user's rigts. |

```
{
    "operation": "whoAmI",
    "request_created": "2018-12-07T08:51:14.324Z",
    "request_id": 3,
    "with_extended_privileges": true
}
```

### Response

| Field | Type | Description |
|---|---|---|
| user | object | The user object with all details about the authenticated user. |
| user_extended_pr | object | Convenience check that returns information about the effective privileges of several CDT entries. These privileges are not the property of the user but are controlled by the access rights of other CDT entries on the controller. |
| user_extended_pr can_create_clust | boole | Whether the user can create clusters. |
| user_extended_pr can_execute_jobs | boole | Whether the user can execute jobs. |

The list of the extended privileges are under construction, the caller may expect more than shown in this example.

**Response example**

```
{
    "request_created": "2018-12-07T08:51:14.324Z",
    "request_id": 3,
    "request_processed": "2018-12-07T08:51:14.332Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "user":
    {
        "acl": "user::rwx,group::r--,other::r--",
        "cdt_path": "/",
```

```
        "class_name": "CmonUser",
        "created": "2018-12-07T07:22:49.165Z",
        "disabled": false,
        "email_address": "laszlo@severalnines.com",
        "first_name": "Laszlo",
        "groups": [
        {
            "acl": "user::rwx,group::rw-,other::---",
            "cdt_path": "/groups",
            "class_name": "CmonGroup",
            "created": "2018-12-07T07:22:49.153Z",
            "group_id": 4,
            "group_name": "testgroup",
            "owner_group_id": 1,
            "owner_group_name": "admins",
            "owner_user_id": 1,
            "owner_user_name": "system"
        } ],
        "last_failed_login": "",
        "last_login": "2018-12-07T08:51:14.317Z",
        "last_name": "Pere",
        "n_failed_logins": 0,
        "owner_group_id": 1,
        "owner_group_name": "admins",
        "owner_user_id": 4,
        "owner_user_name": "pipas",
        "suspended": false,
        "timezone":
        {
            "class_name": "CmonTimeZone",
            "name": "CET",
            "offset": -3600,
            "has_dst_rules": true
        },
        "user_id": 4,
        "user_name": "pipas"
    },
    "user_extended_privileges":
    {
        "can_create_clusters": true,
        "can_execute_jobs": true,
        "class_name": "CmonExtendedPrivileges"
    }
}
```

### getUsers

!!! post "POST /v2/users" A call to read the user database.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `with_tag` | array | Optional | The request may contain this field to limit the list of the returned users to those that have at least one of the tags passed here. |
| `without_tags` | array | Optional | The request may contain this field to limit the list of the returned users to those that has none of the tags passed here. |

```
{
    "operation": "getUsers",
    "request_created": "2020-05-13T06:16:11.197Z",
    "request_id": 3,
    "with_tags": [ "testUser" ]
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | The total number of users managed by the controller. |
| `users` | array | A list of CmonUser class objects. To get more information about the user objects please check the cmonuser and cmongroup sections. |

**Response example**

```
{
    "request_created": "2020-05-13T06:16:11.197Z",
    "request_id": 3,
    "request_processed": "2020-05-13T06:16:11.216Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "total": 16,
    "debug_messages":
    [
        "RPC V2 authenticated user is 'pipas'."
    ],
    "users":
    [
        {
            "class_name": "CmonUser",
            "cdt_path": "/",
```

```
            "owner_user_id": 4,
            "owner_user_name": "pipas",
            "owner_group_id": 1,
            "owner_group_name": "admins",
            "acl": "user::rwx,group::r--,other::r--",
            "tags":
            [
                "testUser"
            ],
            "created": "2020-05-13T06:08:24.564Z",
            "disabled": false,
            "email_address": "laszlo@severalnines.com",
            "first_name": "Laszlo",
            "last_failed_login": "",
            "last_login": "2020-05-13T06:16:11.186Z",
            "last_name": "Pere",
            "n_failed_logins": 0,
            "origin": "CmonDb",
            "suspended": false,
            "user_id": 4,
            "user_name": "pipas",
            "groups":
            [
                {
                    "class_name": "CmonGroup",
                    "cdt_path": "/groups",
                    "owner_user_id": 1,
                    "owner_user_name": "system",
                    "owner_group_id": 1,
                    "owner_group_name": "admins",
                    "acl": "user::rwx,group::rwx,other::---",
                    "created": "2020-05-13T06:08:13.521Z",
                    "group_id": 1,
                    "group_name": "admins"
                }
            ],
            "timezone":
            {
                "class_name": "CmonTimeZone",
                "name": "Central European Time",
                "abbreviation": "CET",
                "offset": -3600,
                "has_dst_rules": true
            }
        }
    ]
}
```

### setUser

!!! post "POST /v2/users" The setUser call can be used to modify certain properties of existing Cmon users.

Here is an example showing how to modify the email address of a user.

The reply to this request will contain the request status and, if an error occurred, the error string. If the request was successfully processed the controller will send back the user object with all of its properties after the change.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| `user` | object | Require | Identifies the user and the properties to change. The "class_name" of this map should be "CmonUser" and the "user_name" should identify an existing user. Any other fields are properties the caller wants to change. Please check out the cmonuser section to see what properties can be changed. |

```
{
    "operation": "setUser",
    "request_created": "2017-06-26T07:25:16.198Z",
    "request_id": 3,
    "user":
    {
        "class_name": "CmonUser",
        "email_address": "system@mydomain.com",
        "user_name": "system"
    }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `user` | object | Shows the properties of the user after the change. |

**Response example**

```
{
    "request_created": "2017-06-26T07:37:51.522Z",
    "request_id": 3,
    "request_processed": "2017-06-26T07:37:51.575Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "user":
```

```
    {
        "class_name": "CmonUser",
        "email_address": "system@mydomain.com",
        "first_name": "System",
        "groups": [
        {
            "class_name": "CmonGroup",
            "group_id": 1,
            "group_name": "admins"
        } ],
        "last_name": "User",
        "user_id": 1,
        "user_name": "system"
    }
}
```

### disable

!!! post "POST /v2/users" This call will set the "disabled" flag for the user and so prevent the user from logging in.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `user` | object | Required | Identifies the user to disable. Must include `class_name` of `"CmonUser"` and `user_name`. |

```
{
    "operation": "disable",
    "request_created": "2017-10-30T08:25:15.685Z",
    "request_id": 3,
    "user":
    {
        "class_name": "CmonUser",
        "user_name": "nobody"
    }
}
```

### Response

The reply contains the user object with the "disabled" flag set to true.

**Response example**

```
{
    "request_created": "2017-10-30T08:25:15.685Z",
    "request_id": 3,
```

```
    "request_processed": "2017-10-30T08:25:16.535Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "user":
    {
        "cdt_path": "/",
        "class_name": "CmonUser",
        "created": "2017-10-27T12:34:47.333Z",
        "disabled": true,
        "first_name": "Default",
        "groups": [
        {
            "cdt_path": "/groups",
            "class_name": "CmonGroup",
            "group_id": 3,
            "group_name": "nobody"
        } ],
        "last_failed_login": "",
        "last_name": "User",
        "n_failed_logins": 0,
        "owner_group_id": 1,
        "owner_group_name": "admins",
        "owner_user_id": 2,
        "owner_user_name": "nobody",
        "suspended": false,
        "user_id": 2,
        "user_name": "nobody"
    }
}
```


### deleteUser

!!! post "POST /v2/users" Permanently deletes an existing CMON user. The user to delete is identified by `user_name` inside the `user` object. All associated SSH keys, group memberships, and preferences for that user are also removed.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `user` | object | Required | Identifies the user to delete. Must include `class_name` of `"CmonUser"` and `user_name`. |


```
{
    "operation": "deleteUser",
    "request_created": "2018-11-26T14:19:29.891Z",
    "request_id": 3,
    "user":
```


(continues on next page)

(continued from previous page)

```
    {
        "class_name": "CmonUser",
        "user_name": "laszlo"
    }
}
```

### enable

!!! post "POST /v2/users" This call will clear the "disabled" flag for the user so the user will be able to log in. The "suspended" flag will also be cleared, the failed login counter set to 0 and the date&time of the last failed login gets deleted, so users who are suspended for failed login attempts will also be able to log in.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `user` | object | Required | Identifies the user to enable. Must include `class_name` of `"CmonUser"` and `user_name`. |

```
{
    "operation": "enable",
    "request_created": "2017-10-30T08:28:29.852Z",
    "request_id": 3,
    "user":
    {
        "class_name": "CmonUser",
        "user_name": "nobody"
    }
}
```

### Response

**Response example**

```
{
    "request_created": "2017-10-30T08:28:29.852Z",
    "request_id": 3,
    "request_processed": "2017-10-30T08:28:29.912Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "user":
    {
        "cdt_path": "/",
        "class_name": "CmonUser",
        "created": "2017-10-27T12:34:47.333Z",
        "disabled": false,
        "first_name": "Default",
```

```
        "groups": [
        {
            "cdt_path": "/groups",
            "class_name": "CmonGroup",
            "group_id": 3,
            "group_name": "nobody"
        } ],
        "last_failed_login": "",
        "last_name": "User",
        "n_failed_logins": 0,
        "owner_group_id": 1,
        "owner_group_name": "admins",
        "owner_user_id": 2,
        "owner_user_name": "nobody",
        "suspended": false,
        "user_id": 2,
        "user_name": "nobody"
    }
}
```


### changePassword

!!! post "POST /v2/users" This call is of course for changing the password of an existing user.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| new_ | string | Require | The new password in plain string format. |
| old_ | string | Optional | The current password in plain format. The user's old password that will be checked using the usually password authentication algorithms. Please note that a superuser can change the passwords for any user without providing the old password, but if the old password is sent it will be checked even if the user that sent it is a superuser. |
| user | object | Require | Identifies the user that should have the new password. |


```
{
    "new_password": "newpass",
    "old_password": "secret",
    "operation": "changePassword",
    "request_created": "2017-06-29T07:56:17.710Z",
    "request_id": 2,
    "user":
    {
        "class_name": "CmonUser",
```


(continues on next page)

(continued from previous page)

```
        "user_name": "system"
    }
}
```

### Response

**Response example**

```
{
    "request_created": "2017-06-29T07:56:17.710Z",
    "request_id": 2,
    "request_processed": "2017-06-29T07:56:17.761Z",
    "request_status": "Ok",
    "request_user_id": 1
}
```

## getKeys

!!! post "POST /v2/users" A call to return the public keys of a user.

### Request

| Parameter | Type | Require | Description |
|---|---|---|---|
| `user` | object | Require | This identifies the user whose public keys will be returned. Only the class name and the user name needed here. Please note that the controller handles the public keys securely, normal users can not read the public keys of other users. Users can always read their own keys of course. |

```
{
    "operation": "getKeys",
    "request_created": "2017-07-05T09:23:04.988Z",
    "request_id": 2,
    "user":
    {
        "class_name": "CmonUser",
        "user_name": "pipas"
    }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| public_keys | array | Contains the public keys of the user. |
| public_keys[].key | string | The actual key string. |
| public_keys[].name | string | The human readable name of the key, a reminder that the user set when adding the key. |
| total | integer | Shows how many keys has been found. |
| user | object | The user that has these keys. This field is sent back for convenience. |

**Response example**


```
{
    "public_keys": [
    {
        "key": "-----BEGIN RSA PUBLIC␣
↪KEY-----\nMIIBCgKCAQEAxupGXCUCtoytVYBD1NKeAB+oWauxJzGYvunMGnVIqhjF/o8gokNf\nV0mcad022S
↪RSA PUBLIC KEY-----\n",
        "name": "Unnamed key"
    } ],
    "request_created": "2017-07-05T09:23:04.988Z",
    "request_id": 2,
    "request_processed": "2017-07-05T09:23:05.034Z",
    "request_status": "Ok",
    "request_user_id": 1,
    "total": 1,
    "user":
    {
        "class_name": "CmonUser",
        "email_address": "laszlo@severalnines.com",
        "first_name": "Laszlo",
        "groups": [
        {
            "class_name": "CmonGroup",
            "group_id": 4,
            "group_name": "testgroup"
        } ],
        "last_login": "2017-07-05T09:22:55.587Z",
        "last_name": "Pere",
        "title": "",
        "user_id": 3,
        "user_name": "pipas"
    }
}
```

### addKey

!!! post "POST /v2/users" A call to add/register a new public key for a user.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `public_key` | object | Required | The public key to add. |
| `user` | object | Required | The user to whom the key will be added. Only the class name and the user name has to be specified here. |

```
{
    "operation": "addKey",
    "public_key":
    {
        "key": "-----BEGIN RSA PUBLIC␣
↪KEY-----\nMIIBCgKCAQEAr6KS5KsnwVuoOzFGJsF2oPhUokBEEdrcGfVPX5XbdlIn0E/Gn44i\nx4HBt0rDKNe
↪RSA PUBLIC KEY-----\n",
        "name": ""
    },
    "request_created": "2017-07-05T11:55:57.885Z",
    "request_id": 3,
    "user":
    {
        "class_name": "CmonUser",
        "user_name": "pipas"
    }
}
```

### Response

**Response example**


```
{
    "request_created": "2017-07-05T11:55:57.885Z",
    "request_id": 3,
    "request_processed": "2017-07-05T11:55:57.939Z",
    "request_status": "Ok",
    "request_user_id": 3
}
```


### deleteKey

!!! post "POST /v2/users" This call can be used to delete a public key by its name. The call looks very similar to the "addKey" call.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `public_key` | object | Required | Identifies the key to delete by its name. |
| `public_key.name` | string | Required | The name of the key to delete. |
| `user` | object | Required | The user from whom the key will be deleted. Only the class name and the user name has to be specified here. |


```
{
    "operation": "deleteKey",
    "public_key":
    {
        "name": "The name"
    },
    "request_created": "2017-07-05T12:35:43.135Z",
    "request_id": 3,
    "user":
    {
        "class_name": "CmonUser",
        "user_name": "pipas"
    }
}
```


### createGroup

!!! post "POST /v2/users" Creates a new CMON user group. Groups are used to organise users and manage access control collectively. The group is identified by `group_name` inside the `group` object.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `group` | object | Required | The group object to create. |
| `group.class_name` | string | Required | Must be `CmonGroup`. |
| `group.group_name` | string | Required | The name of the group to create. |

```
{
    "operation": "createGroup",
    "group":
    {
        "class_name": "CmonGroup",
        "group_name": "GroupName"
    }
}
```

### deleteGroup

!!! post "POST /v2/users" This call is for deleting an existing Cmon Group. Groups can only be deleted if they are empty (no Cmon User is assigned to them) and the authenticateduser has a write access to the actual group itself.

Currently only the group name is used to identify the group that will be deleted.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `group` | object | Required | The group to delete. |
| `group.class_name` | string | Required | Must be `CmonGroup`. |
| `group.group_name` | string | Required | The name of the group to delete. |

```
{
    "group":
    {
        "class_name": "CmonGroup",
        "group_name": "pip"
    },
    "operation": "deleteGroup",
    "request_created": "2019-08-05T07:44:39.972Z",
    "request_id": 3
}
```

### Response

**Response example**

```
{
    "request_created": "2019-08-05T07:44:39.972Z",
    "request_id": 3,
    "request_processed": "2019-08-05T07:44:39.995Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```

### getGroups

!!! post "POST /v2/users" A call to read the user database. In its basic form the RPC call looks like this:

### Request

```
{
    "operation": "getGroups",
    "request_created": "2017-06-26T06:42:12.902Z",
    "request_id": 3
}
```

### Response

| Field | Type | Description |
|---|---|---|
| groups | array | The list of user groups the controller has. |
| total | integer | The total number of groups the controller has. |

**Response example**

```
{
    "groups": [
    {
        "class_name": "CmonGroup",
        "group_id": 1,
        "group_name": "admins"
    },
    {
        "class_name": "CmonGroup",
        "group_id": 2,
        "group_name": "users"
    },

    . . .

    {
        "class_name": "CmonGroup",
        "group_id": 6,
        "group_name": "rpc_group"
```

```
    } ],
    "reply_received": "2017-07-24T08:53:33.498Z",
    "request_created": "2017-07-24T08:53:33.493Z",
    "request_id": 3,
    "request_processed": "2017-07-24T08:53:33.538Z",
    "request_status": "Ok",
    "request_user_id": 3,
    "total": 6
}
```

### addToGroup

!!! post "POST /v2/users" Adds the user to a new group or moves the user from the current primary group to a new group.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `group_name` | string | Required | The name of the group to add the user to. |
| `replace_primary_gro` | boolean | Required | If true, the user is moved from their current primary group to the new group. |
| `user` | object | Required | The user to add to the group. |
| `user.class_name` | string | Required | Must be `CmonUser`. |
| `user.user_name` | string | Required | The name of the user to add. |

```
{
    "group_name": "admins",
    "operation": "addToGroup",
    "replace_primary_group": true,
    "request_created": "2018-01-12T11:07:48.963Z",
    "request_id": 2,
    "user":
    {
        "class_name": "CmonUser",
        "user_name": "pipas"
    }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `user` | object | All the details of the user that was updated. |

**Response example**


```
{
    "request_created": "2018-01-12T11:07:48.963Z",
    "request_id": 2,
    "request_processed": "2018-01-12T11:07:48.984Z",
    "request_status": "Ok",
    "request_user_id": 1,
    "user":
    {
        "cdt_path": "/",
        "class_name": "CmonUser",
        "created": "2018-01-12T07:26:13.677Z",
        "disabled": false,
        "email_address": "laszlo@severalnines.com",
        "first_name": "Laszlo",
        "groups": [
        {
            "cdt_path": "/groups",
            "class_name": "CmonGroup",
            "group_id": 1,
            "group_name": "admins"
        } ],
        "last_failed_login": "",
        "last_login": "2018-01-12T10:54:53.222Z",
        "last_name": "Pere",
        "n_failed_logins": 0,
        "owner_group_id": 1,
        "owner_group_name": "admins",
        "owner_user_id": 3,
        "owner_user_name": "pipas",
        "suspended": false,
        "user_id": 3,
        "user_name": "pipas"
    }
}
```


### removeFromGroup

!!! post “POST /v2/users” Removes the user from a group if it is indeed the member of that group. Here is how the rquest should look like:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `group_name` | string | Required | The name of the group to remove the user from. |
| `user` | object | Required | The user to remove from the group. |
| `user.class_name` | string | Required | Must be `CmonUser`. |
| `user.user_name` | string | Required | The name of the user to remove. |


```
{
    "group_name": "admins",
    "operation": "removeFromGroup",
    "request_created": "2018-10-08T06:03:36.983Z",
    "request_id": 3,
    "user":
    {
        "class_name": "CmonUser",
        "user_name": "sisko"
    }
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `user` | object | All the details of the user after the group removal. |

**Response example**


```
{
    "request_created": "2018-10-08T06:03:36.983Z",
    "request_id": 3,
    "request_processed": "2018-10-08T06:03:37.006Z",
    "request_status": "Ok",
    "request_user_id": 4,
    "user":
    {
        "acl": "user::rwx,user:pipas:rwx,group::r--,other::r--",
        "cdt_path": "/",
        "class_name": "CmonUser",
        "created": "2018-10-05T14:09:14.660Z",
        "disabled": false,
        "email_address": "sisko@ds9.com",
        "first_name": "Benjamin",
        "groups": [
        {
            "cdt_path": "/groups",
            "class_name": "CmonGroup",
            "group_id": 5,
```

```
            "group_name": "ds9",
            "owner_group_id": 1,
            "owner_group_name": "admins",
            "owner_user_id": 1,
            "owner_user_name": "system"
        } ],
        "last_failed_login": "",
        "last_login": "2018-10-05T14:09:18.111Z",
        "last_name": "Sisko",
        "n_failed_logins": 0,
        "owner_group_id": 1,
        "owner_group_name": "admins",
        "owner_user_id": 5,
        "owner_user_name": "sisko",
        "suspended": false,
        "title": "Captain",
        "user_id": 5,
        "user_name": "sisko"
    }
}
```


### canCreateUser

!!! post "POST /v2/users" This call can be used to find out if the currently authenticated user is allowed to create a new user. For the user creation the authenticated user must be the superuser or has write access to a user group and a folder.

Please note that the request will be successfull (the request_status will be "Ok") even if the user can not possibly create a new cmon user.

### Request


```
{
    "operation": "canCreateUser",
    "request_created": "2017-10-19T08:02:53.276Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| can_create_user | boolea | True if the authenticated user has a chance to create a new user. |
| folders_available_for_ | array | The folders that are writable and so can hold a new user. The full path of the folders are enumerated in a list. |
| groups_available_f | array | The names of the groups that are writable for the user and so can hold the new user. |

**Response example**


```
{
    "can_create_user": true,
    "folders_available_for_write": [ "/", "/core1/containers" ],
    "groups_available_for_write": [ "admins", "users", "nobody" ],
    "request_created": "2017-10-19T08:02:53.276Z",
    "request_id": 3,
    "request_processed": "2017-10-19T08:02:56.341Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```


### canCreateGroup

!!! post "POST /v2/users" This call is to decide if the authenticated user is allowed to create user groups.

### Request


```
{
    "operation": "canCreateGroup",
    "request_created": "2017-10-19T09:49:43.632Z",
    "request_id": 3
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `can_create_group` | boolean | True if the user is allowed to create cmon user groups. |
| `folders_available_for_write` | array | The list of folders the user has write access. |

**Response example**


```
{
    "can_create_group": true,
    "folders_available_for_write": [ "/", "/core1/containers", "/
↪galera_001/databases" ],
    "request_created": "2017-10-19T09:49:43.632Z",
    "request_id": 3,
    "request_processed": "2017-10-19T09:49:46.692Z",
    "request_status": "Ok",
    "request_user_id": 4
}
```

### updateUser

!!! post "POST /v2/users" The updateUser RPC call can be used to update the current logged in user.

In the call you may define a new password as well in the `new_password` field.

In the call both the user's password and public kay can be specified so that the user can log in after the account is created.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `new_password` | string | Optional | A new password to set for the user. |
| `user` | object | Required | The user object with updated fields. |
| `user.class_name` | string | Required | Must be `CmonUser`. |
| `user.user_id` | integer | Required | The ID of the user to update. |
| `user.first_name` | string | Optional | The user's first name. |
| `user.last_name` | string | Optional | The user's last name. |
| `user.email_address` | string | Optional | The user's email address. |
| `user.user_name` | string | Optional | The user's new username. |

```
{
    "new_password": "p",
    "operation": "updateUser",
    "user":
    {
        "class_name": "CmonUser",
        "user_id": 1234,
        "first_name": "Firstname",
        "last_name": "Lastname",
        "email_address": "something@xy.com",
        "user_name": "newusername"
    }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `user` | object | All the details of the user that has been updated. |
| `user.user_id` | integer | The user ID of the user that just has been created. The only inmutable property of the user that can not be ever changed. |

Please note that there is no password or public key in the reply. We handle those as a secret.

**Response example**

```
{
    "request_created": "2017-07-04T12:49:50.896Z",
```

```
    "request_id": 2,
    "request_processed": "2017-07-04T12:49:50.954Z",
    "request_status": "Ok",
    "request_user_id": 1,
    "user":
    {
        "class_name": "CmonUser",
        "email_address": "rpc@email.com",
        "groups": [
        {
            "class_name": "CmonGroup",
            "group_id": 6,
            "group_name": "testgroup"
        } ],
        "user_id": 11,
        "user_name": "rpc_user"
    }
}
```


### setUserPreferences

!!! post "POST /v2/users" A call to set the user's preferences.

Possible RPC request command:

To add new preferences:

```
$ s9s user --cmon-user=s9s --password=***** --preferences --
↪preferences-to-set="key1=value1;key2=value2;key3=value3"
```

To update existing preferences:

```
$ s9s user --cmon-user=s9s --password=***** --preferences --
↪preferences-to-set="key2=QWERTY;key3=UIOP"
```

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `user` | object | Required | The ClusterControl user's data in the request to set his/her properties. |
| `user.user_name` | string | Required | Must be a name of a valid logged in user. |
| `user.preferences` | object | Required | The list of one or more user's preferences to store to the DB. |


```
{
  "operation": "setUserPreferences",
  "request_created": "2023-01-25T14:57:37.155Z",
```


(continues on next page)

(continued from previous page)

```
  "request_id": 2,
  "user": {
    "class_name": "CmonUser",
    "user_name": "s9s",
    "preferences": {
      "param7": "val7"
    }
  }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `user` | object | The ClusterControl user's data, including his/her actual properties. |
| `user.preferences` | object | The list of all user's preferences stored in the databaseafter the request is completed. |

**Response example**

```
{
  "controller_id": "b195b016-ed96-49e4-a646-596b52d9d8f3",
  "reply_received": "2023-01-25T14:57:37.168Z",
  "request_created": "2023-01-25T14:57:37.155Z",
  "request_id": 2,
  "request_processed": "2023-01-25T14:57:37.167Z",
  "request_status": "Ok",
  "request_user_id": 3,
  "debug_messages": [ "RPC V2 authenticated user is 'admin'."  ],
  "user": {
    "class_name": "CmonUser",
    "user_id": 3,
    "user_name": "s9s",
    "preferences": {
      "param11": "val11",
      "param12": "val12",
      "param7": "val7"
    }
  }
}
```

### getUserPreferences

!!! post "POST /v2/users" A call to get the user's preferences from DB.

Possible RPC request command with s9s tools:

```
$ s9s user --cmon-user=s9s --password=***** --get-preferences
```

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| user | object | Required | The ClusterControl user's data in the request to get his/her properties. |
| user.user_name | string | Required | Must be a name of a valid logged in user. |

```
{
  "operation": "getUserPreferences",
  "request_created": "2023-01-25T13:55:07.068Z",
  "request_id": 2,
  "user": {
    "class_name": "CmonUser",
    "user_name": "s9s"
  }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| user | object | The ClusterControl user's data, including his/her actual properties. |
| user.preferences | object | The list of all user's preferences stored in the databaseafter the request is completed. |

**Response example**

```
{
  "controller_id": "b195b016-ed96-49e4-a646-596b52d9d8f3",
  "reply_received": "2023-01-25T13:55:07.073Z",
  "request_created": "2023-01-25T13:55:07.068Z",
  "request_id": 2,
  "request_processed": "2023-01-25T13:55:07.072Z",
  "request_status": "Ok",
  "request_user_id": 3,
  "debug_messages": [ "RPC V2 authenticated user is 'admin'."  ],
  "user": {
    "class_name": "CmonUser",
    "user_id": 3,
    "user_name": "s9s",
    "preferences": {
      "pref01": "value01",
```

```
      "pref02": "value02",
      "pref03": "value03"
    }
  }
}
```

### deleteUserPreferences

!!! post "POST /v2/users" A call to read the user's preferences.

Possible RPC request command:

```
$ s9s user --cmon-user=s9s --password=***** --preferences --
↪preferences-to-delete="key1;key2"
```

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| user | object | Required | The ClusterControl user's data in the request to set his/her properties. |
| user.user_name | string | Required | Must be a name of a valid logged in user. |
| user.preferences | object | Required | The list of one or more user's preferences to delete from the DB. |

```
{
  "operation": "deleteUserPreferences",
  "request_created": "2023-01-25T14:57:46.274Z",
  "request_id": 2,
  "user": {
    "class_name": "CmonUser",
    "user_name": "s9s",
    "preferences": {
      "param7": ""
    }
  }
}
```

### Response

| Field | Type | Description |
|---|---|---|
| user | object | The ClusterControl user's data, including his/her actual properties. |
| user.preferences | object | The list of all user's preferences from the databaseafter the request is completed. |

**Response example**


```
{
  "controller_id": "b195b016-ed96-49e4-a646-596b52d9d8f3",
  "reply_received": "2023-01-25T14:57:46.284Z",
  "request_created": "2023-01-25T14:57:46.274Z",
  "request_id": 2,
  "request_processed": "2023-01-25T14:57:46.283Z",
  "request_status": "Ok",
  "request_user_id": 3,
  "debug_messages": [ "RPC V2 authenticated user is 'admin'."  ],
  "user": {
    "class_name": "CmonUser",
    "user_id": 3,
    "user_name": "s9s",
    "preferences": {
      "param11": "val11",
      "param12": "val12"
    }
  }
}
```


### getDbVersions

!!! post "POST /v2/dbversions" Lists the available 1-2 or 3 digits database versions supported by ClusterControl for a given cluster type and vendor. Returned values can be used as values on "provider_version" field on the cluster creation jobs.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `cluster_type` | string | Required | The cluster type to query versions for |
| `vendor` | string | Required | The vendor to query versions for |
| `patch_number` | boolean | Optional | If true, returns 3-digit versions. Default is false |
| `use_vendor_api` | boolean | Optional | If true, queries the vendor's API for versions. Default is false |

**Step 1 - Get 1-2 digit versions**


```
{
  "cluster_type": "galera",
  "operation": "getDbVersions",
  "request_created": "2024-01-04T17:59:22.892Z",
  "request_id": 2,
  "vendor": "mariadb"
}
```


**Step 2 - Get 3-digit versions with patch_number**

```
{
  "cluster_type": "galera",
  "operation": "getDbVersions",
  "patch_number": true,
  "request_created": "2024-01-04T18:06:00.767Z",
  "request_id": 2,
  "vendor": "percona"
}
```


**Step 3 - Get 3-digit versions via vendor API**


```
{
  "cluster_type": "galera",
  "operation": "getDbVersions",
  "patch_number": true,
  "request_created": "2024-01-04T18:06:00.767Z",
  "request_id": 2,
  "use_vendor_api": true,
  "vendor": "percona"
}
```


### Response

| Field | Type | Description |
|---|---|---|
| `db_versions` | array | Possible values on `provider_version` field on deployment jobs |
| `cluster_type` | string | The cluster type queried |
| `vendor` | string | The vendor queried |
| `use_vendor_api` | boolean | Indicates whether the vendor's API was used as the source of versions |

The controller replies with supported 1-2 digit versions for the specified `cluster_type` and `vendor`:

**Response example**


```
{
  "authenticated": true,
  "cluster_type": "GALERA",
  "controller_id": "449bf7e9-a274-41db-bec6-85c915e619c5",
  "reply_received": "2024-01-04T17:59:22.895Z",
  "request_created": "2024-01-04T17:59:22.892Z",
  "request_id": 2,
  "request_processed": "2024-01-04T17:59:22.894Z",
  "request_status": "Ok",
  "request_user_id": 6,
  "rpc_version": "2.0",
  "vendor": "mariadb",
  "db_versions": [
    "10.4",
    "10.5",
    "10.6",
```

```
    "10.7",
    "10.8",
    "10.9",
    "10.10",
    "10.11"
  ],
}
```


It is also possible to specify the `patch_number` option on some clusters (for example: "galera"). The default value for this option is false. The controller then replies with supported 3-digit versions only for the specified `cluster_type` and `vendor`:

**Response example (patch_number=true)**


```
{
  "authenticated": true,
  "cluster_type": "GALERA",
  "controller_id": "449bf7e9-a274-41db-bec6-85c915e619c5",
  "reply_received": "2024-01-04T18:06:00.771Z",
  "request_created": "2024-01-04T18:06:00.767Z",
  "request_id": 2,
  "request_processed": "2024-01-04T18:06:00.771Z",
  "request_status": "Ok",
  "request_user_id": 6,
  "rpc_version": "2.0",
  "use_vendor_api": false,
  "vendor": "percona",
  "db_versions": [
    "8.0.21",
    "8.0.22",
    "8.0.23",
    "8.0.25",
    "8.0.26",
    "8.0.27",
    "8.0.28",
    "8.0.29",
    "8.0.30",
    "8.0.31",
    "8.0.32",
    "8.0.33",
    "8.0.34"
  ],
}
```


The source of the versions can be the stored values of ClusterControl or the vendor's API. The `use_vendor_api` option can be used to specify the source of the versions (default value is false). The controller then queries the vendor's API to reply with supported 3-digit versions only for the specified `cluster_type` and `vendor`:

**Response example (use_vendor_api=true)**


```
{
  "authenticated": true,
  "cluster_type": "GALERA",
  "controller_id": "449bf7e9-a274-41db-bec6-85c915e619c5",
  "reply_received": "2024-01-04T18:06:00.771Z",
  "request_created": "2024-01-04T18:06:00.767Z",
  "request_id": 2,
  "request_processed": "2024-01-04T18:06:00.771Z",
  "request_status": "Ok",
  "request_user_id": 6,
  "rpc_version": "2.0",
  "use_vendor_api": false,
  "vendor": "percona",
  "db_versions": [
    "8.0.21",
    "8.0.22",
    "8.0.23",
    "8.0.25",
    "8.0.26",
    "8.0.27",
    "8.0.28",
    "8.0.29",
    "8.0.30",
    "8.0.31",
    "8.0.32",
    "8.0.33",
    "8.0.34"
  ],
}
```


### getClusterTypes

!!! post "POST /v2/dbversions" Gets the list of supported clusters types. Returned values can be used as an argument to the `cluster_type` option (upper or lower case).

### Request


```
{
  "operation": "getClusterTypes",
  "request_created": "2024-01-04T17:43:14.776Z",
  "request_id": 2
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `cluster_types` | array | Possible cluster types supported |

The controller replies with supported cluster types:

**Response example**


```
{
  "authenticated": true,
  "controller_id": "449bf7e9-a274-41db-bec6-85c915e619c5",
  "reply_received": "2024-01-04T17:43:14.779Z",
  "request_created": "2024-01-04T17:43:14.776Z",
  "request_id": 2,
  "request_processed": "2024-01-04T17:43:14.779Z",
  "request_status": "Ok",
  "request_user_id": 6,
  "rpc_version": "2.0",
  "cluster_types": [
    "MYSQLCLUSTER",
    "REPLICATION",
    "MYSQL_SINGLE",
    "MONGODB",
    "POSTGRESQL_SINGLE",
    "GROUP_REPL",
    "REDIS",
    "MSSQL_SINGLE",
    "MSSQL_AO_ASYNC",
    "MSSQL_AO_SYNC",
    "ELASTIC",
    "GALERA"
  ],
}
```


### getVendors

!!! post “POST /v2/dbversions” Gets the list of supported vendors on all supported clusters. If no version is retrieved for a specific cluster type-vendor, it means that it is not supported.

### Request


```
{
  "operation": "getVendors",
  "request_created": "2024-01-04T17:49:46.012Z",
  "request_id": 2
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `vendors` | array | Possible vendor values on `vendor` field |

The controller replies with supported vendors on all cluster types:

**Response example**


```
{
  "authenticated": true,
  "controller_id": "449bf7e9-a274-41db-bec6-85c915e619c5",
  "reply_received": "2024-01-04T17:49:46.016Z",
  "request_created": "2024-01-04T17:49:46.012Z",
  "request_id": 2,
  "request_processed": "2024-01-04T17:49:46.015Z",
  "request_status": "Ok",
  "request_user_id": 6,
  "rpc_version": "2.0",
  "debug_messages": [ "RPC V2 authenticated user is 'ccalvaro'."  ],
  "vendors": [
    "percona",
    "mariadb",
    "oracle",
    "codership",
    "10gen",
    "mongodbenterprise",
    "microsoft",
    "elasticsearch",
    "redis",
    "postgresql",
    "enterprisedb"
  ]
}
```


## 4.12.25 Controllers manipulation on database

The `/v2/poolcontrollers` path is to retrieve controllers managed by the system.

| Operation | Description |
|---|---|
| *listPoolControllers* | Retrieve controllers from the DB, optionally filtered by ID or status. |
| *startController* | Start a controller in the pool remotely via SSH. |
| *stopController* | Stop a controller in the pool remotely via SSH. |
| *setPoolMode* | Change the pool mode state on the controller. |

### listPoolControllers

!!! post "POST /v2/poolcontrollers" The getControllers RPC call can be used to retrieve controllers from the DB. By default, only controllers that are not in 'stopped' status are returned.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `controller_id` | integer | Optional | If provided and non-zero, only the controller with this id is returned. |
| `all_controller` | boolean | Optional | If set to true, all controllers including stopped ones are returned. |

```
{
  "operation": "listcontrollers",
  "controller_id": 0,
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `controllers` | array | List of controller objects. |
| `total` | integer | Total number of controllers returned. |

**Response example**

```
{
  "controllers": [ {
    "controller_id": 1,
    "hostname": "controller1",
    "port": 3306,
    "status": "active",
    "properties": "full_controller",
    "report_ts": "2025-06-28 15:54:00",
    "clusters": [ 1, 2, 3 ]
  }, {
    "controller_id": 2,
    "hostname": "controller2",
    "port": 3306,
    "status": "stopped",
    "properties": "full_controller",
    "report_ts": "2025-06-28 15:54:00",
    "clusters": [ 4, 5, 6 ]
  } ],
  "total": 2
}
```

### startController

!!! post "POST /v2/poolcontrollers" Starts a controller in the pool remotely via SSH. This operation is only allowed when the controller is in pool mode and requires superuser privileges.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `controller_id` | integer | Required | The ID of the controller to start. |

```
{
  "operation": "startcontroller",
  "controller_id": 2
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `controller` | object | The controller object reflecting the updated state. |
| `message` | string | Status message for the start operation. |

**Response example**

```
{
  "controller": {
    "controller_id": 2,
    "hostname": "controller2",
    "port": 9500,
    "status": "active",
    "properties": "full_controller",
    "clusters": [ 4, 5 ]
  },
  "message": "Controller start command sent successfully."
}
```

### stopController

!!! post "POST /v2/poolcontrollers" Stops a controller in the pool remotely via SSH. This operation is only allowed when the controller is in pool mode and requires superuser privileges. Cannot be used to stop the local controller.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `controller_id` | integer | Required | The ID of the controller to stop. |

```
{
  "operation": "stopcontroller",
  "controller_id": 2
}
```

### Response

| Field | Type | Description |
|---|---|---|
| controller | object | The controller object reflecting the updated state. |
| message | string | Status message for the stop operation. |

**Response example**

```
{
  "controller": {
    "controller_id": 2,
    "hostname": "controller2",
    "port": 9500,
    "status": "stopped",
    "properties": "full_controller",
    "clusters": [ 4, 5 ]
  },
  "message": "Controller stop command sent successfully."
}
```

### setPoolMode

!!! post "POST /v2/poolcontrollers" Changes the pool mode state on the controller. When the state changes, the controller will restart automatically. This operation is only allowed on the main controller and requires superuser privileges.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| pool_mode | boolean | Optional | Whether to enable or disable pool mode. Defaults to true. |
| conf_storag | string | Optional | Storage type for pool configuration. Valid values: "nfs" or "k8s". Defaults to "nfs". |

```
{
  "operation": "setPoolMode",
  "pool_mode": true,
  "conf_storage": "nfs"
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `pool_mode` | boolean | The resulting pool mode state. |
| `conf_storage` | string | The resulting storage type for pool configuration. |
| `controller` | object | The controller object reflecting the updated state. |

**Response example**


```
{
  "pool_mode": true,
  "conf_storage": "nfs",
  "controller": {
    "controller_id": 1,
    "hostname": "controller1",
    "port": 9500,
    "status": "restarting",
    "properties": "main_controller",
    "clusters": [ 1, 2, 3 ]
  }
}
```


## 4.12.26 Watchlists manipulation on database

The `/v2/watchlists` path is to manipulate whatchlists to visualize clusters according to user configuration.

| Operation | Description |
|---|---|
| *createWatchlist* | Create a new watchlist. |
| *updateWatchlist* | Update an existing watchlist. |
| *listWatchlists* | List watchlists stored on the controller database. |
| *deleteWatchlist* | Delete an existing watchlist. |

### createWatchlist

!!! post "POST /v2/watchlists" The `createWatchlist` RPC call can be used to create a new Watchlist.

Here are the most important elements of the request:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| watchlist_name | string | Required | Name given by the user to identify watchlist. |
| topics | string | Required | Themes for the different information types to be visualized. |
| clusters | string | Required | List of clusters IDs to be visualized on watchlist. |
| page_by | string | Required | Pagination to be used on watchlist. Initially only 'cluster' or 'topic' are supported. |
| grid | string | Required | The grid layout to be used on watchlist. |
| owner_id | integer | Required | User ID of the owner (creator) of the watchlist. |

```
{
    "operation": "createWatchlist",
    "watchlist":
    {
          "watchlist_name": "my watch",
          "topics": "topic1, topic3",
          "clusters": "11, 13, 14",
          "page_by": "cluster",
          "grid": "2x2",
          "owner_id": 1
    }
}
```

### updateWatchlist

!!! post "POST /v2/watchlists" The `updateWatchlist` RPC call can be used to update an existing watchlist.

Here are the most important elements of the request:

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| watchlist_id | integer | Required | The internal ID of the watchlist once it has been persisted. |
| watchlist_name | string | Required | Name given by the user to identify watchlist. |
| topics | string | Required | Themes for the different information types to be visualized. |
| clusters | string | Required | List of clusters IDs to be visualized on watchlist. |
| page_by | string | Required | Pagination to be used on watchlist. Initially only 'cluster' or 'topic' are supported. |
| grid | string | Required | The grid layout to be used on watchlist. Initially only '2x2' \| '2x3' \| '3x2' \| '3x3' are allowed. |
| owner_id | integer | Required | User ID of the owner (creator) of the watchlist. |

```
{
    "operation": "updateWatchlist",
    "watchlist":
    {
          "watchlist_id" : 2,
          "watchlist_name": "my watch",
          "topics": "topic1, topic3",
          "clusters": "11, 13, 14",
          "page_by": "cluster",
          "grid": "2x2",
          "owner_id": 1
    }
}
```

### listWatchlists

!!! post "POST /v2/watchlists" A call to read the watchlists on controller database.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| watchlist_id | integer | Optional | Limit the list of returned watchlists to those that have this id. Use '0' to get all watchlists. |
| watchlist_name | string | Optional | Limit the list of returned watchlists to those that have this name. |

```
{
  "operation": "listWatchlists",
  "request_created": "2025-04-01T08:35:17.696Z",
  "request_id": 2,
  "watchlist_id": 0
}
```

### Response

| Field | Type | Description |
|---|---|---|
| `total` | integer | The total number of watchlists stored on controller db. |
| `watchlists` | array | A list of watchlists class objects. To get more information about the user objects please check the watchlists sections. |

**Response example**


```
{
  "controller_id": "3d797309-1bfe-4b6d-be07-e65d65f3b411",
  "reply_received": "2025-04-01T08:35:23.397Z",
  "request_created": "2025-04-01T08:35:17.696Z",
  "request_id": 2,
  "request_processed": "2025-04-01T08:35:23.396Z",
  "request_status": "Ok",
  "request_user_id": 7,
  "total": 2,
  "debug_messages": [ "RPC V2 authenticated user is 'ccalvaro2'."  ],
  "watchlists": [
    {
      "name": "dbWatchlist",
      "id": 1,
      "owner_id": 3,
      "page_by": "cluster",
      "properties": "empty",
      "clusters": [
        "12",
        "13",
        "14"
      ],
      "grid": [ "2x2"  ],
      "topics": [
        "topic1",
        "topic3"
      ]
    },
    {
      "name": "w1",
      "id": 2,
```

```
      "owner_id": 0,
      "page_by": "cluster",
      "properties": "",
      "clusters": [ "1"  ],
      "grid": [  ],
      "topics": [ "t1"  ]
    }
  ]
}
```

### deleteWatchlist

!!! post "POST /v2/watchlists" This call can be used to delete an existing watchlist.

### Request

| Parameter | Type | Required | Description |
|---|---|---|---|
| `watchlist_id` | integer | Required | The ID of the watchlist to delete. |

```
{
  "operation": "deleteWatchlist",
  "request_created": "2025-04-01T08:49:50.860Z",
  "request_id": 2,
  "watchlist_id": 2
}
```

## 4.13 ClusterControl Domain Specific Language (CCDSL)

CCDSL is the language used to write ClusterControl **advisors**, **programs**, and **widgets**, small scripts that run inside the ClusterControl `cmon` controller process and are packaged into tarballs (such as `s9s.tar.gz` and `common.tar.gz`) that `cmon` loads on first start, or after an upgrade.

The syntax is JavaScript-*flavored*, with extensions for accessing ClusterControl's internal data structures and functions, for example, executing SQL statements, running shell commands across cluster hosts, reading historical monitoring data, editing configuration files, raising alarms, enqueueing controller jobs, and rendering graphs. The results are then processed into advisors, alerts, or other actions.

---

### 4.13.1 1. Language basics

#### 1.1 How CCDSL differs from JavaScript

- Semicolons are mandatory, as in C/C++.
- Numbers are not all double-precision floats: `Int`, `Ulonglong`, and `Double` are distinct types, needed to represent disk sizes and network traffic measured in bytes without losing precision.
- Associative arrays are a dedicated `Map` type.

- Arrays are two-dimensional but can be used as one-dimensional (`a[10, 11]` and `a[10]` are both valid).
- There is a distinct `List` type.
- Namespaced/static-style functions use C++ notation (`JSON::parse(text)`, `cluster::hosts()`) rather than JavaScript's dot notation.
- Variables created on the fly inside a function are local to that function, not global.
- The language implements a C-like `#include` preprocessor directive.

### 1.2 Types

```
var a = new Int;
var b = new CmonHost();
passed = a === 0 && b.typeName() == "CmonHost";
```

Built-in types: `Int`, `Bool`, `Double`, `Ulonglong`, `String`, `Error`, `Map`, `List`, `Array`, `CmonHost`, `CmonMySqlHost`, `CmonGaleraHost`, `CmonGroupReplHost`, `CmonPostgreSqlHost`, `CmonNdbHost`, `CmonMongoHost`, `CmonMaxScaleHost`, `CmonProxySqlHost`, `CmonAdvice`, `CmonClusterConfig`, `CmonFile`, `CmonGraph`, `CmonJob`, `CmonRegExp`, `CmonDateTime`.

### 1.3 Literals

**Boolean literals**: `true` and `false`.

**String literals** — single or double quoted; single-quoted strings may contain double quotes and vice versa. Adjacent string literals concatenate at parse time, C-style:

```
var carName1 = "Volvo XC60";
var carName2 = 'Volvo XC60';
var answer1  = "It's alright";
var answer2  = "He is called 'Johnny'";

var a = "one "
        "two "
        "three";
passed = a == "one two three";
```

**Integer literals**, including hex:

```
var1 = 0xff;
var2 = 0XFFFF;
passed = var1 == 255 && var2 == 65535;
```

**Unsigned long long literals** — an integer literal too large for `Int` is automatically stored as `Ulonglong`. A literal can also be forced to this type with an `ull` suffix:

```
var a = 91872698761001;
var b = 10ull;
passed = a.typeName() == "Ulonglong" && b.typeName() == "Ulonglong";
```

**Double literals** — any number not fitting an integer type:

```
var a = 10.2;
var b = 10.8E11;
var c = 2.8e-10;
```

**Error literals** — sentinel values representing "not a real value" conditions:

| Literal | Meaning |
|---|---|
| `#NULL` | Null value |
| `#DIV/0!` | Division by zero |
| `#VALUE!` | Type mismatch, e.g. the log of a string |
| `#REF!` | Invalid variable reference, missing variable |
| `#NAME?` | Function name not found |
| `#NUM!` | Invalid numerical value, e.g. `sqrt(-1)` |
| `#N/A` | Value is not available — this is what an unpassed argument to `main()` evaluates to |
| `#SYNTAX!` | Syntax error |
| `#ARGS!` | Wrong argument count for a function call |

**Map literals** — associative arrays that can hold any value type, including nested maps, indexed by string keys:

```
var a = {};
a["one"] = {};
a["one"]["two"] = "value";
passed = a.typeName() == "Map" && a["one"]["two"] == "value";

var testmap = {};
testmap["key1"] = "test";
testmap["key2"] = "test";
keys = testmap.keys();
for (i = 0; i < keys.size(); ++i) {
    print(keys[i] + ": " + testmap[keys[i]]);
}
```

**Regular expression literals** — same syntax as JavaScript, typed as `CmonRegExp`:

```
var regexp = /([0-9]+)x([0-9]+)/i;
var string = "s: 640x480";
string.replace(regexp, "$2x$1");
// string === "s: 480x640"
```

### 1.4 Preprocessor directives

```
#include "filepath"
#pragma once
```

`#include` works as in C/C++. Two kinds of path are used:

- **Built-in headers** — `cmon/alarms.h`, `cmon/io.h`, `cmon/graph.h` — bring named constants into scope (severities and alarm types, `toString()` format specifiers, graph plot styles).

- **Script includes**, such as `common/mysql_helper.js` or `common/helpers.js` — shared library files, included by their path relative to the project root, that provide reusable helper functions.

`#pragma once` protects an include file from being included multiple times.

---

### 4.13.2 2. Program structure

#### 2.1 Entry points

- **`main(...)`** — if a script defines `main()`, it runs after any top-level statements. The arguments passed to `main()` come from the calling environment; the return value of `main()` becomes the script's exit status, unless `exit()` was called first.

```
var global1 = 10;
function main(arg1) {
    return arg1 == "UtCmonImperative" && global1 === 10 && global2 ===␣
↪11;
}
var global2 = 11;
```

- **`exit(exitstatus)`** — ends script execution and returns the given exit status to the Cmon environment; a normal program termination.
- **`abort()`** — aborts execution and presents a backtrace showing exactly where `abort()` was called.

If a function is called with the wrong number of arguments, the call evaluates to an `#ARGS!` error rather than raising an exception.

#### 2.2 Input/output functions

- `print([value]...)` — prints all arguments as one message at `info` severity, and returns the printed string.
- `warning([value]...)` — same, at `warning` severity.
- `error([value]...)` — same, at `critical` severity.

```
print("ERROR: ", retval["errorMessage"]);
print(host, ": ", msg);
```

---

### 4.13.3 3. Advisors, programs, and widgets

A CCDSL script is one `.js` file. Its location within the bundle and its calling convention determine what kind of script it is.

#### 3.1 Advisors

An advisor lives under a category directory (for example `mysql/innodb/`, `host/`, `ndbcluster/`), runs on a schedule, and produces `CmonAdvice` for the cluster it examines. An advisor takes no arguments (or, occasionally, one optional flag) and follows this shape:

```
#include "common/mysql_helper.js"
#include "cmon/alarms.h"

var DESCRIPTION = "One or two sentences describing what this checks and why␣
↪it matters.";
var TITLE = "Short human-readable title";
var OK_THRESHOLD = 999;
var WARNING_THRESHOLD = 990;

function main()
{
    var hosts = cluster::mySqlNodes();
    var advisorMap = {};

    for (idx = 0; idx < hosts.size(); ++idx)
    {
        host = hosts[idx];
        if (!host.connected())
            continue;

        var advice = new CmonAdvice();
        advice.setHost(host);
        advice.setTitle(TITLE);

        // ... compute a value, decide a severity ...

        advice.setSeverity(Ok);   // Ok, Warning, or Critical
        advice.setJustification("measured value and how it was computed");
        advice.setAdvice("what the administrator should do, if anything");

        advisorMap[idx] = advice;
    }
    return advisorMap;
}
```

`DESCRIPTION`, `TITLE`, and any threshold or message constants are declared as top-level `vars` before `main()`. `main()` returns a `Map` keyed by loop index, with one `CmonAdvice` per examined host — `advisorMap[idx] = advice` for each host, `return advisorMap;` at the end. This is the return contract every advisor implements, including ones that only ever produce a single entry.

`cluster::hosts()` can return the same physical host more than once — for example once by its controller role and once by its database role. When an advisor must examine each physical host only once, it deduplicates by host name:

```
var examinedHostnames = "";
for (idx = 0; idx < hosts.size(); ++idx) {
    host = hosts[idx];
    if (!host.connected()) continue;
    if (examinedHostnames.contains(host.hostName())) continue;
    examinedHostnames += host.hostName();
    ...
```

(continues on next page)

(continued from previous page)

```
}
```

Rate- or ratio-based checks should confirm the server has been running long enough, and handling enough traffic, for the computed value to be meaningful, before comparing it against a threshold — see `checkPrecond()` in *section 21.1*.

Some advisors also remediate: they read a configuration value, compute a corrected one, apply it immediately with `setGlobalVariable()` (a runtime `SET GLOBAL`), and persist it with `host.config()` → `config.setVariable()` → `config.save()` (a config-file change, which only takes effect after a restart). Doing both keeps the running value and the on-disk value consistent.

An advisor that only applies to certain cluster types can check `cluster::type()` and return a single `Ok` advice with an explanatory justification for cluster types it does not support:

```
var clusterType = cluster::type().toString();
if (clusterType != "REPLICATION") {
    var advice = new CmonAdvice();
    advice.setSeverity(Ok);
    advice.setJustification("Advisor supports only REPLICATION clusters.");
    advice.setAdvice("Nothing to do.");
    advisorMap[0] = advice;
    return advisorMap;
}
```

An advisor is not required to produce `CmonAdvice` at all — a script whose purpose is a human-readable report can simply `print()` its output and `return true;`.

### 3.2 Programs

A program lives under a `programs/` directory, is invoked on demand with explicit arguments (rather than on a schedule), and validates every argument before doing any work:

```
/**
 * explain query
 * Format:   db  query
 * db : "mydb"
 * query : "select 1"
 */
function main(db, q)
{
    var result = [];

    if (db.toString() == "" || db == #N/A || db.empty()) {
        result["error_msg"] = "Argument 'db' not specified";
        print(result["error_msg"]);
        exit(result);
    }
    if (q.toString() == "" || q == #N/A || q.empty()) {
        result["error_msg"] = "Argument 'query' not specified";
        print(result["error_msg"]);
        exit(result);
```

(continues on next page)

(continued from previous page)

```
    }
    // ... perform the work, then exit(result) ...
}
```

An argument is checked against all three conditions — `x.toString() == ""`, `x == #N/A`, and `x.empty()` — since any one check alone can miss a case depending on how the caller passed or omitted the argument. A doc comment above `main()` documenting the expected argument order and format is standard practice.

### 3.3 Widgets

A widget lives under a `widgets/` directory and is invoked by a user-interface element rather than a schedule. Widgets follow the same argument-validation convention as programs and are typically grouped into subdirectories by the UI surface that calls them (for example `widgets/config/`, `widgets/qm/` for query-monitor actions such as `explain`, `kill`, and `processlist`, `widgets/schema/`, `widgets/usermgmt/`). Because a widget backs a specific UI element, its header comment should note that modifying it can affect that UI feature.

---

## 4.13.4 4. Cluster and host objects

### 4.1 Cluster functions

| Function | Returns |
|---|---|
| `cluster::hosts()` | every host considered part of the cluster, including the controller's own host, as plain `CmonHost` objects (see below) |
| `cluster::mySqlNodes()` | the `CmonMySqlHost`s of the cluster |
| `cluster::galeraNodes()` | the `CmonGaleraHost`s of the cluster |
| `cluster::groupReplNode` | the `CmonGroupReplHost`s of the cluster (MySQL Group Replication) |
| `cluster::postgreSqlNod` | the `CmonPostgreSqlHost`s of the cluster |
| `cluster::mongoNodes()` | the `CmonMongoHost`s of the cluster |
| `cluster::ndbdNodes()` | the `CmonNdbHost`s of the cluster (NDB Cluster data nodes) |
| `cluster::maxscaleNodes` | the `CmonMaxScaleHost`s of the cluster |
| `cluster::proxysqlNodes` | the `CmonProxySqlHost`s of the cluster |
| `cluster::id()` | the cluster's numeric ID |
| `cluster::name()` | the cluster's display name |
| `cluster::type()` | the cluster type, as a string (for example `"REPLICATION"`, `"GALERA"`, `"MONGODB"`) |
| `cluster::state()` | the numeric cluster state |
| `cluster::vendor()` | the database vendor of the cluster, as a string (for example `"percona"`, `"mariadb"`) |
| `cluster::rollingRestar` | restarts the nodes of the cluster one at a time, without stopping it |

`cluster::hosts()` always builds its result as literal `CmonHost` objects, regardless of the underlying node's real type — including for node types that have no dedicated `cluster::*Nodes()` accessor of their own (Redis/Valkey, MSSQL, Elasticsearch, and ClickHouse nodes; see *section 4.4*). Every `CmonHost` function documented in *section 4.2* works on the result, but subtype-specific functions (*section 4.3*) require going through the matching `cluster::*Nodes()` call instead.

### 4.2 `CmonHost` tag functions

Identity and status:

```
host.hostName()          // string
host.port()              // int, usually the SQL server port
host.ipAddress()         // string, IPv4 address
host.dataHostName()      // string, the hostname used for data-layer
↪connections
host.internalHostName()  // string, the hostname used for internal cluster
↪communication
host.clusterId()         // int
host.nodeType()          // "controller" | "mysql" | "galera" | "mongo" |
```

(continues on next page)

(continued from previous page)

```
↪"maxscale" | "proxysql" |
                          // "redis" | "valkey" | "mssql" | "elastic" |
↪"clickhouse" | ...
host.hostStatus()         // string, human-readable node status
host.role()               // "controller" | "master" | "slave" | ...
host.connected()          // boolean
host.message()            // human-readable status description
host.serverVersion()      // SQL server version string, or the Cmon version␣
↪for the controller host
host.isAtLeastVersion(versionString)   // boolean, minimum-version check
host.dataDir()            // string, database data directory
host.distributionName() / .distributionCodeName() / .distributionRelease()
host.toMap()              // the host's properties as a Map
host.toJSonString()       // the host as a JSON string
```

`host.toMap()` returns a map of the host's properties; keys commonly used include `connected`, `nodetype`, `isgalera`, `readonly`, and, for Galera hosts, a nested `galera` map with `galerastatus` and `localstatusstr` (for example `"Synced"`).

Statistics:

```
host.memoryInfo()                                  // latest memory statistics,␣
↪as a Map
host.memoryStats(startTime, endTime)               // memory statistics over a␣
↪period, as a List
host.sqlInfo()                                      // latest SQL server␣
↪statistics, as a Map
host.sqlStats(startTime, endTime)                  // SQL server statistics␣
↪over a period, as a List
host.mongoStats(startTime, endTime)                // MongoDB statistics over a␣
↪period, as a List
host.networkInfo()                                  // one Map per monitored␣
↪network interface
host.diskInfo()                                     // one Map per monitored␣
↪disk partition
host.diskStats(startTime, endTime, [deviceName]) // disk statistics over a␣
↪period, as a List
host.cpuInfo()                                      // one Map per CPU core
host.cpuStats(startTime, endTime, [coreId])        // CPU statistics over a␣
↪period, as a List
```

Each `*Stats(startTime, endTime)` call returns a `List` of per-sample `Maps` (each including a `created` timestamp field). Convert a `List` to an `Array` for statistical processing with `.toArray(fieldNames)` — see *section 6*.

Execution and configuration:

```
host.system(command)                          // shell command; {success,␣
↪errorMessage, result}
host.executeSqlQuery(query)                   // SELECT-style query;
↪{success, errorMessage, result}
```

(continues on next page)

(continued from previous page)

```
host.executeSqlCommand(sqlCommand)          // INSERT/UPDATE/DDL-style␣
↪command; {success, errorMessage}
host.executeMongoQuery(dbname, query)       // Mongo query; {success,␣
↪errorMessage, result}
host.sqlPing([timeoutSeconds])              // boolean
host.sqlSystemVariable(name) / host.sqlSystemVariables()   // cached SHOW␣
↪GLOBAL VARIABLES (SHOW ALL on PostgreSQL)
host.sqlStatusVariable(name) / host.sqlStatusVariables()   // cached SHOW␣
↪GLOBAL STATUS
host.config([fileName])                     // configuration as a␣
↪CmonClusterConfig, see [section␣
↪9](#9-configuration-files-cmonclusterconfig)
```

Alarms:

```
host.checkValue(type, value)     // checks a value against a registered␣
↪check type, raising/clearing as needed
host.raiseAlarm(type, severity, [message])
host.clearAlarm(type)
host.alarms()                    // active alarms for this host
```

### 4.3 Subtype tag functions

`CmonMySqlHost` inherits all `CmonHost` properties and functions, and adds:

```
host.isGalera()    // boolean
host.uptime()      // ulonglong, SQL server uptime in seconds
host.readOnly()    // boolean, the 'read_only' SQL variable
```

Everywhere a `CmonHost` function is documented, it is also usable on `CmonMySqlHost`, `CmonGaleraHost`, `CmonGroupReplHost`, `CmonPostgreSqlHost`, `CmonNdbHost`, `CmonMongoHost`, `CmonMaxScaleHost`, and `CmonProxySqlHost` — this is the complete set of `CmonHost` subclasses a script can obtain a typed handle for, via the matching `cluster::*Nodes()` function in *section 4.1* or via `getSelf()` inside a subtype-specific tag function.

### 4.4 Node types without a typed host class

Redis/Valkey, MSSQL (Availability Group or standalone), Elasticsearch, and ClickHouse nodes are fully managed cluster types, but none of them has a `cluster::*Nodes()` accessor or a corresponding entry in *section 1.2*'s built-in type list — there is no `cluster::redisNodes()`, `cluster::mssqlNodes()`, `cluster::elasticNodes()`, or `cluster::clickhouseNodes()`, and no way to construct a typed handle for one of these hosts from a script.

These nodes are still reachable through `cluster::hosts()`, which returns them as plain `CmonHost` objects, distinguishable by `host.nodeType()` (`"redis"`, `"valkey"`, `"mssql"`, `"elastic"`, or `"clickhouse"`). Every generic `CmonHost` function from *section 4.2* works on them normally — `hostName()`, `port()`, `connected()`, `system()`, `raiseAlarm()`/`clearAlarm()`, `diskInfo()`/`diskStats()`, `cpuInfo()`/`cpuStats()`, `memoryInfo()`/`memoryStats()`, `networkInfo()`, `config()`. What isn't available is a type-specific query method equivalent to `executeSqlQuery()` or `executeMongoQuery()`, so engine-specific data has to come from shelling out with `host.system()` to that engine's own CLI or REST API:

```
var hosts = cluster::hosts();
for (idx = 0; idx < hosts.size(); ++idx) {
    host = hosts[idx];
    if (host.nodeType() != "redis" && host.nodeType() != "valkey")
        continue;

    var retval = host.system("redis-cli -p " + host.port() + " INFO␣
↪replication");
    if (retval["success"])
        print(retval["result"]);
}
```

The same pattern (a real, managed node type reached only through `cluster::hosts()` plus `host.system()`, with no dedicated query method) already applies to MaxScale-adjacent tooling such as `maxscale/ms_admin.js`, which shells out to `maxctrl` rather than calling a typed MaxScale query function.

---

### 4.13.5 5. Executing SQL commands and queries

`host.executeSqlQuery()`, `host.executeSqlCommand()`, and `host.system()` all return a `Map` with a `success` boolean, an `errorMessage` string, and, for queries and shell commands, a `result` value.

```
function getSqlVariable(host, variableName)
{
    var query = "SHOW GLOBAL STATUS LIKE '" + variableName + "'";
    var retval = host.executeSqlQuery(query);
    if (!retval["success"]) {
        print("ERROR:", retval["errorMessage"]);
        return #N/A;
    }
    var value = retval["result"][0, 1];   // two-dimensional indexing: row␣
↪0, column 1
    if (value.looksInteger()) return value.toInt();
    if (value.looksULongLong()) return value.toULongLong();
    if (value.looksDouble()) return value.toDouble();
    return value;
}
```

`host.executeSqlQuery()` and `host.executeSqlCommand()` work identically on `CmonMySqlHost`, `CmonGaleraHost`, `CmonGroupReplHost`, and `CmonPostgreSqlHost` — a PostgreSQL advisor can run the same query/command pattern shown here against a `CmonPostgreSqlHost`, substituting PostgreSQL SQL for MySQL SQL (for example `SELECT * FROM pg_stat_activity` in place of `SHOW GLOBAL STATUS`).

Executing a shell command:

```
function listFiles(host)
{
    var retval = host.system("ls -lha /home");
    if (!retval["success"])
```

(continues on next page)

(continued from previous page)

```
        error("ERROR: ", retval["errorMessage"]);
    print("Host    : ", host.hostName());
    print("Result  : ", retval["result"]);
    print("Success : ", retval["success"]);
    return retval["success"];
}
```

Executing a query against the Cmon database itself (rather than a managed host):

```
var retval = CmonDb::executeSqlQuery("select * from mysql_states;");
if (!retval["success"])
    error("Executing SQL query failed: ", retval["errorMessage"]);

for (idx = 0; idx < retval["result"].rows(); ++idx) {
    print(retval["result"][idx, 2]);
}
```

---

### 4.13.6 6. Obtaining and processing statistical information

```
#include "cmon/io.h"
function toGigaBytes(value) { return value / (1024 * 1024 * 1024); }
function main()
{
    var host = cluster::hosts()[0];
    var endTime = CmonDateTime::currentDateTime();
    var startTime = endTime - 10 * 60;
    var stats = host.memoryStats(startTime, endTime);
    for (idx = 0; idx < stats.size(); ++idx) {
        map = stats[idx];
        created = CmonDateTime::fromUnixTime(map["created"]);
        ramfree = toGigaBytes(map["ramfree"]);
        print(created.toString(LongTimeFormat), " ",␣
↪ramfree.toString(TwoDecimalNumber), "GBytes");
    }
    return true;
}
```

For most tasks, converting a `List` to an `Array` with `.toArray(fieldName)` and using the high-level statistical functions is more efficient than looping over individual samples:

```
function printUtil(host, startTime, endTime)
{
    var list = host.memoryStats(startTime, endTime);
    var array = list.toArray("memoryutilization");
    var min = min(array);
    var max = max(array);
    var ninth = percentile(array, 0.9);
    print(host.hostName(), " ",
```

(continues on next page)

(continued from previous page)

```
        min.toString(TwoDecimalPercent), " - ",
        ninth.toString(TwoDecimalPercent), " - ",
        max.toString(TwoDecimalPercent));
    return true;
}
```

`.toArray()` also accepts a comma-separated list of fields to produce a multi-column array, useful for graphing or forecasting aligned x/y series:

```
var stats = host.sqlStats(startTime, endTime);
var array = stats.toArray("created,interval,COM_SELECT,COM_INSERT");
```

Linear forecasting from historical data:

```
var list = host.diskStats(startTime, endTime, device);
var freeOverTime      = list.toArray("free");
var timeOfEachSample  = list.toArray("created");
var expectedFree = forecast(futureTime.toInt(), freeOverTime,␣
↪timeOfEachSample);
expectedFree = expectedFree < 0 ? 0 : expectedFree;   // linear regression␣
↪can predict below zero
```

### 4.13.7 7. Advice: `CmonAdvice`

`CmonAdvice` represents an action to be taken by the administrator, together with the information needed to understand why it was raised.

```
var advice = new CmonAdvice();
advice.setHost(host);
advice.setTitle("Short title");
advice.setSeverity(Warning);          // Ok, Warning, or Critical — from
↪#include "cmon/alarms.h"
advice.setJustification("Measured 96% CPU over the last 60 minutes.");
advice.setAdvice("Investigate top queries or scale up the instance.");
```

| Function | Description |
|---|---|
| `advice.setTitle(title)` / `advice.title()` | a short description of the advice |
| `advice.setCreator(creator)` / `advice.creator()` | the owner that created the advice; defaults to the source file |
| `advice.setJustification(text)` / `advice.justification()` | a detailed, human-readable description of the measured value and why an action is suggested |
| `advice.setAdvice(text)` / `advice.advice()` | the recommended action |
| `advice.setSeverity(severity)` / `advice.severity()` | the severity level: `Ok` (0), `Warning` (1), or `Critical` (2) |
| `advice.setHost(host)` | the host the advice was generated for |

`advice.toString(format)` supports these format specifiers:

| Specifier | Meaning |
|---|---|
| `%t` | the title |
| `%j` | the justification |
| `%a` | the advice text |
| `%c` | the creator |
| `%h` | the host name, if set |
| `%E` | a multi-line summary of all of the above |

An advisor's `main()` returns a `Map` keyed by loop index, holding one `CmonAdvice` per examined host: `advisorMap[idx] = advice;`, then `return advisorMap;`.

---

### 4.13.8 8. Alarms

`#include "cmon/alarms.h"` provides the severity constants (`Ok`, `Warning`, `Critical`) and a set of predefined alarm-type constants that can be raised directly, without registration:

```
#include "cmon/alarms.h"
host.raiseAlarm(HostDiskUsage, Warning, "message");
host.clearAlarm(HostDiskUsage);
```

Predefined alarm types include `HostDiskUsage`, `StorageUtilizationPrediction`, `MySqlReplicationConfigProblem`, `ClusterSystemCheck`, `MySqlAdvisor`, `MySqlTimeZoneCheck`, `StorageMyIsam`, `MySqlTableAnalyzer`, `MySqlIndexAnalyzer`, and `MySqlAutoIncrementAnalyzer`, among others. A condition specific to an engine that has no matching predefined type (for example PostgreSQL table bloat) needs a custom alarm type registered as shown below before it can be raised with `host.raiseAlarm()`; until such a type exists, an advisor can only report the condition through `CmonAdvice` severity.

For conditions that don't match a predefined type, register a custom one:

```
#include "cmon/alarms.h"
function myAlarm() {
    return Alarm::alarmId(
        Node, true,
        "Computer is on fire",
        "The computer is on fire, it is on flames.",
        "Pour some water on it.");
}
var myAlarmId = myAlarm();
var host = cluster::hosts()[0];
host.raiseAlarm(myAlarmId, Critical);
```

- `Alarm::alarmId(category, isHost, title, message, recommendation)` registers a new alarm type (or returns the existing one if an identical type is already registered) and returns its ID.
- `Alarm::checkId(category, isHost, warningLevel, criticalLevel, title, message, recommendation)` registers a check type — an alarm with warning and critical numeric thresholds — for use with `host.checkValue(type, value)`.

Reading active alarms:

```
#include "cmon/alarms.h"
var alarms = host.alarms();
var found = false;
for (idx = 0; idx < alarms.size(); ++idx) {
    if (alarms[idx]["title"] == "MySQL advisor alarm") {
        found = true;
        break;
    }
}
```

An alarm that is raised in a failing branch should be cleared with a matching `clearAlarm()` call in the corresponding healthy branch, so that it does not remain active once the underlying condition is resolved.

---

### 4.13.9 9. Configuration files: `CmonClusterConfig`

```
function getConfiguredClientPort(host)
{
    var config   = host.config();
    var variable = config.variable("port");
    for (idx = 0; idx < variable.size(); ++idx) {
        print("*** section  : ", variable[idx]["section"]);
        print("*** value    : ", variable[idx]["value"]);
        print("*** location : ", variable[idx]["filepath"], ":",␣
↪variable[idx]["linenumber"]);
        if (variable[idx]["section"] == "client")
            return variable[idx]["value"].toInt();
    }
    return #N/A;
}
```

| Function | Description |
|---|---|
| `host.config([fileName])` | loads the configuration from the host, together with its include files and directories, as a `CmonClusterConfig` |
| `config.errorMessage()` | a human-readable message describing the state of the last operation |
| `config.variable([variableName])` | a `List` of `Maps` (`variablename`, `linenumber`, `value`, `filepath`, `section`); returns every variable in the file if no name is given |
| `config.setVariable(section, variableName, value)` | sets (or adds) a variable in the given section; changes the in-memory configuration object only |
| `config.save()` | saves the configuration back to the original host(s) and file name(s); returns `{success, errorMessage}` |

Changing a variable with `config.setVariable()` and `config.save()` updates the configuration file; it does not change the running value. To apply a dynamic variable immediately as well, issue a `SET GLOBAL` through `host.executeSqlCommand()` in addition to saving the configuration file.

### 4.13.10 10. Files: `CmonFile`

`CmonFile` represents a single file — either local to the controller, or on a managed host — and provides read-only inspection and reading functions. It is a different tool from `CmonClusterConfig` (*section 9*): `CmonClusterConfig` understands a config file's `section`/`variable`/`value` structure, while `CmonFile` reads any file as raw text or lines, or just inspects its metadata.

```
var file = new CmonFile("/etc/mysql/my.cnf", host);    // a file on a␣
↪managed host
// var file = new CmonFile("/etc/cmon.cnf");            // a file on the␣
↪controller itself

if (!file.exists()) {
    print("File not found: ", file.fullPath());
} else if (file.isReadable()) {
    var content = file.readTxtFile();
    var lines   = file.readLines();
    print(file.fileName(), " is ", content.length(), " bytes, ",␣
↪lines.size(), " lines");
}
```

| Function | Description |
|---|---|
| `new CmonFile(path, [host])` | opens `path`; with no `host` argument the file is on the controller, otherwise on the given `CmonHost` |
| `file.fileName()` | the file name, without its directory |
| `file.baseName()` | the file name without its extension |
| `file.fullPath()` | the full, absolute path |
| `file.isRemote()` | true if the file is on a managed host rather than the controller |
| `file.exists()` | true if the file exists |
| `file.isDirectory()` | true if the path is a directory |
| `file.isExecutable()` | true if the file has execute permission |
| `file.isReadable()` | true if the file has read permission |
| `file.readTxtFile()` | the file's contents as a `String` |
| `file.readLines()` | the file's contents as a `List` of lines |
| `file.errorString()` | a human-readable message describing the last failed operation |

### 4.13.11 11. Jobs: `CmonJob`

A `CmonJob` represents a task the Cmon controller executes, usually asynchronously: the script builds a `CmonJob`, sets its properties, and enqueues it; the controller then executes it and reports the results to the UI.

```
var job = CmonJob::createBackupJob("127.0.0.1", "/var/tmp");
job.setBackupMethod("mysqldump");
var passed = job.enqueue();
```

(continues on next page)

(continued from previous page)

```
if (passed !== true) {
    error("ERROR: ", job.errorString());
    exit(false);
}
```

| Function | Description |
|---|---|
| `job.enqueue()` | checks the job for consistency and sends it to the execution queue; returns `true` on success |
| `job.errorString()` | a human-readable error, or an empty string if there was none |
| `job.jobId()` | the job's numeric ID; valid (greater than 0) only after a successful `enqueue()` |
| `CmonJob::getJob(jobId)` | reads a previously enqueued job from the Cmon database; `#N/A` if not found |
| `CmonJob::createBackupJob(host, dir)` | creates a job that backs up a host's data |
| `CmonJob::createDoCheckJob()` | creates a job that runs schema/index checks on the controller |
| `CmonJob::createAddNodeJob(hostN [install], [configFile])` | creates a job that adds a new node to the cluster |
| `job.setBackupMethod(method)` | `null`/`"auto"`/`"none"` (default method), `"mysqldump"`, `"xtrabackupfull"`, `"xtrabackupincr"`, or `"pg-dump"` |
| `job.setCompression(compression)` | whether the backup file should be compressed |
| `job.setIncludeDatabases(value)` | which databases to include |
| `job.setIsCcStorage(value)` | whether the controller should store the backup file |
| `job.setNetcatPort(value)` | the netcat port used when copying the backup file over the network |

### 4.13.12 12. Graphs: `CmonGraph`

```
#include "cmon/graph.h"
var host      = cluster::hosts()[0];
var endTime   = CmonDateTime::currentDateTime();
var startTime = endTime - 10 * 60;
var stats     = host.sqlStats(startTime, endTime);
var array     = stats.toArray("created,interval,COM_SELECT,COM_INSERT");

for (idx = 0; idx < array.columns(); idx++) {
    array[5, idx] = 1000 * array[2, idx] / array[1, idx];
    array[6, idx] = 1000 * array[3, idx] / array[1, idx];
}

var graph = new CmonGraph;
graph.setXDataIsTime();
graph.setTitle("SQL Statistics " + host.toString());
graph.setSize(800, 600);
```

(continues on next page)

(continued from previous page)

```
graph.setPlotLegend(1, "Select (1/s)");
graph.setPlotColumn(1, 0, 5);
graph.setPlotStyle(1, Impulses);
graph.setPlotLegend(2, "Insert (1/s)");
graph.setPlotColumn(2, 0, 6);
graph.setPlotStyle(2, Impulses);
graph.setData(array);
exit(graph);
```

| Function | Description |
|---|---|
| `graph.setTitle(title)` | text shown at the top of the graph image |
| `graph.setSize(width, height)` | size of the generated graph, in pixels |
| `graph.setXDataIsTime([boolean])` | whether X-axis values should be printed as date/time |
| `graph.setPlotLegend(plotIdx, legend)` | legend text for the given plot |
| `graph.setPlotColumn(plotIdx, xColumn, yColumn)` | which array columns feed the given plot |
| `graph.setPlotStyle(plotIdx, style)` | plot rendering style; available styles are defined in `cmon/graph.h` |

Plot indices are 1-based.

### 4.13.13 13. JSON functions and the Cmon database

```
var req = new Map;
req["operation"] = "clusters";
var retval = JSON::postRequest("http://localhost:9500/0/clusters", req);
print("retval of clusters:\n" + retval);
```

| Function | Description |
|---|---|
| `JSON::parse(text)` | parses a JSON string, returns it as a `Map` |
| `JSON::toString(map` | converts a `Map` to a well-formatted JSON string |
| `JSON::postRequest(` `map)` | sends a POST request to the given URL with the JSON-encoded map as its body, returning the reply |
| `CmonDb::executeSql(` | executes a query against the Cmon database (the controller's own metadata store); returns `{success, errorMessage, result}` |

### 4.13.14 14. Date and time: `CmonDateTime`

`CmonDateTime` is distinct from JavaScript's `Date` type. It supports direct arithmetic in seconds:

```
var endTime    = CmonDateTime::currentDateTime();
var startTime  = endTime - 60 * 60;          // one hour ago
var futureTime = endTime + 2 * 24 * 3600;    // two days from now
```

| Function | Description |
|---|---|
| `CmonDateTime::currentDateTime()` | the current real-time clock time from the controller host |
| `CmonDateTime::fromUnixTime(time)` | converts a Unix timestamp to a `CmonDateTime` |
| `CmonDateTime::fromString(string)` | parses a date/time string |
| `dateTime.second()` / `.minute()` / `.hour()` | time-of-day components |
| `dateTime.month()` | the month, 1–12 |
| `dateTime.year()` | the year, e.g. 2024 |
| `dateTime.weekday()` | day of the week, Sunday = 1 … Saturday = 7 |
| `dateTime.timeZone()` | seconds to add to local time to get UTC |
| `dateTime.dayLight()` | seconds to add for daylight saving time |

`toString([format])` accepts either a named format constant from `cmon/io.h`, or a `strftime()`-style format string:

```
#include "cmon/io.h"
var dateTime = CmonDateTime::fromUnixTime(1424686003);
str01 = dateTime.toString(FileNameFormat);       // "2015-02-23_110643"
str03 = dateTime.toString(LogFileFormat);        // "Feb 23 11:06:43"
str04 = dateTime.toString(MySqlLogFileFormat);   // "2015-02-23 11:06:43"
str09 = dateTime.toString(ShortDateFormat);      // "02/23/15"
str10 = dateTime.toString(LocalDateTimeFormat);  // "Mon Feb 23 11:06:43␣
↪2015"
str11 = dateTime.toString(EmailDateTimeFormat);  // "Mon, 23 Feb 2015␣
↪11:06:43 +0100"
```

### 4.13.15 15. General value functions

Available on any value:

| Function | Description |
|---|---|
| `value.typeName()` | the type name of the value |
| `value.toString([formatId])` | converts to string, optionally using a format specifier from `cmon/io.h` |
| `value.empty()` | true if a string has no characters, or a container has no items |
| `value.size()` | character count for strings, item count for containers; for an `Array`, the number of columns |
| `value.isNull()` | true for a null string, e.g. an unset SQL value |
| `value.isInvalid()` | true if the value was never set |
| `value.isString()` / `.isInt()` / `.isULongLong()` / `.isDouble()` / `.isBoolean()` / `.isNumber()` / `.isError()` / `.isMap()` / `.isList()` / `.isArray()` | type checks |
| `value.toInt()` / `.toULongLong()` / `.toDouble()` / `.toBoolean()` | conversions |

```
#include "cmon/io.h"
var theDouble = 42.0;
var str1 = theDouble.toString(TwoDecimalNumber);   // "42.00"
var str2 = theDouble.toString(FourDecimalNumber);  // "42.0000"
```

### 4.13.16 16. String functions

Tag functions on a string value:

| Function | Description |
|---|---|
| `string.length()` | number of characters |
| `string.indexOf(substring, [start])` | position of the first match, or -1 |
| `string.split(separator)` | array of substrings |
| `string.substr(begin, length)` | substring extraction |
| `string.trim()` | removes leading/trailing whitespace |
| `string.toLowerCase()` / `.toUpperCase()` | case conversion |
| `string.contains(substring)` | boolean |
| `string.replace(from, to)` | substring replacement |
| `string.startsWith(other)` / `.endsWith(other)` | boolean, prefix/suffix check |
| `string.leftAlign(length)` / `.rightAlign(length)` | pads with trailing/leading spaces up to `length` (capped at 1000) — left- or right-aligns the text within a fixed-width field |
| `string.search(pattern)` | position of the first match of `pattern` (a string or a `CmonRegExp`), or -1 |
| `string.looksInteger()` / `.looksULongLong()` / `.looksDouble()` / `.looksBoolean()` / `.looksEmail()` / `.looksIpAddress()` | format checks |
| `string.toInteger()` / `.toULongLong()` | conversions |

Free functions: `asc(text)`, `char(number)` / `chr(number)`, `left(text, n)`, `right(text, n)`, `startswith(text1, text2)`, `endwith(text1, text2)`, `mid(text, start, length)`, `escape(text)` / `unescape(text)`, `upper(text)` / `lower(text)`, `trim(text)`, `len(text)`, `concatenate(text, [text]...)`.

---

### 4.13.17 17. Mathematical, array, and statistical functions

**Mathematical**: `rand()`, `pi()`, `degrees(n)`, `radians(n)`, `sign(n)`, `sin`/`asin`/`sinh`, `cos`/`acos`/`cosh`/`acosh`, `tan`/`atan`/`tanh`/`atanh`, `fisher(n)`/`fisherinv(n)`, `log(n, [base])`, `sqrt(n)`, `abs(n)`, `exp(n)`, `floor(n, [significance])`, `ceiling(n, [significance])`, `round(n, digits)`, `roundup(n, digits)`, `rounddown(n, digits)`, `mround(n, multiple)`, `even(n)`, `iseven(n)`, `odd(n)`, `isodd(n)`, `convert(n, fromUnit, toUnit)` (byte, kbyte, mbyte, gbyte, tbyte, celsius, kelvin, fahrenheit, hz, mhz, ghz).

**Array**: `choose(position, value, [value]...)`, `transpose(array)`, `filter rows(array, column, value)`, `columns(array)`, `rows(array)`, `vlookup(value, array, column, [notExact])`, `hlookup(value, array, column, [notExact])`, `match(value, array, [matchType])`.

**Statistical** — accept either individual arguments or arrays:

```
b = [10, 8, 5];
c = average(b);
d = average(10, 11, 12);
```

`count`, `countblank`, `min`, `max`, `sum`, `sumsq`, `product`, `average`, `geomean`, `mode`, `emaverage(alpha, ...)`, `median`, `percentile(array, [n])`, `small(array, n)`,

`large(array, n)`, `stdev`, `avedev`, `pearson(array1, array2)`, `correl(array1, array2)`, `covar(array1, array2)`, `devsq`, `var`, `forecast(x, knownYValues, knownXValues)`, `linest(knownYValues, knownXValues)`, `slope(knownYValues, knownXValues)`, `intercept(knownYValues, knownXValues)`.

**Value classification**: `iserr(value)`, `isnumeric(value)`, `istext(value)`, `isnumber(value)`, `isarray(value)`.

---

### 4.13.18 18. Regular expression functions

`CmonRegExp` tag functions:

| Function | Description |
|---|---|
| `regexp.test(string)` | true if the pattern matches |
| `regexp.lastIndex()` | index where the next match check begins; 0 without the "global" modifier |
| `regexp.match(string)` | a `List` with the matched text at index 0, and sub-expression matches from index 1 |
| `regexp.exec(string)` | a `Map` with detailed match information |

`toString()` format specifiers for `CmonRegExp`: `%r` (the pattern itself), `%j` (pattern and modifiers in JavaScript notation, e.g. `/[0-9]+/ig`), `%m` (the matched string, if any), `%nm` (the nth matched sub-expression).

---

### 4.13.19 19. Cmon, license, and introspection functions

| Function | Description |
|---|---|
| `cmon::version()` | the Cmon version, as a string |
| `cmon::build()` | the Cmon build number |
| `cmon::uptime()` | seconds since Cmon started managing this cluster |
| `cmon::running()` | true if Cmon is managing this cluster |
| `cmon::hostname()` | hostname of the controller's computer |
| `cmon::domainname()` | domain name of the controller's computer |
| `license::statustext()` | a short description of the license status |
| `license::status()` | true if the cluster has a valid license |
| `license::expires()` | days remaining on the license; negative if expired or not found |
| `conf::values()` | the controller's own `cmon.cnf`-level configuration, as a `Map` |
| `CmonMetaType::getParamSpec(className propertyName)` | the metadata (as a `Map`) describing one property of one internal class |
| `CmonCluster-Info::getClusterInfo([clusterId])` | a `CmonClusterInfo` for the given cluster, or the current one if omitted |
| `CmonServer::containers()` | the containers known to this controller (LXC/cloud), as a `CmonContainer` |
| `CmonUser::currentUser()` | the `CmonUser` the script is running as |

`conf::values()` is how a script checks a controller-level toggle before doing expensive work — for example, several `information_schema`-based MySQL advisors check `enable_is_queries` before running:

```
var cmonConfig = conf::values();
if (cmonConfig.keys().contains("enable_is_queries") &&
    !cmonConfig["enable_is_queries"].toBoolean())
{
    print("Information_schema queries are not enabled.");
    exit(result);
}
```

### 4.13.20 20. Advisor bundle structure

An advisor bundle is a tarball (for example `s9s.tar.gz`) containing a `project.conf` file at its root and a directory tree of `.js` scripts organized by category.

#### 20.1 Directory layout

```
s9s.tar.gz
└── s9s/
    ├── project.conf
    ├── host/                        # OS-level checks
    │   └── widgets/
    ├── mysql/
    │   ├── auto_tuners/             # advisors that also change configuration
    │   ├── connections/
    │   ├── galera/
    │   ├── general/
    │   ├── group_repl/
    │   ├── health/
    │   ├── i_s/                     # information_schema-based checks
    │   ├── innodb/
    │   ├── p_s/                     # performance_schema-based checks
    │   ├── programs/                # on-demand scripts, not scheduled
    │   ├── query_cache/
    │   ├── replication/
    │   ├── schema/
    │   ├── security/
    │   ├── table_cache/
    │   └── widgets/
    │       ├── config/
    │       ├── qm/                   # query monitor: explain, kill,␣
↪processlist
    │       ├── schema/
    │       └── usermgmt/
    ├── mongodb/
    │   ├── connections/, mmap/, replication/, sharding/
    │   └── widgets/qm/
```

(continues on next page)

(continued from previous page)

```
    ├── maxscale/
    ├── ndbcluster/
    ├── predictions/                # forward-looking checks
    └── reports/                    # print-only output, no CmonAdvice

common.tar.gz
└── common/
    ├── project.conf
    ├── helpers.js                  # generic, non-database-specific helpers
    ├── mysql_helper.js             # SQL helpers shared by MySQL/Galera␣
↪advisors
    └── cmonrpc.js                  # helpers for calling the Cmon RPC API␣
↪from a script
```

A shared library such as `common/` contains helper `.js` files, included by the other bundles with `#include "common/mysql_helper.js"`, and is not itself scheduled.

Neither tarball currently contains a category directory for PostgreSQL, Redis/Valkey, MSSQL, Elasticsearch, or ClickHouse; `project.conf` already reserves `[schedules-postgresql_single]` (see below) but it has no entries.

### 20.2 `project.conf`

```
[project]
# Increase this value to force cmon to reinstall the scripts (a backup
# of the previous version is made automatically) on the next start/upgrade.
version=20210922
# Increase this value when only the [schedules*] sections below changed.
schedule_version=1029

[schedules]
# Applies to every cluster type.
s9s/host/cpu_usage.js="*/30 * * * *"
s9s/host/disk_space_usage.js="*/30 * * * *"
s9s/host/swappiness.js="0 1 * * *"

[schedules-mysqlcluster]
s9s/mysql/connections/connections_used_pct.js="*/1 * * * *"
s9s/mysql/schema/schema_check_nopk.js="0 1 * * *"

[schedules-replication]
...

[schedules-group_replication]
...

[schedules-galera]
...

[schedules-mysql_single]
```

(continues on next page)

(continued from previous page)

```
...

[schedules-postgresql_single]
...

[schedules-mongodb]
...
```

- `[project]` holds a `version` and a `schedule_version`. Increasing `version` causes `cmon` to reinstall the bundle's scripts (backing up the previous versions first); increasing `schedule_version` is sufficient when only the schedule sections changed.
- `[schedules]` lists scripts scheduled for every cluster type.
- `[schedules-<clustertype>]` lists scripts scheduled only for that cluster type; `<clustertype>` is the same cluster-type identifier used when the cluster was created (for example `mysqlcluster`, `replication`, `group_replication`, `galera`, `mysql_single`, `postgresql_single`, `mongodb`).
- Each entry's key is the script's path relative to the tarball root, including the top-level directory name, and its value is a quoted five-field cron expression.
- A script that is present in the bundle but commented out of its `[schedules*]` entry is shipped but not scheduled by default.
- Scripts under a `programs/` or `widgets/` directory are not listed in any `[schedules*]` section; they are invoked on demand instead.
- `common/project.conf` contains only a `[project]` section with a `version`, since the common bundle is a shared library and has nothing to schedule.

---

### 4.13.21 21. Shared helper library (`common/*.js`)

### 21.1 common/mysql_helper.js

| Function | Description |
|---|---|
| `readVariable(host, name)` | runs `SHOW GLOBAL VARIABLES LIKE '<name>'`; returns the value, or `false` on error |
| `readStatusVariable(host, name)` | runs `SHOW GLOBAL STATUS LIKE '<name>'`; returns the value, or `false` on error |
| `getSingleValue(host, query)` | runs a query, returns the single scalar from its first row/column |
| `getValueMap(host, query)` | runs a query, returns the result set as a `{row: {column: value}}` map |
| `getResultSet(host, query)` | runs a query, returns the raw `executeSqlQuery()` reply |
| `executeSqlCommand(host, query)` | runs a command, prints any error, and returns `true`/`false` |
| `executeSqlCommand2(host, query)` | runs a command and returns the raw reply map |
| `setGlobalVariable(host, variable, value)` | builds and runs the appropriate `SET GLOBAL variable=value` statement, quoting the value correctly for numeric or string types |
| `mySleep(host, seconds)` | runs `SELECT SLEEP(seconds)` |
| `checkPrecond(host)` | returns true when the host has been up for more than an hour and is handling more than 10 queries per second — used to avoid evaluating rate- or ratio-based checks against a freshly started or idle server |
| `isSystemTable(name)` | true for `performance_schema`, `information_schema`, `mysql`, or `sys` |
| `checkHostVersion(host, versionPrefix)` | true if the host's server version starts with the given prefix |
| `isMySqlHost(host)` | true for MySQL or Galera node types |
| `isMySql80Host(host)` / `isMySql55Host(host)` | version-specific MySQL checks |
| `isMariaDb100Host(host)` / `isMariaDb101Host(host)` / `isMariaDb102Host(host)` / `isMariaDb103Host(host)` / `isMariaDb104Host(host)` / `isMariaDb10xHost(host)` | version-specific MariaDB checks |

Despite the filename, `getSingleValue()`, `getValueMap()`, and `getResultSet()` are just thin wrappers around `host.executeSqlQuery()` and carry no MySQL-specific logic, so they are safe to reuse from a PostgreSQL advisor. `readVariable()` and `readStatusVariable()` are MySQL-specific (`SHOW GLOBAL VARIABLES`/`SHOW GLOBAL STATUS` syntax) and have no PostgreSQL equivalent in this file.

### 21.2 common/helpers.js

| Function | Description |
|---|---|
| `hostMatchesFilter(hostAndPort)` | true if a host matches a `"hostname:port"` filter string, or if the filter is empty or `"*"` |
| `executeOnController(command)` | runs a shell command specifically on the host whose node type is `controller` |
| `isMongoDbPrimary(host)` | true if the given MongoDB host is currently the primary, using the version-appropriate admin command (`hello` on newer releases, `isMaster` on older ones) |

### 21.3 common/cmonrpc.js

Helpers for calling the Cmon RPC API from within a script, by running `curl` on the controller host:

| Function | Description |
|---|---|
| `getCmonController()` | returns the `CmonHost` whose role is `controller`, or `false` if none is found |
| `curl(controller, path, json)` | POSTs a JSON payload to the controller's RPC endpoint for the given cluster and path, returning the result |
| `mapToJSON(map)` / `JSONToMap(text)` | converts between `Map` and JSON text |
| `addNode(node)` | enqueues an `addnode` job for the given hostname via the RPC API |
| `setCmonrpcToken(json, clusterId)` | attaches the RPC authentication token for the given cluster ID to a request payload |
| `findUnusedHost()` | returns the first configured free host not already part of the cluster |

`CMONRPC_HOST`, `CMONRPC_PORT`, and `CMONRPC_TOKEN` are the controller's RPC host, port, and per-cluster authentication tokens, referenced by these helpers when building a request.

## 4.14 ClusterControl Client SDK

ClusterControl has a HTTPS based RPC API that provides complete access and control of the CMON Controller which is its workflow engine. The ClusterCOntrol CMON API documentation is readily available *here*.

Using the CMON RPCv2 API directly though requires that clients are passing JSON documents of jobs specs to be executed. Creating these CMON JSON job specs are straightforward however cumbersome given its text based nature where correct syntax and manual validations is needed in order for the jobs to be executed.

A more user (developer) friendly way of using the CMON RPCv2 API is to use the OpenAPI specification to document and generate API client libraries for various system languages. The OpenAPI documentation is machine readable and makes it easy formally describe an API and also use tools such as the openapi-generator to generate client libraries.

You can use your favourite programming language such as Go or Python to create applications which integrates with ClusterControl to manage and monitor your databases.

This project is hosted in Github called clustercontrol-client-sdk.

### 4.14.1 SDK for various programming languages

ClusterControl API (v2) SDK has bindings for various programming languages.

#### API binding for Java

API binding for Java is available in the `java` subdirectory. It requires Apache Maven to build. Here are the build instructions.

Setting up Apache Maven Here are instructions on how to setup Maven.

1. Download the latest maven from here and extract it to your machine.
2. Install JDK 8 and set `JAVA_HOME` to the directory where JDK is installed.

```
export JAVA_HOME=<jdk-install-dir>
```

3. Update `PATH` environment variable to include the Maven and JDK `bin` directories.

```
export PATH=<maven-bin-dir>:$JAVA_HOME/bin:$PATH
```

4. Building Java bindings:

```
git clone https://github.com/severalnines/clustercontrol-client-sdk.git
cd clustercontrol-client-sdk/java
mvn -DskipTests clean package
```

We also provide an API Abstraction Layer above the underlying low-level API bindings. This makes it super easy to use the API SDK. See the instructions here.

#### API binding for Python

API binding for Python is available in the `python` subdirectory.

We hope to provide in the very near future an API Abstraction Layer above the underlying low-level API bindings. This makes it super easy to use the API SDK. See the instructions here.

#### API binding for GoLang

API binding for GoLang is available in the `go` subdirectory.

We hope to provide in the very near future an API Abstraction Layer above the underlying low-level API bindings. This makes it super easy to use the API SDK. See the instructions here.

### 4.14.2 Generating Sources from the API definition

**Note**

Note It is not required for you to generate new sources from the API definition. However, if you happend to modify the API definition, then you will have to generate new sources from the definition in order for the sources to reflect changes to the definition.

1. Install Java Runtime 8. Skip these steps if you already have the latest generated language bindings:
   - Follow the instructions for Linux.

- Follow the instructions for MacOS.

2. Download and setup the openapi-generator:

```
$ cd clustercontrol-client-sdk
$ mkdir -p $PWD/bin/openapitools
$ curl -o $PWD/bin/openapitools/openapi-generator-cli.sh␣
↪https://raw.githubusercontent.com/OpenAPITools/openapi-generator/master/bin/utils/
$ chmod +x $PWD/bin/openapitools/openapi-generator-cli.sh
$ export PATH=$PATH:$PWD/bin/openapitools
```

3. Install Apache Maven. Follow the instructions here for your favourite OS:

```
$ curl -o apache-maven-3.8.6-bin.tar.gz␣
↪https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/apache-maven-3.8.6-bin.tar.g
$ tar xzvf apache-maven-3.8.6-bin.tar.gz
$ export PATH=apache-maven-3.8.6/bin:$PATH
```

4. Check if `openapi-generator.sh` is setup properly:

```
$ openapi-generator.sh version
6.0.1
```

5. Generate API language bindings:

```
# install make with 'apt install make'
#make python
#make java
make go
...
[main] INFO  o.o.codegen.TemplateManager - writing file /
↪root/clustercontrol-client-sdk/./generated/go/.openapi-generator/FILES
################################################################################
↪#########
# Thanks for using OpenAPI Generator.                                          ␣
↪          #
# Please consider donation to help us maintain this project                    ␣
↪       #
# https://opencollective.com/openapi_generator/donate                          ␣
↪          #
################################################################################
↪#########
```

The next step is to write your first applicaton which calls upon these generated client API stubs. Checkout the respective binding subdiirectories for `java`, `python` and `go`.

The GitHub repository contains examples of deploying database clusters of various types (MySQL/MariaDB replication or galera with ProxySQL, PostgreSQL replication, MongoDB replica set and/or sharded, Redis sentinel, Microsoft SQL server, and Elasticsearch)

Navigate to the repository docs folder for generated documentation on the terraform provider plugin for ClusterControl.

The sub-folders contain examples of the following:

| Database type | Description |
|---|---|
| MySQL/MariaDB | MySQL and/or MariaDB database (both Master/Slave and Galera multi-master) |
| ProxySQL | ProxySQL load balancer with MySQL/MariaDB database clusters |
| PostgreSQL | Postgres (Primary with Hot-Standby clusters) |
| MongoDB | Both sharded clusters and single Replicaset clusters |
| Redis | Redis sentinel clusters |
| Microsoft SQL Server | Both standalone and hot-standby cluster with one hot-standby (async) |
| Elasticsearch | Elasticsearch clusters |
| ClickHouse | ClickHouse clusters |

## 4.15 Provider Configuration

Use the following to configure the Terraform provider for ClusterControl:

```
terraform {
  required_providers {
    clustercontrol = {
      source = "severalnines/clustercontrol"
      version = "0.2.15"
    }
  }
}

provider "clustercontrol" {
  # Configuration options:
  # cc_api_url = ""
  # cc_api_user = ""
  # cc_api_user_password = ""
}
```

## 4.16 Resources

| Name |
|---|
| *clustercontrol_db_cluster* |
| *clustercontrol_db_cluster_backup* |
| *clustercontrol_db_cluster_backup_schedule* |
| *clustercontrol_db_cluster_maintenance* |

### 4.16.1 clustercontrol_db_cluster

#### Required

- `db_admin_user_password` (String, Sensitive) Password for the admin/root user for the database. Note that this may show up in logs, and it will be stored in the state file
- `db_cluster_name` (String) The name of the database cluster.

- db_cluster_type (String) Type of cluster – replication, galera, postgresql_single (single is a misnomer), etc
- db_vendor (String) Database vendor – oracle, percona, mariadb, 10gen, microsoft, redis, elasticsearch, for postgresql it is defaultetc
- db_version (String) The database version
- ssh_key_file (String) SSH Key file. The path to the private key file for the Sudo user on the ClusterControl host
- ssh_user (String) The SSH user ClusterControl will use to SSH to the DB host from the ClusterControl host

### Optional

- db_admin_username (String) Name for the admin/root user for the database
- db_auto_recovery (Boolean) Have cluster auto-recovery on (or off)
- db_cluster_create (Boolean) Whether to create this resource or not?
- db_cluster_import (Boolean) Whether to import this resource or not?
- db_config_server (Block List) Specification for the MongoDB Configuration Server. (see *below for nested schema*)
- db_data_directory (String) The data directory for the database data files. If not set explicily, the default for the respective DB vendor will be chosen
- db_deploy_agents (Boolean) Automatically deploy prometheus and other relevant agents after setting up the intial DB cluster.
- db_elasticsearch_http_port (String) The port on which Elasticsearch will accept client http connections
- db_elasticsearch_transfer_port (String) The port on which Elasticsearch will accept client connections for data transfer(?)
- db_enable_pbm_agent (Boolean) Enable percona backup for mongodb.
- db_enable_pgbackrest_agent (Boolean) Enable PgBackRest for Postgres.
- db_enable_ssl (Boolean) Enable SSL based comms between the cluster nodes and client access to node.
- db_enable_timescale (Boolean) Whether to setup TimescaleDB extension or not
- db_enable_uninstall (Boolean) When removing DB cluster from ClusterControl, enable uinstalling DB packages.
- db_host (Block List) The list of nodes/hosts that make up the cluster (see *below for nested schema*)
- db_install_software (Boolean) Install DB packages from respective repos
- db_load_balancer (Block List) The list of nodes/hosts that make up the cluster (see *below for nested schema*)
- db_mongo_auth_db (String) The mongodb database to use for authentication purposes
- db_mongo_config_server_port (String) The port on which MongoDB config server will accept client connections. MongoS server will use same port number as db_mongo_port

- `db_mongo_port` (String) The port on which MongoDB will accept client connections
- `db_mongos_server` (Block List) Specification for the MongoDB mongos Server. (see *below for nested schema*)
- `db_mssqlserver_port` (String) The port on which MSSQL will accept client connections
- `db_mysql_port` (String) The port on which MySQL will accept client connections
- `db_pbm_backup_dir` (String) Backup dir, nfs mounted directory/path for PBM backup.
- `db_postgres_port` (String) The port on which PostgreSql will accept client connections
- `db_redis_port` (String) The port on which Redis will accept client connections
- `db_replica_set` (Block List) The hosts that make up the cluster. (see *below for nested schema*)
- `db_semi_sync_replication` (Boolean) Semi-synchronous replication for MySQL and MariaDB non-galera clusters
- `db_sentinel_port` (String) The port Redis Sentinel uses to communicate
- `db_snapshot_host` (String) Elasticsearch snapshot host
- `db_snapshot_location` (String) Elasticsearch snapshot location
- `db_snapshot_repository` (String) Elasticsearch snapshot repository
- `db_tags` (Set of String) Tags to associate with a DB cluster. The tags are only relevant in the ClusterControl domain.
- `db_topology` (Block List) Only applicable to MySQL/MariaDB non-galera clusters. A way to specify Master and Slave(s). (see *below for nested schema*)
- `disable_firewall` (Boolean) Disable firewall on the host OS when installing DB packages.
- `disable_selinux` (Boolean) Disable SELinux on the host OS when installing DB packages.
- `ssh_port` (String) The ssh port.
- `ssh_user_password` (String, Sensitive) Sudo user's password. If sudo user doesn't have a password, leave this field blank
- `db_clickhouse_native_port` (String) The port for ClickHouse internal TLS communication. Non-TLS is not supported
- `db_clickhouse_keeper_port` (String) The port for ClickHouse Keeper node/role communication

### Read-Only

- `db_cluster_id` (String) TODO
- `id` (String) The ID of this resource.
- `last_updated` (String) TODO

### Nested Schema for `db_config_server`

Required:

- `member` (Block List, Min: 1) The host that make up the replicaset member. (see *below for nested schema*)

- `rs` (String) The replicaset's name.

### Nested Schema for `db_config_server.member`

Required:

- `hostname` (String) Hostname of the DB host. Can be IP address as well.

Optional:

- `hostname_data` (String) Hostname/IP used for data comms (may be legacy ClusterControl).
- `hostname_internal` (String) If there's a private net that all DB hosts can communicate, use it here.

### Nested Schema for `db_host`

Required:

- `hostname` (String) Hostname of the DB host. Can be IP address as well

Optional:

- `datadir` (String) The data directory for the database data files. If not set explicily, default for the DB type will be used, or inherited from earlier/top-level specification.
- `hostname_data` (String) Hostname/IP used for data comms (may be legacy ClusterControl).
- `hostname_internal` (String) If there's a private net that all DB hosts can communicate, use it here.
- `protocol` (String) TODO.
- `roles` (String) Applicable to Elasticsearch and ClickHouse - {Elasticsearch: master-data: host will be designated as the master node and a data node; ClickHouse: replica, keeper)
- `sync_replication` (Boolean) Applicable to PostgreSQL hot-standby nodes only. Use synchronous replication (or not)

### Nested Schema for `db_load_balancer`

Required:

- `db_lb_admin_user_password` (String, Sensitive) The load balancer admin user's password
- `db_lb_monitor_user_password` (String, Sensitive) The load balancer monitor user's password
- `db_lb_type` (String) The load balancer type (e.g., proxysql, haproxy, etc)
- `db_lb_version` (String) Software version
- `ssh_user` (String) The SSH user ClusterControl will use to SSH to the DB host from the ClusterControl host

Optional:

- `db_lb_admin_port` (String) The load balancer admin port that will be used to administer it.
- `db_lb_admin_username` (String) The load balancer admin user
- `db_lb_enable_uninstall` (Boolean) When removing DB cluster from ClusterControl, enable uinstalling DB packages.

- `db_lb_install_software` (Boolean) Install DB packages from respective repos
- `db_lb_monitor_username` (String) The load balancer monitor user (only applicable to proxysql)
- `db_lb_port` (String) The load balancer port that it will accept connections on behalf of the database it is front-ending.
- `db_lb_use_clustering` (Boolean) Whether to use ProxySQL clustering or not. Only applicable to ProxySQL at this time
- `db_lb_use_rw_splitting` (Boolean) Whether to Read/Write splitting for queries or not?
- `db_my_host` (Block List) The load balancer host in question (i.e, self) (see *below for nested schema*)
- `disable_firewall` (Boolean) Disable firewall on the host OS when installing DB packages.
- `disable_selinux` (Boolean) Disable SELinux on the host OS when installing DB packages.
- `ssh_key_file` (String) SSH Key file. The path to the private key file for the Sudo user on the ClusterControl host.
- `ssh_port` (String) The ssh port.
- `ssh_user_password` (String, Sensitive) Sudo user's password. If sudo user doesn't have a password, leave this field blank

### Nested Schema for `db_load_balancer.db_my_host`

Required:

- `hostname` (String) Hostname/IP of this load balancer. Can be IP address as well.

### Nested Schema for `db_mongos_server`

Required:

- `hostname` (String) Hostname of the DB host. Can be IP address as well.

Optional:

- `hostname_data` (String) Hostname/IP used for data comms (may be legacy ClusterControl).
- `hostname_internal` (String) If there's a private net that all DB hosts can communicate, use it here.

### Nested Schema for `db_replica_set`

Required:

- `member` (Block List, Min: 1) The hosts that make up the replicaset HA nodes. (see *below for nested schema*)
- `rs` (String) The replicaset's name.

### Nested Schema for `db_replica_set.member`

Required:

- `hostname` (String) Hostname of the DB host. Can be IP address as well

Optional:

- `arbiter_only` (Boolean) The host is acting as an arbiter only.
- `hidden` (Boolean) TODO.
- `hostname_data` (String) Hostname/IP used for data comms (may be legacy ClusterControl).
- `hostname_internal` (String) If there's a private net that all DB hosts can communicate, use it here.
- `priority` (Number) Priority of the host in the mongo replication setup.
- `slave_delay` (String) Used in non-galera MySQL/MariaDB standby setup. Specifies the lag for the slave.

#### Nested Schema for `db_topology`

Optional:

- `primary` (String) The Master host
- `replica` (String) The Slave host.

### 4.16.2 clustercontrol_db_cluster_backup

#### Required

- `db_backup_method` (String) mariabackup, xtrabackup, …
- `db_cluster_id` (String) The database cluster ID for which this LB is being deployed to.

#### Optional

- `db_backup_compression` (Boolean) Whether to compress backups or not
- `db_backup_compression_level` (Number) Compression level
- `db_backup_dir` (String) Base direcory where backups is to be stored
- `db_backup_encrypt` (Boolean) Whether to encrypt or not
- `db_backup_failover_host` (String) If backup failover is enabled, which host to use as backup host in the event of failure of first choice host.
- `db_backup_host` (String) Where there are multiple hosts, which host to choose to create backup from.
- `db_backup_retention` (Number) Backup retention period in days
- `db_backup_storage_controller` (Boolean) Whether to store the backup on CMON controller host or not
- `db_backup_storage_host` (String) Which host to store the backup on. Typically, used with mongodump backup method.
- `db_backup_subdir` (String) Sub-dir for backups – default: "BACKUP-%I"
- `db_backup_system_db` (Boolean) Whether to enable backup failover to another host in case the host crashes
- `db_enable_backup_failover` (Boolean) Whether to enable backup failover to another host in case the host crashes

#### Read-Only

- `db_resource_id` (String) TODO
- `id` (String) The ID of this resource.
- `last_updated` (String) TODO

### 4.16.3 clustercontrol_db_cluster_backup_schedule

#### Required

- `db_backup_method` (String) mariabackup, xtrabackup, …
- `db_backup_sched_time` (String) The time to kick off a backup (e.g. 'TZ=UTC 0 0 * * *')
- `db_cluster_id` (String) The database cluster ID for which this LB is being deployed to.

#### Optional

- `db_backup_compression` (Boolean) Whether to compress backups or not
- `db_backup_compression_level` (Number) Compression level
- `db_backup_dir` (String) Base direcory where backups is to be stored
- `db_backup_encrypt` (Boolean) Whether to encrypt or not
- `db_backup_failover_host` (String) If backup failover is enabled, which host to use as backup host in the event of failure of first choice host.
- `db_backup_host` (String) Where there are multiple hosts, which host to choose to create backup from.
- `db_backup_retention` (Number) Backup retention period in days
- `db_backup_sched_title` (String) A title for the backup schedule (e.g., Daily full, Hourly incremental, etc)
- `db_backup_storage_controller` (Boolean) Whether to store the backup on CMON controller host or not
- `db_backup_storage_host` (String) Which host to store the backup on. Typically, used with mongodump backup method.
- `db_backup_subdir` (String) Sub-dir for backups – default: "BACKUP-%I"
- `db_backup_system_db` (Boolean) Whether to enable backup failover to another host in case the host crashes
- `db_enable_backup_failover` (Boolean) Whether to enable backup failover to another host in case the host crashes

#### Read-Only

- `db_resource_id` (String) TODO
- `id` (String) The ID of this resource.
- `last_updated` (String) TODO

### 4.16.4 clustercontrol_db_cluster_maintenance

#### Required

- `db_cluster_id` (String) The database cluster ID for which this LB is being deployed to.
- `db_maint_start_time` (String) Format: `Jan-02-2006T15:04`
- `db_maint_stop_time` (String) Format: `Jan-02-2006T15:04`

#### Optional

- `db_maint_reason` (String) TODO

#### Read-Only

- `db_resource_id` (String) TODO
- `id` (String) The ID of this resource.
- `last_updated` (String) TODO

## 4.17 Common fields in the resource definition

### 4.17.1 Resource – clustercontrol_db_cluster

`db_host`

The `db_host` block inside the `clsutercontrol_db_cluster` resource specifies the hosts that make up the cluster. Each host that makes up the DB cluster should have one of these blocks. The mandatory attribute for each `db_host` block is the **hostname**.

Example:

```
resource "clustercontrol_db_cluster" "this" {
    ...
    db_host {
        hostname = "host-1"
    }
    db_host {
        hostname = "host-2"
    }
    ...

}
```

### 4.17.2 Scheduling Backups using the clustercontrol_db_cluster_backup_scheduler resource

The backup schedule resource allows you to create a backup schedule for a cluster in ClusterControl through the terraform provider. Here's an example of a daily full backup schedule using `xtrabackup`. As can be seen the `clustercontrol_db_cluster_backup_schedule`resource depends on the `clustercontrol_db_cluster` resource.

```
resource "clustercontrol_db_cluster_backup_schedule" "full-1" {
  depends_on                    = [clustercontrol_db_cluster.this]
  db_backup_sched_title         = "Daily full"
  db_backup_sched_time          = "TZ=UTC 0 0 * * *"
  db_cluster_id                 = clustercontrol_db_cluster.this.id
  db_backup_method              = "xtrabackupfull"
  db_backup_dir                 = var.db_backup_dir
  db_backup_subdir              = var.db_backup_subdir
  db_backup_encrypt             = var.db_backup_encrypt
  db_backup_host                = var.db_backup_host
  db_backup_storage_controller  = var.db_backup_storage_controller
  db_backup_compression         = var.db_backup_compression
  db_backup_compression_level   = var.db_backup_compression_level
  db_backup_retention           = var.db_backup_retention
}
```

### 4.17.3 Taking adhoc backups using the clustercontrol_db_cluster_backup resource

You can a maintenance window for a cluster using the `clustercontrol_db_cluster_backup` resource. Here's an example of a full backup using `xtrabackup`.

```
resource "clustercontrol_db_cluster_backup" "full-1" {
  depends_on                    = [clustercontrol_db_cluster.this]
  db_cluster_id                 = clustercontrol_db_cluster.this.id
  db_backup_method              = "xtrabackupfull"
  db_backup_dir                 = var.db_backup_dir
  db_backup_subdir              = var.db_backup_subdir
  db_backup_encrypt             = var.db_backup_encrypt
  db_backup_host                = var.db_backup_host
  db_backup_storage_controller  = var.db_backup_storage_controller
  db_backup_compression         = var.db_backup_compression
  db_backup_compression_level   = var.db_backup_compression_level
  db_backup_retention           = var.db_backup_retention
}
```

### 4.17.4 Setting a maintenance window using the clustercontrol_db_cluster_maintenance resource

You can take adhoc backups (full or incremental) of a cluster using the `clustercontrol_db_cluster_backup` resource. Here's an example of a full backup using `xtrabackup`.

```
resource "clustercontrol_db_cluster_maintenance" "server-upgrade-03312024"
↪{
  depends_on = [clustercontrol_db_cluster.this]
  db_cluster_id       = clustercontrol_db_cluster.this.id
  db_maint_start_time = "Mar-31-2024T00:00"
  db_maint_stop_time  = "Mar-31-2024T23:30"
  db_maint_reason     = "Hardware refresh March 31, 2024"
}
```

**Note**

Note The `db_maint_start_time` and `db_maint_stop_time` should be specified in local time (without the timezone).

## 4.18 Supported backup methods for the respective database types (and vendors)

The following types are supported:

| Database type | Vendor | Backup method |
|---|---|---|
| MySQL | Oracle, Percona | xtrabackupfull, xtrabackupincr, mysqldump |
| MariaDB | MariaDB | mariabackupfull, mariabackupincr, mysqldump |
| PostgreSQL | Community | pg_basebackup, pgdumpall, pgbackrest(full,incr,diff) |
| MongoDB | MongoDB | mongodump, percona-backup-mongodb |
| Redis | Redis | Use the value “” to indicate (aof – default redis) |
| SQL Server | Microsoft | mssql_full |
| Elasticsearch | Elastic | TBD – default snapshot |
| ClickHouse | ClickHouse | clickhouse-native, clickhouse-native-incr |

## 4.19 Toggling cluster auto-recovery option

You can toggle the cluster-auto-recovery feature in ClusterControl using the `db_auto_recovery` field of the `clustercontrol_db_cluster`resource.

```
resource "clustercontrol_db_cluster" "this" {
...
  db_auto_recovery          = true
...
}
```

cmon_exporter is a tool intended to extract and expose metrics collected by ClusterControl. This project is hosted in Github, cmon_exporter.

## 4.20 Build

```
make
```

## 4.21 Create a CMON user for the cmon_exporter

```
s9s user \
    --create \
    --group=admins \
    --generate-key \
    --controller=https://127.0.0.1:9501 \
```

(continues on next page)

(continued from previous page)

```
    --new-password="SECRET" \
    --email-address="admin@example.com" \
    cmon_exporter
```

## 4.22 How to run

### 4.22.1 Running in foreground

```
CMON_USERNAME=cmon_exporter CMON_PASSWORD=SECRET␣
↪CMON_ENDPOINT=https://127.0.0.1:9501 ./cmon_exporter
```

### 4.22.2 Systemd

1. Build or download a released version from here.
2. Copy the `cmon_exporter` binary to `/usr/local/bin`
3. Download cmon_exporter.service and place it at `/etc/systemd/system/cmon_exporter.service`.
4. Change `USER` and `SECRET` to a ClusterControl (cmon user) that has access to all database clusters.
5. Enable and start:

```
systemctl enable cmon_exporter
systemctl restart cmon_exporter
systemctl status cmon_exporter
```

### 4.22.3 Docker

```
sudo docker run --net=host -it  -e CMON_USERNAME=cmon_exporter -e␣
↪CMON_PASSWORD=SECRET -e CMON_ENDPOINT=https://127.0.0.1:9501 ␣
↪severalnines/cmon_exporter
```

## 4.23 Access the metrics

```
curl 127.0.0.1:9954/metrics
```

## 4.24 Current metrics

```
# HELP cmon_alarms_backup_failed_total Total number of failed backups␣
↪alarms.
# TYPE cmon_alarms_backup_failed_total counter
cmon_alarms_backup_failed_total{ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_alarms_critical_total Total number of clusters in unknown state.
# TYPE cmon_alarms_critical_total counter
cmon_alarms_critical_total{ControllerId=
```

(continues on next page)

(continued from previous page)

```
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_backup_failed Is there an active backup failed alarm on␣
↪the cluster.
# TYPE cmon_cluster_backup_failed counter
cmon_cluster_backup_failed{ClusterID="397",ClusterName=
↪"251635de-08f5-4099-8cf0-6637c4fbeddc",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
cmon_cluster_backup_failed{ClusterID="402",ClusterName=
↪"d34fe2fd-17a4-4c1f-8cb9-730331c305bd",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_backup_upload_failed Is there an active backup failed␣
↪alarm on the cluster.
# TYPE cmon_cluster_backup_upload_failed counter
cmon_cluster_backup_upload_failed{ClusterID="397",ClusterName=
↪"251635de-08f5-4099-8cf0-6637c4fbeddc",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
cmon_cluster_backup_upload_failed{ClusterID="402",ClusterName=
↪"d34fe2fd-17a4-4c1f-8cb9-730331c305bd",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_degraded Is the cluster up (DEGRADED) or not.
# TYPE cmon_cluster_degraded counter
cmon_cluster_degraded{ClusterID="397",ClusterName=
↪"251635de-08f5-4099-8cf0-6637c4fbeddc",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_degraded_total Total number of clusters in degraded␣
↪state.
# TYPE cmon_cluster_degraded_total counter
cmon_cluster_degraded_total{ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_failed_init Cluster failed to initialize.
# TYPE cmon_cluster_failed_init counter
cmon_cluster_failed_init{ClusterID="397",ClusterName=
↪"251635de-08f5-4099-8cf0-6637c4fbeddc",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
cmon_cluster_failed_init{ClusterID="402",ClusterName=
↪"d34fe2fd-17a4-4c1f-8cb9-730331c305bd",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_failed_init_total Total number of clusters that failed␣
↪to initialize..
# TYPE cmon_cluster_failed_init_total counter
cmon_cluster_failed_init_total{ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_failed_total Total number of clusters in failed state.
# TYPE cmon_cluster_failed_total counter
cmon_cluster_failed_total{ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 1
# HELP cmon_cluster_failure Is the cluster up (FAILURE) or not.
# TYPE cmon_cluster_failure counter
cmon_cluster_failure{ClusterID="397",ClusterName=
```

(continues on next page)

(continued from previous page)

```
↪"251635de-08f5-4099-8cf0-6637c4fbeddc",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
cmon_cluster_failure{ClusterID="402",ClusterName=
↪"d34fe2fd-17a4-4c1f-8cb9-730331c305bd",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 1
# HELP cmon_cluster_started_total Total number of clusters in started state.
# TYPE cmon_cluster_started_total counter
cmon_cluster_started_total{ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 1
# HELP cmon_cluster_stopped_total Total number of clusters in stopped state.
# TYPE cmon_cluster_stopped_total counter
cmon_cluster_stopped_total{ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_total Total number of clusters.
# TYPE cmon_cluster_total counter
cmon_cluster_total{ControllerId="00000000-0000-0000-0000-000000000000"} 2
# HELP cmon_cluster_unknown_total Total number of clusters in unknown state.
# TYPE cmon_cluster_unknown_total counter
cmon_cluster_unknown_total{ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_cluster_up Is the cluster up (STARTED) or not.
# TYPE cmon_cluster_up counter
cmon_cluster_up{ClusterID="397",ClusterName=
↪"251635de-08f5-4099-8cf0-6637c4fbeddc",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 1
cmon_cluster_up{ClusterID="402",ClusterName=
↪"d34fe2fd-17a4-4c1f-8cb9-730331c305bd",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 0
# HELP cmon_up Was the last  CMON query successful.
# TYPE cmon_up gauge
cmon_up{CmonVersion="1.9.8.7039",ControllerId=
↪"00000000-0000-0000-0000-000000000000"} 1
```

The `cmon_sd` service simplifies Prometheus target discovery for all nodes managed by ClusterControl by exposing an HTTP endpoint. This feature is particularly useful for external Prometheus instances not managed by ClusterControl. The service works by monitoring ClusterControl's internal objects and converting them into readily discoverable Prometheus targets.

This project is publicly hosted in Github, cmon_sd.

**Read more**

- HTTP-based Service Discovery Configuration for Prometheus

## 4.25 Install

Navigate to `/usr/local/bin` directory, download and extract the latest binary from here. At the moment, only amd64/x86-64/x64 architecture is supported:

```
cd /usr/local/bin
wget␣
↪https://github.com/severalnines/cmon-sd/releases/download/v0.0.8/cmon_sd_0.0.8_linux_am
tar -xzf cmon_sd_*_linux_amd64.tar.gz
rm -Rf cmon_sd_*_linux_amd64.tar.gz     # clean up
```

## 4.26 Options

| Name, shorthand | Description |
|---|---|
| `--port` | Listening port, defaults to 8080. |

## 4.27 Usage

1. Create a ClusterControl control user that can log in into ClusterControl. The user must have a global view of all clusters and can be a read-only user. In this example, we will create a user called `cmonsd` which belongs to a global read-only group called `global-ro`:

### ClusterControl GUI

1. To create a team, go to ***ClusterControl GUI → User management → Create user or team → Create team***.
2. In the **Team name** field, specify the group name: `global-ro`. Click **Continue**.
3. Under the **Users** section, skip adding a user first as we will add it later. Click **Continue**.
4. Under the **Permissions** section, toggle off the following:
   - Change controller configuration
   - Change LDAP settings
   - Deploy clusters
5. Under the **Clusters permission level** dropdown, choose "View". This only allows users in this group to have read-only permission for all clusters. Click **Continue** and **Finish**.
6. To create a user, go to ***ClusterControl GUI → User management → Create user or team → Create user***.
7. Under the **Details** section, specify the all necessary information. For username, we can specify "cmonsd". Toggle off **Force change password** and click **Continue**.
8. Under the **Team** section, choose "global-ro" from the dropdown. Click **Continue** and **Finish** to create the user.

2. Pass `CMON_USERNAME` and `CMON_PASSWORD` environment variables to `cmon_sd` binary:

```
CMON_USERNAME='cmonsd' CMON_PASSWORD='SuperSecret$$$P455' ./cmon_sd -p␣
↪9909
```

3. Verify the HTTP response:

```
$ curl http://127.0.0.1:9909
[
    {
        "targets": [
            "10.10.10.17:9100",
            "10.10.10.17:9011",
            "10.10.10.17:9104",
            "10.10.10.16:9100",
            "10.10.10.16:9011",
            "10.10.10.16:9104",
            "10.10.10.18:9100",
            "10.10.10.18:9011",
            "10.10.10.18:9104"
        ],
        "labels": {
            "ClusterID": "641",
            "ClusterName": "PXC57",
            "ClusterType": "GALERA",
            "cid": "641"
        }
    }
]
```

4. Alternatively, you can configure it as a systemd unit. See *Run as systemd*.

## 4.28 Run as systemd

1. To create a systemd unit file for `cmon_sd` service, save the following content to `/etc/systemd/system/cmon_sd.service`:

```
[Unit]
Description=CMON Service Discovery
# Ensures cmon starts before this service
After=cmon.service
# Makes this service dependent on cmon
Requires=cmon.service

[Service]
Type=simple
ExecStart=/usr/local/bin/cmon_sd -p 9909
WorkingDirectory=/usr/local/bin/

# Environment variables
Environment="CMON_USERNAME=cmonsd"
Environment="CMON_PASSWORD=SuperSecret$$$P455"

# Security and Reliability
Restart=on-failure

[Install]
```

(continues on next page)

(continued from previous page)

```
WantedBy=multi-user.target
```

2. Apply the configuration by running:

```
sudo systemctl daemon-reload
```

3. Start the service immediately and enable it to run on boot:

```
sudo systemctl start cmon_sd
sudo systemctl enable cmon_sd
```

## 4.29 Sample prometheus.yml

On your Prometheus instance, configure the `http_sd_configs` accordingly:

```
---
alerting:
  alertmanagers:
  - static_configs:
    - targets:
global:
  evaluation_interval: 10s
  external_labels:
    monitor: clustercontrol
  scrape_interval: 10s
  scrape_timeout: 10s
rule_files:
scrape_configs:
- http_sd_configs:
  - url: http://10.10.10.1:9909
  job_name: clustercontrol
...
```

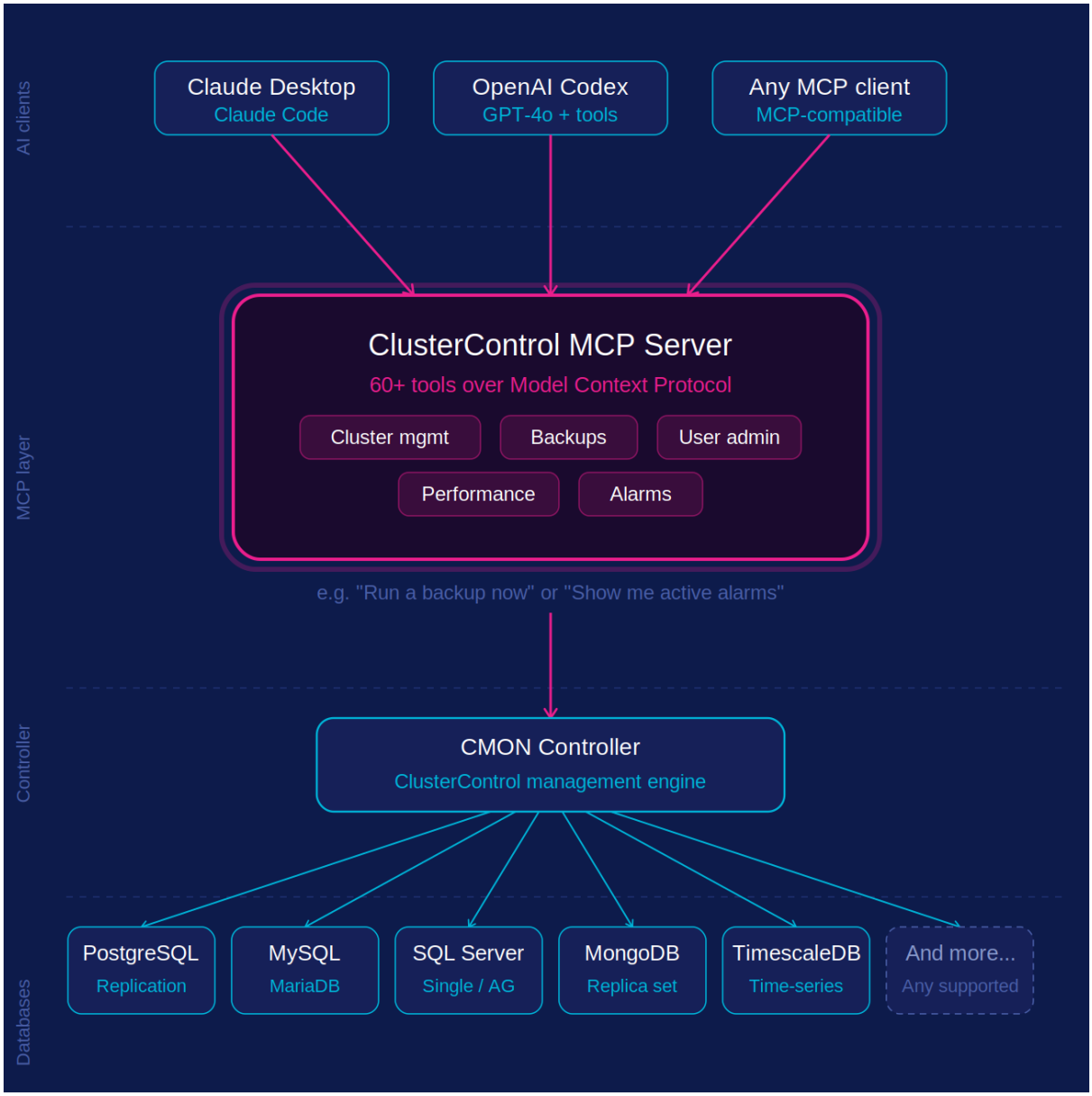
Restart Prometheus to apply the change. The Prometheus should start scraping all targets managed by ClusterControl.

## 4.30 ClusterControl MCP

The ClusterControl MCP Server is a bridge between your AI assistant and your databases. You type what you need in plain English or other languages, and it figures out the rest. No dashboards to click through, no commands to memorize. It works with AI tools like Claude Desktop, Claude Code, and OpenAI Codex, and supports 72 actions and counting, covering backups, cluster management, user accounts, and performance monitoring. It works with MySQL, MariaDB, PostgreSQL, MongoDB, SQL Server, and TimescaleDB.

### 4.30.1 How It Works

The diagram below shows how everything connects. Your AI tool sits at the top. When you type a request, it goes through the MCP Server, which translates it into the right action and passes it to the CMON Controller. The controller then talks to whichever database cluster you are working with.




Before you get started, make sure you have the following in place:

- A running ClusterControl installation with CMON Controller accessible over the network (CMON controller reachable on port 9501)
- A CMON user account with access to the clusters you want to query
- An MCP-compatible AI client such as Claude Desktop, Claude Code, or OpenAI Codex

Once you have confirmed all the requirements above are in place, you are ready to install and configure the ClusterControl MCP Server. Packages are published to the Severalnines repository alongside other ClusterControl components.

### RHEL / Rocky / AlmaLinux

```
sudo yum install clustercontrol-mcp
```

#### Debian / Ubuntu

```
sudo apt-get install clustercontrol-mcp
```

The binary installs to `/usr/bin/cmon-mcp`. A systemd service unit and default configuration file are also installed:

- `/etc/systemd/system/cmon-mcp.service` : service unit
- `/etc/default/cmon-mcp` : environment file for credentials and transport options

### 4.30.2 Tool Annotations

Every tool the MCP Server exposes advertises the standard MCP annotations `readOnlyHint` and `destructiveHint` alongside its name and description. Of the 72 tools, 43 are pure reads, 26 change state on the CMON Controller, and 3 write files on the local filesystem of the MCP Server host; 22 tools carry `destructiveHint: true`. The annotations are generated at tool registration from a committed, human-reviewed manifest, so the hints a client sees cannot drift from what was reviewed.

MCP clients can use these annotations to auto-approve read-only tools and to require explicit confirmation before state-changing or destructive ones. Keep in mind that annotations are advisory metadata for the client: the server does not enforce them, and the dry-run preview for write operations is likewise a client-side safeguard, not server-side enforcement.

If you need the server itself to refuse an action rather than rely on the client, see *Access Control* below.

### 4.30.3 Configuration

The ClusterControl MCP Server is configured through the `/etc/default/cmon-mcp` file on your CMON host. Open the file and edit the values to match your environment:

```
sudo vi /etc/default/cmon-mcp
```

#### Configuration Parameters

#### CMON Controller Connection

| Parameter | Description | Example |
|---|---|---|
| `CMON_ENDPOINT` | The URL of your CMON Controller API | `https://127.0.0.1:9501` |
| `CMON_USERNAME` | The ClusterControl username | `admin` |
| `CMON_PASSWORD` | Password for the user (leave empty if using key auth) | `severalNINES999` |
| `CMON_KEY_FILE` | Path to RSA private key (alternative to password) | `/etc/clustercontrol/id_rsa` |

You can authenticate using either a password or an RSA key file, not both at the same time. If `CMON_PASSWORD` is set, leave `CMON_KEY_FILE` empty, and vice versa.

The CMON user you specify here is the identity the MCP Server uses for every action it performs. We recommend creating a dedicated CMON user with only the privileges your team actually needs and please avoid reusing your main admin account.

### MCP Server

| Parameter | Description | Example |
|---|---|---|
| `MCP_BIND_ADDRESS` | Address and port the MCP Server listens on | `0.0.0.0:3000` |
| `MCP_BASE_URL` | Public URL that AI clients use to connect | `http://your-cc-host:3000` |
| `MCP_AUTH_TOKEN` | Bearer token to secure all incoming connections | `your-strong-random-token` |

For `MCP_BIND_ADDRESS`, use `127.0.0.1:3000` if you only need local access, or `0.0.0.0:3000` to allow remote AI clients to connect.

**MCP Transport Endpoints**

The MCP Server exposes two transports on the same port: `/sse` and `/message` is SSE transport, used by Claude Desktop and Claude Code. `/mcp` is Streamable HTTP transport, used by OpenAI Codex

### Access Control

These settings are enforced by the **server**, not by the AI client. Unlike tool annotations and `dry_run` previews, a refusal here cannot be overridden by the client.

| Parameter | Description | Example |
|---|---|---|
| `MCP_READ_ONL` | Refuses all 29 state-changing tools for the lifetime of the process | `1` |
| `MCP_TOOL_ALL` | Comma-separated list of the only tools to expose | `list_clusters, list_alarms` |
| `MCP_TOOL_DEN` | Comma-separated list of tools to refuse; wins over `MCP_TOOL_ALLOW` | `create_job, restore_backup` |

Read-only mode both hides the write tools from the client's tool list *and* refuses a direct call on one, so a client that calls a tool it was never shown is still refused. Scoping works the same way, and the resource surfaces follow their tools automatically — a surface is served only when every tool it maps to is in scope.

**Warning**

A name that matches no tool stops the server

An unknown entry in either list is a startup error naming every offender, and the service exits non-zero. This is deliberate: a silently ignored deny entry would leave a tool you meant to remove fully exposed.

**Prefer `MCP_TOOL_ALLOW` when the requirement is *only these***

A deny list names what to remove, so a tool added in a later release is exposed by default. An allow list names what to keep, so a new tool is out of scope until you add it.

**Read the startup warnings.** Scoping is by tool *name*, and a tool name is not a data class. Several tools read the same data from the CMON Controller, so denying one can leave a sibling serving the same rows. The server names every such case at startup as `tool scope: WARNING: ...`. If you denied a tool for a security reason, that line tells you whether you actually removed the data.

One case deserves singling out: `create_job` takes a free-form `command`, which reaches CMON job commands that have no tool of their own and that no deny list can name — including `drop_cluster`, `remove_node` and `failover`. If `create_job` is in scope, treat the assistant as able to run any CMON job.

### Auditing

| Parameter | Description | Example |
|---|---|---|
| `MCP_AUDIT_LOG` | Path to the JSONL audit log; `off` disables auditing | `/var/log/cmon-mcp/audit.jsonl` |
| `MCP_DIGEST_HMAC_KEY_` | Key file the argument digests are computed under | `/etc/cmon-mcp/digest.key` |

Every tool call and resource read is recorded **twice** — an `intent` record before the action runs and a `completion` record after, correlated by `call_id`. Background tasks add a third, `acceptance`, correlated by `task_id`. Two records rather than one, because a single record written at the end cannot be told apart from an action that never happened, and the actions that matter most are the ones that did not return.

Raw arguments are never written. What is recorded is a keyed digest of them, so that someone who can read the log cannot work backwards to a low-entropy argument such as a password.

**A fresh install is audited by default; an upgrade changes nothing**

A first install provisions `/etc/cmon-mcp/digest.key` and `/etc/cmon-mcp/governed.env`, and that second file switches auditing on — so a newly installed host records from its first call. An **upgrade enables nothing**, because a host that has never been audited has no key, and a server that is told to audit without one refuses to start. To turn auditing off on a fresh install, set `MCP_AUDIT_LOG=off` in `/etc/default/cmon-mcp`, which is read after the package's file and wins.

**Warning**

Auditing is fail-closed

If the `intent` record cannot be written, the call is refused with `AUDIT_UNAVAILABLE` and the action never runs. This is the intended behaviour: an unauditable server does not act.

It also means the log's permissions matter. The server refuses any log file whose mode is wider than `0600`, on every record, so a rotation policy that recreated it world-readable would refuse every later call. The shipped `/etc/logrotate.d/cmon-mcp` already handles this correctly — do not loosen its `create 0600 root root` line.

To find failed actions you need **both** fields, because they mean different things:

```
jq 'select(.outcome=="error" or .tool_is_error==true)' /
↪var/log/cmon-mcp/audit.jsonl
```

`outcome` covers transport and task-level failure; `tool_is_error` covers a tool that ran and returned an error. The absence of `tool_is_error` is not evidence that the tool succeeded.

### File Exports

| Parameter | Description | Example |
|---|---|---|
| `MCP_EXPORT_DI` | The one directory local file writes are confined to; `off` refuses them all | `/srv/cc-exports` |

Three tools write to the MCP Server host's own filesystem: `download_error_report`, `export_cluster_log` and `unpack_error_report`. Each takes a destination path **from the client**, and the service runs as root, so without a boundary, granting an assistant `export_cluster_log` alone is not meaningfully different from granting it root write access to the host. Every path the client supplies is resolved inside this directory and may not leave it.

The default is `/var/lib/cmon-mcp/exports`, created at mode `0700` on start.

**Warning**

This can change behaviour on an existing host

The boundary is on by default, so a server whose configuration you never touched will begin refusing paths such as `output_path=/tmp/report.tar.gz` that previously worked. Point `MCP_EXPORT_DIR` at the directory those callers already use to keep their existing paths valid.

The boundary is only as strong as the directory's own permissions. If it is a symlink, group- or world-writable, or owned by another user, the server says so at startup as `export dir: WARNING: …` rather than refusing — a link onto a larger volume is a legitimate thing to want.

### Documentation Search

| Parameter | Description | Example |
|---|---|---|
| `MCP_DOCS_DI` | Installed ClusterControl documentation bundle used by `search_docs` | `/usr/share/cmon-mcp/docs` |

The package ships a documentation bundle and the `search_docs` tool answers from it locally, with no network calls, so it works in an air-gapped installation. The bundle is read once at startup; if it cannot be loaded, only `search_docs` is disabled and the reason is written to the journal, leaving every other tool working. Restart the service after repairing or upgrading it.

### Complete Configuration Example

The following shows the ClusterControl MCP Server example configuration after it has been set up through the /etc/default/cmon-mcp file on your CMON host.

```
# CMON Controller connection
CMON_ENDPOINT=https://127.0.0.1:9501
CMON_USERNAME=admin
CMON_PASSWORD='severalNINES999'
# RSA key auth alternative (leave CMON_PASSWORD empty if using this):
# CMON_KEY_FILE=/etc/clustercontrol/id_rsa

# MCP Server
MCP_BIND_ADDRESS=0.0.0.0:3000
MCP_BASE_URL=http://your-cc-host:3000
MCP_AUTH_TOKEN=

# Access control — enforced by the server, not the AI client.
# Start here if you are connecting an assistant to a production controller.
MCP_READ_ONLY=1
# MCP_TOOL_ALLOW=list_clusters,get_cluster,list_alarms,search_docs
# MCP_TOOL_DENY=create_job

# Auditing — set automatically on a fresh install; shown here for reference.
# MCP_AUDIT_LOG=/var/log/cmon-mcp/audit.jsonl
# MCP_DIGEST_HMAC_KEY_FILE=/etc/cmon-mcp/digest.key

# File exports — defaults to /var/lib/cmon-mcp/exports. Set this if callers
# already rely on a different directory.
# MCP_EXPORT_DIR=/srv/cc-exports
```

**Start read-only**

`MCP_READ_ONLY=1` is the setting to reach for first when pointing an assistant at a production controller. All 43 read tools keep working, so the assistant can still investigate, explain and report — it simply cannot change anything. Turn it off deliberately, once you know which write actions you actually want to allow.

### Generating a Secure Token

It is strongly recommended to protect your MCP Server with a bearer token, especially when `MCP_BIND_ADDRESS` is set to `0.0.0.0`. Generate one with:

```
openssl rand -hex 32
```

Copy the output and set it as the value of `MCP_AUTH_TOKEN`.

### Starting the Service

Save the configuration file, then restart the service:

```
systemctl restart cmon-mcp
```

Confirm it started successfully:

```
journalctl -u cmon-mcp -n 20
```

You should see output confirming the MCP Server is running and listening on the address defined in `MCP_BIND_ADDRESS`.

## 4.30.4 Connecting to AI Clients

Once the MCP Server is running, you need to register it with your AI client. The steps differ slightly depending on which client you use.

### Claude Desktop

Open your Claude Desktop configuration file:

- **macOS:** `~/Library/Application Support/Claude/claude_desktop_config.json`
- **Windows:** `%APPDATA%\Claude\claude_desktop_config.json`

Add the following entry under `mcpServers` so that it looks like this :

```
{
  "mcpServers": {
    "clustercontrol": {
      "type": "sse",
      "url": "http://your-cc-host:3000/sse",
      "headers": {
        "Authorization": "Bearer your-auth-token"
      }
    }
```

(continues on next page)

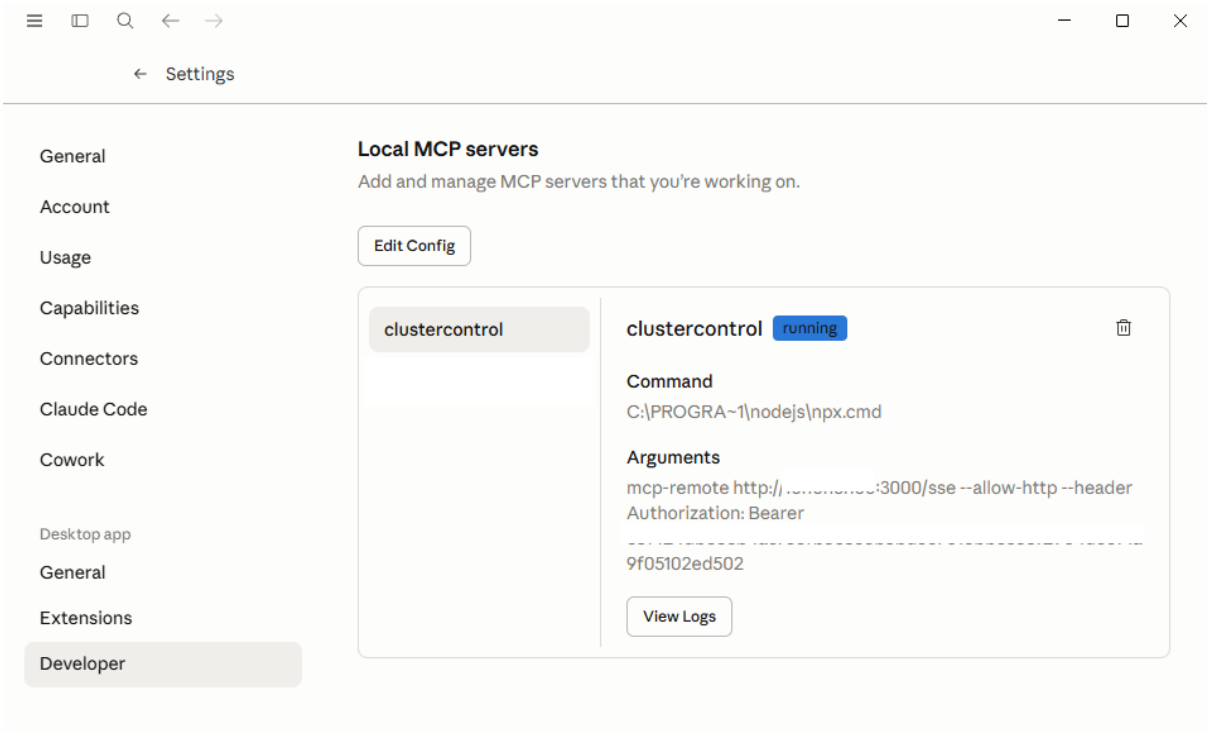
(continued from previous page)

```
  }
}
```

Alternatively, you can use mcp-remote with Node.js installed on your environment with the following JSON configuration:

```
{
  "mcpServers": {
    "clustercontrol": {
      "command": "npx",
      "args": [
        "mcp-remote",
        "http://your-cc-host:3000/sse",
        "--allow-http",
        "--header",
        "Authorization: Bearer your-auth-token"
      ]
    }
  }
}
```

Save the file and restart Claude Desktop. You should see clustercontrol appear in the File → Settings → developer → Local MCP servers panel like this after restart Claude Desktop.




## Claude Code

Run the following command in your terminal:

```
claude mcp add clustercontrol -- cmon-mcp \
  -endpoint https://your-cc-host:9501 \
  -username admin \
  -password your-password
```

To verify it was added:

```
claude mcp list
```

#### OpenAI Codex

Add the MCP Server to your Codex tool configuration using the `/mcp` endpoint:

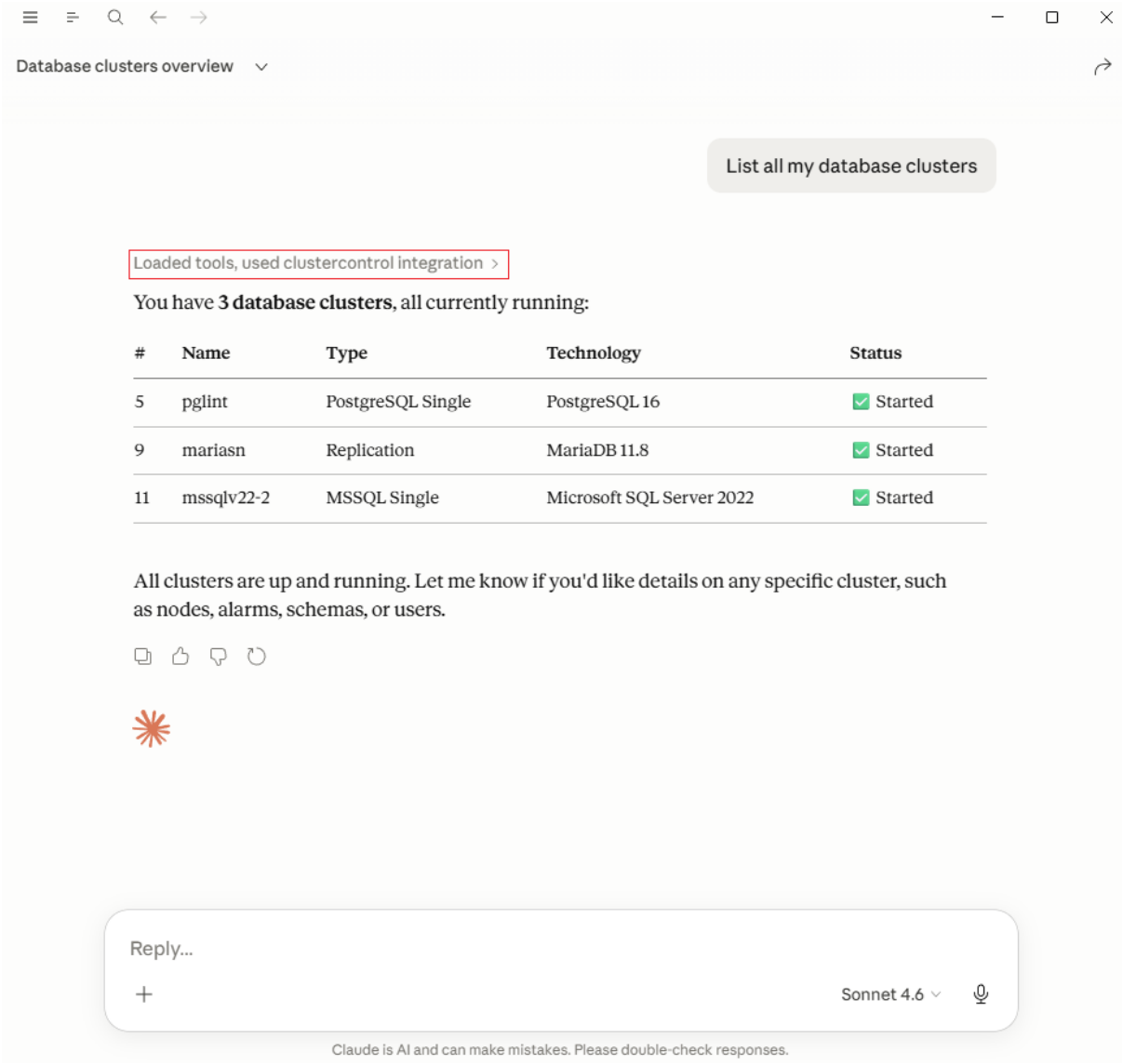
```
{
  "tools": [
    {
      "type": "mcp",
      "server_label": "clustercontrol",
      "server_url": "http://your-cc-host:3000/mcp",
      "headers": {
        "Authorization": "Bearer your-auth-token"
      },
      "require_approval": "never"
    }
  ]
}
```

### 4.30.5 Verifying the Connection

Once connected, ask your AI assistant the following to confirm everything is working:

```
List all my database clusters
```

If the setup is correct, it will return the list of clusters managed by your ClusterControl instance.


Database clusters overview
List all my database clusters
Loaded tools, used clustercontrol integration
You have 3 database clusters, all currently running:
# Name Type Technology Status
5 pglint PostgreSQL Single PostgreSQL 16 Started
9 mariasn Replication MariaDB 11.8 Started
11 mssqlv22-2 MSSQL Single Microsoft SQL Server 2022 Started
All clusters are up and running. Let me know if you'd like details on any specific cluster, such as nodes, alarms, schemas, or users.
Reply...
Sonnet 4.6
Claude is AI and can make mistakes. Please double-check responses.