MySQL Cluster Manager 9.6.0 User Manual

Abstract

This is the User Manual for the *MySQL Cluster Manager*, version 9.6.0. It documents the *MySQL Cluster Manager Agent* and *MySQL Cluster Manager Client* software applications, which can be used to administer *MySQL NDB Cluster*, a version of the *MySQL Database System* (referred to hereafter as “MySQL Server” or simply “MySQL”) that incorporates the `NDB` storage engine for high availability and data redundancy in a distributed computing environment.

This manual applies to MySQL Cluster Manager 9.6.0 and contains information that may not apply to older versions of the MySQL Cluster Manager software. For documentation covering previous MySQL Cluster Manager releases, see MySQL Documentation: MySQL NDB Cluster, on the MySQL website.

MySQL Cluster Manager features. This manual describes features that may not be included in every version of MySQL Cluster Manager, and such features may not be included in the version of MySQL Cluster Manager licensed to you. If you have any questions about the features included in your version of MySQL Cluster Manager, refer to your MySQL Cluster Manager license agreement or contact your Oracle sales representative.

MySQL Cluster Manager, MySQL Server, and MySQL NDB Cluster features. This manual contains certain basic information about MySQL Server and MySQL NDB Cluster; however, it is not in any way intended as an exhaustive reference for either of these products.

MySQL Cluster Manager 9.6 supports MySQL NDB Cluster 9.6, MySQL NDB Cluster 8.4, and MySQL NDB Cluster 8.0.

MySQL NDB Cluster functionality varies between MySQL NDB Cluster releases; MySQL Cluster Manager cannot supply or emulate MySQL NDB Cluster features that are not present in the version of the MySQL NDB Cluster software in use.

For complete information about MySQL Server and MySQL NDB Cluster see MySQL NDB Cluster 9.6, MySQL NDB Cluster 8.4, and MySQL NDB Cluster 8.0.

If you do not have the MySQL Server and MySQL NDB Cluster documentation, you can obtain it free of charge from the MySQL Documentation Library, on the MySQL website.

For notes detailing the changes in each release of MySQL Cluster Manager , see MySQL Cluster Manager 9.6 Release Notes.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2026-01-15 (revision: 84253)

Table of Contents

Preface and Legal Notices

This is the User Manual for the *MySQL Cluster Manager*, version 9.6.0. It documents the *MySQL Cluster Manager Agent* and *MySQL Cluster Manager Client* software applications, which can be used to administer *MySQL NDB Cluster*, a version of the *MySQL Database System* (referred to hereafter as “MySQL Server” or simply “MySQL”) that incorporates the NDB storage engine for high availability and data redundancy in a distributed computing environment.

Licensing information. This product may include third-party software, used under license. See the MySQL Cluster Manager 9.6 License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this release.

Legal Notices

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Chapter 1 Overview of MySQL Cluster Manager

Table of Contents

This chapter provides a overview of MySQL Cluster Manager, as well as its architecture, purpose, and capabilities.

1.1 MySQL Cluster Manager Terminology

This section provides definitions of key terms used to describe MySQL Cluster Manager and its components in this manual and in other documentation relating to MySQL Cluster Manager and MySQL NDB Cluster.

Site.
A set of hosts on which MySQL NDB Cluster processes to be managed by MySQL Cluster Manager are located. A site can include one or more clusters.

Cluster.
A MySQL NDB Cluster deployment. A cluster consists of a set of MySQL NDB Cluster processes running on one or more hosts. A minimal cluster is usually considered to include one management node, two data nodes, and one SQL node. A typical production cluster may have one or two management nodes, several SQL nodes, and 4 or more data nodes. The exact numbers of data and SQL nodes can vary according to data size, type and rating of hardware used on the hosts, expected throughput, network characteristics, and other factors; the particulars are beyond the scope of this document, and you should consult MySQL NDB Cluster 9.6, for more specific information and guidelines.

Host.
A computer. The exact meaning depends on the context:

- A computer where one or more MySQL NDB Cluster processes are run. In this context, we sometimes refer more specifically to a *cluster host*.

  The number of cluster processes and number of cluster hosts may be, but are not necessarily, the same.

- A computer where an instance of the MySQL Cluster Manager agent runs.

In order to run a MySQL NDB Cluster using MySQL Cluster Manager, the MySQL Cluster Manager agent must be running on each host where cluster processes are to be run. In other words, when using MySQL Cluster Manager, all cluster hosts must also be MySQL Cluster Manager agent hosts (although the reverse is not necessarily true). Therefore, you should understand that anytime we use the term *host*, we are referring to a host computer in both of the senses just given.

Process.
In the context of MySQL NDB Cluster, a process (more specifically, a cluster process) is a MySQL NDB Cluster node, of one of the following 3 types: management node (`ndb_mgmd`), data node (`ndbd` or `ndbmtd`), or SQL node (`mysqld`). For more information about these node types and their functions in a cluster, see NDB Cluster Core Concepts, and NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions.

Package.
A copy of the MySQL NDB Cluster software. This includes the binary executables needed to run the cluster processes of the desired types on a given host. The simplest way to make sure that this is done is to place a copy of the entire MySQL NDB Cluster distribution on each computer that you intend to use as a cluster host.

Configuration attribute.
A value whose setting affects cluster operations in a clearly defined and measurable way. When running MySQL NDB Cluster manually, configuration is accomplished using cluster configuration parameters, MySQL server options, and MySQL system and status variables; MySQL Cluster Manager masks the differences between these, providing a unified view of them; see Configuration attributes, for more information.

Agent.
A MySQL Cluster Manager process that runs on each cluster host, responsible for managing the cluster processes running on that host.

Client.
The MySQL Cluster Manager client is a software application that allows a user to connect to MySQL Cluster Manager and perform administrative tasks, such as (but not limited to): creating, starting, and stopping clusters; obtaining cluster and cluster process status reports; getting cluster configuration information and setting cluster configuration attributes.

1.2 MySQL Cluster Manager Architecture

This section provides an architectural overview of MySQL Cluster Manager, its components, and their deployment.

MySQL Cluster Manager is a distributed client/server application consisting of two main components. The MySQL Cluster Manager agent is a set of one or more agent processes that manage NDB Cluster nodes, and the MySQL Cluster Manager client provides a command-line interface to the agent's management functions.

Agent.
The MySQL Cluster Manager agent is comprised of the set of all MySQL Cluster Manager agent processes running on the hosts making up a given management site. A MySQL Cluster Manager agent process is a daemon process that runs on each host to be used in the cluster. In MySQL Cluster Manager, there is no single central server or process; all agents collaborate in managing a cluster as a whole. This means that any connected agent can be used to carry out tasks that effect the entire cluster.

Each agent process is responsible for managing the MySQL NDB Cluster nodes running on the host where the agent is located. MySQL NDB Cluster management and SQL nodes are managed directly by the MySQL Cluster Manager agent; cluster data nodes are managed indirectly, using the cluster management nodes.

Management responsibilities handled by the MySQL Cluster Manager agent include the following:

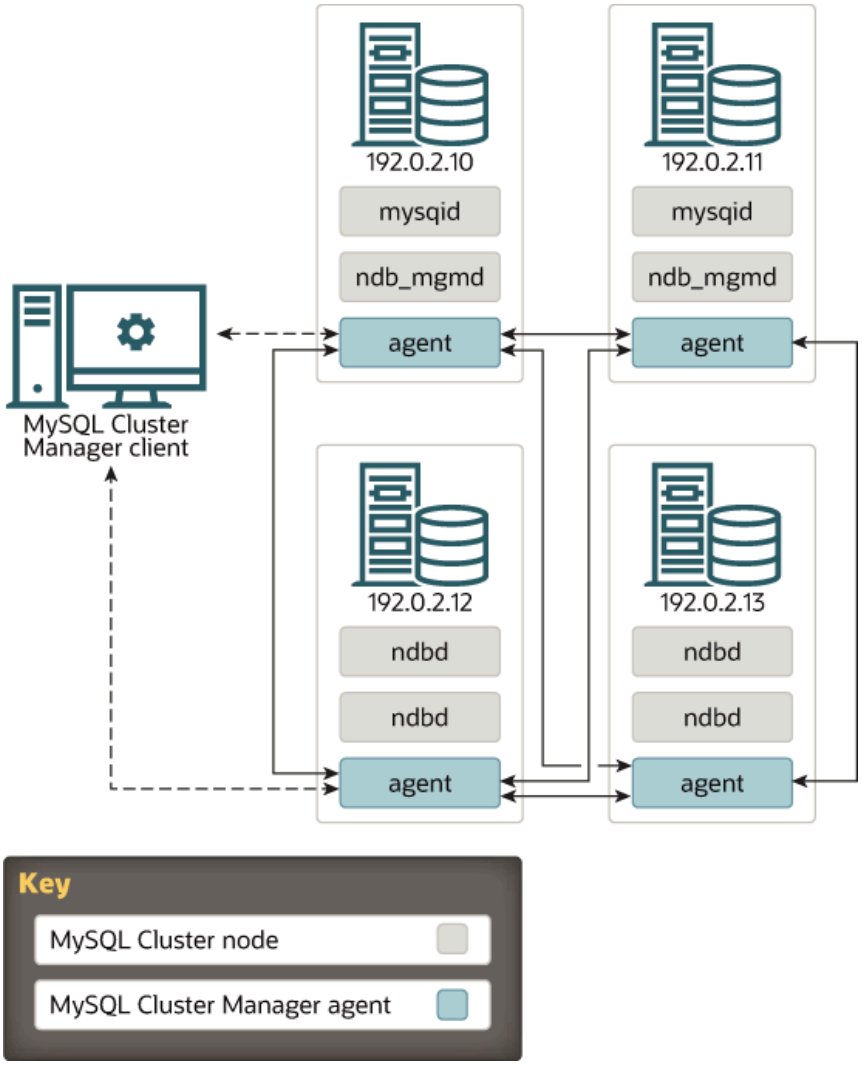
- Creating and deleting clusters
- Starting, stopping, and restarting cluster nodes
- Cluster configuration changes
- Backing up and restoring clusters
- Cluster software upgrades
- Host and node status reporting
- Recovery of failed cluster nodes

Creating, performing initial configuration of, or starting a cluster, requires that agent processes be running on all cluster hosts. Once the cluster has been started, it continues to run even if one or more agent processes fail. However, any failed agent processes must be restarted before you can perform additional cluster management functions.

Client.
A MySQL Cluster Manager client is a software application used to access an MySQL Cluster Manager agent. The `mcm` client in the MySQL Cluster Manager release package is based on the `mysql` client.

By way of example, we show how MySQL Cluster Manager would be deployed for use with a MySQL NDB Cluster running on 4 host computers. This is illustrated in the following diagram:

Figure 1.1 MySQL Cluster Manager Deployment



In this example cluster, two of the hosts each house a management server (`ndb_mgmd`) and an SQL node (`mysqld`); the other 2 hosts each house two data nodes (`ndbd`). However, regardless of the distribution of cluster nodes among the hosts, a MySQL Cluster Manager agent process must be running on each host.

A MySQL Cluster Manager client can be used to access the agent from any of the hosts making up the management site to which the cluster belongs. In addition, the client can be used on any computer that has a network connection to at least one of the hosts where an agent process is running. The computer where the client itself runs is not required to be one of these hosts. The client can connect to and use different agent processes on different hosts within the management site, at different times, to perform cluster management functions.

1.3 Basic Operational Concepts for MySQL Cluster Manager

This section explains some basic operational concepts for MySQL Cluster Manager.

1.3.1 Quorum Requirement

MySQL Cluster Manager uses the XCom (a Paxos variant) as its consensus protocol. Before XCom allows a new message to be delivered, it requires a majority vote by all agents to decide who can send the next message. This majority is referred to as a "quorum." In general, a simple majority of the agents (that is, half of the total number of agents plus one) constitutes a quorum. So, the quorum for a cluster with 4 agents is 3, and for 6 agents is 4, and so on. Here are some special cases:

- An agent for a single-host cluster forms a quorum. All messages are automatically delivered.
- A 2-host group does *not* require a simple majority of agents—if full consensus by both agents is not obtained, XCom allows a message to be sent only by the agent on the first host in the host list of the `create site` command.

On top of the quorum requirement described above, for any execution plans, any agent that has work to do in the plan needs to accept the plan before it can be executed.

1.3.2 Eventual Consistency

MySQL Cluster Manager guarantees *eventual consistency* among agents, meaning that:

- Any message communicated among agents is either delivered or not delivered to ALL agents (instead of delivered to some and missed by others).
- Order of delivery for any sequence of messages is always identical for all agents (that is, messages cannot get out of order for some agents).

Beyond that, there is no guarantee for message synchronization: a message is not guaranteed to be received and executed within a specific window of time for all agents. The result is that any agent may lag behind in processing messages for any reasons such as network traffic, machine loading, or thread scheduling. Situations like the following might then occur: while Agent C is lagging behind Agent A and B, the two agents have completed some cluster reconfiguration that does not involve any local actions for Agent C; a client connected to Agent A or B might have received a success message for the reconfiguration, while a client querying Agent C is told that the reconfiguration is still in process, since the completion message has not yet reached Agent C.

Such temporary inconsistency among the agents should not be a concern. While an agent might be lagging behind, the guaranteed eventual consistency means that, unless prevented by a network error or some other issues, any lagging agent will eventually catch up with the other agents, and all agents will eventually get a consistent view of the site.

Chapter 2 What Is New in MySQL Cluster Manager 9.6

MySQL Cluster Manager 9.6 is an Innovation release, which means it has new features in addition to bug fixes, and it is supported until the next Innovation release comes out. MySQL Cluster Manager 9.6 is recommended for use on production systems. With this new Innovation series, the existing 8.4 series focuses on security and bug fixes only.

For notes detailling the changes in MySQL Cluster Manager 9.6, see MySQL Cluster Manager 9.6 Release Notes

Chapter 3 MySQL Cluster Manager Installation, Configuration, Cluster Setup

Table of Contents

This chapter discusses basic installation and configuration of the MySQL Cluster Manager Management Agent, connecting to the agent with the MySQL Cluster Manager client, and the basics of creating or importing a cluster using MySQL Cluster Manager.

3.1 Obtaining MySQL Cluster Manager

MySQL Cluster Manager is available only through commercial license. To learn more about licensing terms, and to obtain information about where and how to download MySQL Cluster Manager, visit https://www.mysql.com/products/cluster/mcm/, or contact your Oracle representative.

3.2 Supported Platforms and MySQL NDB Cluster Versions

For a list of platforms supported by MySQL Cluster Manager 9.6.0, see *Supported Platforms: MySQL Cluster Manager* at https://www.mysql.com/support/supportedplatforms/cluster-manager.html, or contact your Oracle representative.

MySQL Cluster Manager 9.6.0 supports the following MySQL NDB Cluster release versions:

- MySQL NDB Cluster 9.6 (see MySQL NDB Cluster 9.6).
- MySQL NDB Cluster 8.4 (see MySQL NDB Cluster 8.4). Notice that there are Unsupported MySQL 8.4 and 8.0 Features.
- MySQL NDB Cluster 8.0 (see MySQL NDB Cluster 8.0), beginning with MySQL NDB Cluster 8.0.24. Notice that there are Unsupported MySQL 8.4 and 8.0 Features.

Prior to installation, you must obtain the correct build of MySQL Cluster Manager for your operating system and hardware platform. For Unix-like platforms, MySQL Cluster Manager is delivered either as a `.tar.gz` archive, whose name is in the format of `mcm-9.6.0-cluster-9.6.0-`*`linux-distro-arch`*`.tar.gz`, or as an RPM package (for selected distros). For Windows platforms, an MSI installer file is provided. All MySQL Cluster Manager 9.6.0 packages include MySQL NDB Cluster 9.6.0.

3.3 MySQL Cluster Manager Installation

Installation of the MySQL Cluster Manager agent and client programs varies according to platform. See the installation instructions below.

3.3.1 Installing MySQL Cluster Manager on Unix-like Platforms

Install MySQL Cluster Manager on Linux and similar platforms by following the instructions below for the different methods.

3.3.1.1 Installing MySQL Cluster Manager Using Tarballs

Extract the MySQL Cluster Manager 9.6.0 program and other files from the distribution archive.
You must install a copy of MySQL Cluster Manager on each computer that you intend to use as a MySQL NDB Cluster host. In other words, you need to install MySQL Cluster Manager on each host that is a member of a MySQL Cluster Manager management site. For each host, you should use the MySQL Cluster Manager build that matches that computer's operating system and processor architecture.

On Linux systems, you can unpack the archive using the following command, which uses `mcm-9.6.0-cluster-9.6.0-linux-glibc2.17-x86-64bit.tar.gz` as an example (the actual filename will vary according to the MySQL Cluster Manager build that you intend to deploy):

```
$> tar -zxvf mcm-9.6.0-cluster-9.6.0-linux-glibc2.17-x86-64bit.tar.gz
```

This command unpacks the archive into a directory having the same name as the archive, less the `.tar.gz` extension. The top-level directories under the unpacked directory are `cluster` and `mcm-9.6.0`.

Important

Because the Solaris version of `tar` cannot handle long filenames correctly, the MySQL Cluster Manager program files may be corrupted if you try to use it to unpack the MySQL Cluster Manager archive. To get around this issue on Solaris operating systems, you should use GNU `tar` (`gtar`) rather than the default `tar` supplied with Solaris. On Solaris, `gtar` is often already installed in the `/usr/sfw/bin` directory, although the `gtar` executable may not be included in your path. If `gtar` is not present on your system, please consult the Oracle Solaris Documentation for information on how to obtain and install it.

In general, the location where you place the unpacked MySQL Cluster Manager directory and the name of this directory can be arbitrary. However, we recommend that you use a standard location for optional software, such as `/opt` on Linux systems, and that you name the directory using the 9.6.0 version number (this facilitates subsequent upgrades). On a typical Linux system you can accomplish this task like this:

```
$> cd mcm-9.6.0-cluster-9.6.0-linux-glibc2.17-x86-64bit
$> mv mcm-9.6.0 /opt/mcm-9.6.0
```

For ease of use, we recommend that you put the MySQL Cluster Manager files in the same directory on each host where you intend to run it.

Contents of the MySQL Cluster Manager Unix Distribution Archive.
If you change to the directory where you placed the extracted MySQL Cluster Manager archive and list the contents, you should see something similar to what is shown here:

```
$> cd /opt/mcm-9.6.0
$> ls
bin  docs  etc  lib  licenses  share  var
```

These directories are described in the following table:

Table 3.1 Contents of the MySQL Cluster Manager Unix distribution archive, by directory

Directory	Contents
`bin`	MySQL Cluster Manager agent and client executables
`docs`	Contains the sample configuration file, `sample_mcmd.conf`, the `LICENSE` file, and the `README.txt` file

Directory	Contents
`etc/init.d`	Contains the `init` scripts
`lib` and subdirectories	Libraries needed to run the MySQL Cluster Manager agent
`var`	XML files containing information needed by MySQL Cluster Manager about processes, attributes, and command syntax

Normally, the only directories of those shown in the preceding table that you need be concerned with are the `bin`, `docs`, and `etc` directories.

For MySQL Cluster Manager 9.6.0 distributions that include MySQL NDB Cluster, the complete MySQL NDB Cluster 9.6.0 binary distribution is included in the `cluster` directory. Within this directory, the layout of the MySQL NDB Cluster distribution is the same as that of the standalone MySQL NDB Cluster binary distribution. For example, MySQL NDB Cluster binary programs such as `ndb_mgmd`, `ndbd`, `ndbmtd`, and `ndb_mgm` can be found in `cluster/bin`. For more information, see MySQL Installation Layout for Generic Unix/Linux Binary Package, and Installing an NDB Cluster Binary Release on Linux, in the *MySQL Manual*.

If you wish to use the included MySQL NDB Cluster software, it is recommended that you move the `cluster` directory and all its contents to a location outside the MySQL Cluster Manager installation directory, such as `/opt/ndb-version`. For example, on a Linux system, you can move the MySQL NDB Cluster NDB 9.6.0 software that is bundled with MySQL Cluster Manager 9.6.0 to a suitable location by first navigating to the directory unpacked from the distribution archive and then using a shell command similar to what is shown here:

```
$> mv cluster /opt/ndb-9.6.0
```

Note

The `mcmd --bootstrap` option uses the MySQL NDB Cluster binaries in the `cluster` folder that is under the same directory as the MySQL Cluster Manager installation directory, and bootstrapping fails if the binaries cannot be found there. To work around this issue, create a symbolic link to the correct directory in the directory above the installation directory, like this:

```
$> ln -s /opt/ndb-9.6.0 cluster
```

After doing this, you can use the `mcm` client commands `add package` and `upgrade cluster` to upgrade any desired cluster or clusters to the new MySQL NDB Cluster software version.

Important

On Linux platforms, do not attempt to install the MySQL NDB Cluster software by the RPM, Debian, or other installation packages for any package management systems. They install MySQL NDB Cluster differently than the binary distribution that comes with the MySQL Cluster Manager archive, and that will cause issue in the future when you try to upgrade your cluster with MySQL Cluster Manager.

The MySQL Cluster Manager agent by default writes its log file as `mcmd.log` in the same directory where the installation directory is found. When the agent runs for the first time, it creates a directory where the agent stores its own configuration data; by default, that is `mcm_data` in the parent directory of the MySQL Cluster Manager installation directory. The configuration data, log files, and data node file systems for a given MySQL NDB Cluster under MySQL Cluster Manager control, and named *`cluster_name`*, can be found in `clusters/cluster_name` under this data directory (sometimes also known as the MySQL Cluster Manager *data repository*).

The location of the MySQL Cluster Manager agent configuration file, log file, and data directory can be controlled with `mcmd` startup options or by making changes in the agent configuration file. To

simplify upgrades of MySQL Cluster Manager, we recommend that you change the data repository to a directory outside the MySQL Cluster Manager installation directory, such as `/var/opt/mcm`. See Section 3.4, “MySQL Cluster Manager Configuration File”, and Section 4.2, “Starting and Stopping the MySQL Cluster Manager Agent”, for more information.

MySQL Cluster Manager init script. On Linux and other Unix-like systems, you can set up the MySQL Cluster Manager agent to run as a daemon, using the init script that is supplied with the MySQL Cluster Manager distribution.

To do this, follow the steps listed here:

1. Copy the file `/etc/init.d/mcmd` under the MySQL Cluster Manager installation directory to your system's `/etc/init.d/` directory (or equivalent). On a typical Linux system, you can do this using the following command in the system shell, where *`mcmdir`* is the MySQL Cluster Manager installation directory:

   ```
   $> cd mcmdir/etc/init.d
   $> cp mcmd /etc/init.d/mcmd
   ```

2. Make sure that this file has appropriate permissions and is executable by the user account that runs MySQL Cluster Manager. On a typical Linux system, this can be done by executing commands in your system shell similar to those shown here:

   ```
   $> chown mcmuser /etc/init.d/mcmd
   $> chmod 755 /etc/init.d/mcmd
   ```

   Be sure to refer to your operating system documentation for exact information concerning the commands needed to perform these operations, as they may vary between platforms.

3. Open the file `/etc/init.d/mcmd` in a text editor. Here, we show a portion of this file, in which we have highlighted the two lines that need to be updated:

   ```
   MCMD_SERVICE="mcmd"
   MCMD_PSERVICE="MySQL Cluster Manager"
   MCMD_ROOTDIR=@@MCMD_ROOTDIR@@
   MCMD_BIN="$MCMD_ROOTDIR/bin/mcmd"
   MCMD_CONFIG="$MCMD_ROOTDIR/etc/mcmd.conf"

   # Run service as non-root user
   MCMD_USER=@@MCMD_USER@@
   SU="su --login $MCMD_USER --command"
   ```

   In the first of the highlighted lines, replace the placeholder `@@MCMD_ROOTDIR@@` with the complete path to the MySQL Cluster Manager installation directory. In the second of these lines, replace the placeholder `@@MCMD_USER@@` with the name of the system user that runs the MySQL Cluster Manager agent (note that this must *not* be the system `root` account). Save the edited file.

The MySQL Cluster Manager agent should now be started automatically whenever the system is restarted.

When the agent is configured as a daemon, cluster processes are started automatically when the agent is restarted, as long as the cluster was running when the agent shut down; however, *`StopOnError` must be disabled (set to 0) for all data nodes in order for that to work*. If the cluster was stopped when the agent shut down, it is necessary to have in place a script that waits for the agent to complete its startup and recovery phases, and then, when the agent is ready, starts the cluster using a command such as *`mcmdir`*`/bin/mcm -e 'start cluster `*`cluster_name`*`;'`.

Install MySQL Cluster Manager as a service using systemd. On Linux and other Unix-like systems that supports systemd, you can set up the MySQL Cluster Manager agent to run as a service by following theses steps:

1. Create the system user `mcm` to run the `mcm` service

   ```
   sudo useradd --no-create-home -s /bin/false mcm
   ```

2. Set the necessary file and folder permissions (replace *`mcmdir`* with the path for your MySQL Cluster Manager installation directory)

```
sudo chown -R mcm:mcm mcmdir
chmod 600 mcmdir/mcmd.conf
```

3. Create the systemd configuration file `/etc/systemd/system/mcm.service` for the `mcm` service:

```
[Unit]
Description=MySQL Cluster Manager
Documentation=https://dev.mysql.com/doc/mysql-cluster-manager/en/
After=network-online.target

[Service]
User=mcm
Group=mcm
Restart=always
Type=simple

ExecStart=mcmdir/mcm9.6.0/bin/mcmd --config=mcmdir/mcmd.conf

[Install]
WantedBy=multi-user.target
```

4. Reload systemd configuration files for your system, to make service addition take effect:

```
sudo systemctl daemon-reload
```

5. Start, enable, and check status of the service by these commands

```
sudo systemctl start mcm
sudo systemctl enable mcm
sudo systemctl status mcm
```

   If the service is not started correctly, look in the `messages` file:

```
sudo tail -150f /var/log/messages
```

When the agent is configured as a service, cluster processes are started automatically when the agent is restarted, as long as the cluster was running when the agent shut down; however, *`StopOnError` must be disabled (set to 0) for all data nodes in order for this to happen*. If the cluster was stopped when the agent shut down, it is necessary to have in place a script that waits for the agent to complete its startup and recovery phases, and then, when the agent is ready, starts the cluster using a command such as *`mcmdir`*`/bin/mcm -e 'start cluster `*`cluster_name`*`;'`.

3.3.1.2 Installing MySQL Cluster Manager Using RPM Packages

Oracle provides RPM packages for installing MySQL Cluster Manager on Oracle Linux, Red Hat Enterprise Linux, CentOS, and SUSE Linux Enterprise Server.

Note

RPM packages for MySQL Cluster Manager are not available on the MySQL Yum Repository.

Use the `rpm` command (instead of your OS distribution's package management system) to install the RPM package. For example:

```
rpm -ivh mcm-9.6.0-linux-distro-arch.rpm #install mcm by stand-alone rpm
```

3.3.2 Installing MySQL Cluster Manager on Windows Platforms

To install MySQL Cluster Manager on Windows platforms, obtain the MSI installer for it (see Section 3.1, “Obtaining MySQL Cluster Manager”), then follow these steps:

- Run the installer by double-clicking it in Windows Explorer (some versions of Windows also provide an **Install** item in the Windows Explorer menu that can be used to run the installer). When you start the installer, you may see a Windows **Security Warning** screen. If you obtained the installer from a trusted source and know that it has not been tampered with, choose **Run** from the dialog, which allows the installer to continue to the **Welcome** screen.

- Click the **Next** button to continue to the **License Agreement** screen. You should read the license text in the text area, and when you have done so, check the box labelled **I accept the terms in the License Agreement**. Until you have checked the box, you cannot complete the MySQL Cluster Manager installation; it is possible only to print the license, go back to the previous screen, or cancel the installation (using the buttons labelled **Print**, **Back**, and **Cancel**, respectively). Checking the box enables the **Next** button.

- Click the **Next** button to continue to the **Destination Folder** screen, where you can choose the installation directory. The default location is `C:\Program Files\MySQL\MySQL Cluster Manager\`. You can click the **Change** button to change the directory where MySQL Cluster Manager should be installed; the default directory is adequate for most cases.

  Once you have selected the destination directory, the installer has gathered all the information that it requires to perform the installation. Click **Next** to continue to the **Ready** screen

- Click the **Install** button to install MySQL Cluster Manager. As the installer begins to copy files and perform other tasks affecting the system, you may see a warning dialog from Windows User Access Control. If this occurs, click the **Yes** button to allow the installation to continue. A **Setup Wizard** screen with a progress bar is displayed while the installer runs.

  The **Setup Wizard** may require several minutes to copy all of the necessary files for MySQL Cluster Manager and MySQL NDB Cluster to the installation directory and to perform other required changes.

  Note

  The MySQL Cluster Manager installer places MySQL NDB Cluster in the `cluster` directory under the installation directory (by default, that is `C:\Program Files\MySQL\MySQL Cluster Manager\cluster`). The location of the MySQL NDB Cluster binaries is not separately configurable when using the MySQL Cluster Manager installer.

- When the **Setup Wizard** finishes, the installer displays the **Installation Completed** screen. MySQL Cluster Manager has now been installed to the destination directory; click the **Finish** button to exit the installer.

As mentioned elsewhere (see, for example, Section 4.4.1, “Creating a MySQL NDB Cluster with MySQL Cluster Manager”), you must install a copy of MySQL Cluster Manager on each computer where you intend to host a MySQL NDB Cluster node. Therefore, the above procedure must be performed separately on each host computer. For ease of installations and upgrades on multiple machines, it is recommended that you install MySQL Cluster Manager to the same location on each host. `C:\Program Files\MySQL\MySQL Cluster Manager\` is the default location for installation, but it is possible to install MySQL Cluster Manager to an alternate location such as `C:\mcm\`.

3.3.2.1 Installing the MySQL Cluster Manager Agent as a Windows Service

Important

Installation of the MySQL Cluster Manager agent as a service is recommended. However, you should *not* install MySQL NDB Cluster processes (`ndb_mgmd.exe`, `ndbd.exe`, `ndbmtd.exe`, `mysqld.exe`) as services on Windows hosts to be used as MySQL NDB Cluster nodes under management by MySQL Cluster Manager, since the MySQL Cluster Manager agent itself

controls MySQL NDB Cluster nodes independently of the Windows **Services** application.

After installing the MySQL Cluster Manager Agent as a Windows service, you can start and stop the agent using the Windows **Services** application. The installation also configures the agent to start automatically whenever Windows starts, and to shut down safely whenever Windows shuts down.

Note

The Windows service can be used to control the running of MySQL Cluster Manager agents on a single host only. To shut down agents on multiple hosts, you can use the `stop agents` command in the MySQL Cluster Manager client.

The installation is performed using the command prompt (`cmd.exe`); as with installing or removing any Windows service, it must also be done as a user having sufficient permissions, such the system Administrator account. Follow these steps:

- If the account you are currently using has Administrator privileges, you can simply start `cmd.exe`. Otherwise, you must run the command prompt program as the Administrator. To do this, first locate a shortcut to the command prompt. You can do this by typing **cmd** into the search box in the Windows **Taskbar**, and then select from the search results **Command Prompt** > **Run as Administrator**.

  If a Windows UAC dialog referring to `cmd.exe` appears, click **Yes** to allow the command prompt to run as Administrator and thus to continue. You should now have a command prompt window open on your desktop, running a session with Administrator privileges.

- To install the MySQL Cluster Manager agent as a service, we use the `SC CREATE` command. This command allows us to specify a name for the service (for use with `SC START` and `SC STOP` or `NET START` and `NET STOP` commands), a display name (to be shown in the **Services** application), a startup mode (automatic or manual start), and a path to the executable to be run as a service. The path must also include any arguments needed by the program; in the case of MySQL Cluster Manager, `mcmd.exe` must be told where to find its configuration file by the `--config` option. Both of these paths must be absolute. Assume that you have installed MySQL Cluster Manager to the default location (`C:\Program Files\MySQL\MySQL Cluster Manager\mcm9.6.0`), and that its configuration file is located in `C:\Program Files\MySQL\MySQL Cluster Manager\mcm9.6.0\`; then, the following command installs MySQL Cluster Manager as a service named `MCM`, with the display name “MySQL Cluster Manager 9.6.0”:

  ```
  SC CREATE "MCM" DisplayName= "MySQL Cluster Manager 9.6.0" Start= "auto"
    BinPath= "C:\Program Files\MySQL\MySQL Cluster Manager\mcm9.6.0\bin\mcmd.exe
    --config=\"C:\Program Files\MySQL\MySQL Cluster Manager\mcm9.6.0\mcmd.conf\""
  ```

  This command can be quite long. For enhanced legibility, we have broken it across several lines, *but you should always enter it on a single line*, allowing it to wrap naturally. In addition, you should keep in mind that the spaces after the equal signs following the `DisplayName`, `Start`, and `BinPath` arguments are required.

Starting and stopping the MySQL Cluster Manager agent Windows service. After installing the service successfully, you can start and stop the service manually, if the need arises, with the `SC START` and `SC STOP` commands.

```
C:\>SC START MCM
C:\>SC STOP MCM
```

Alternatively, use the `NET START` and `NET STOP` commands:

```
C:\Windows\system32>NET START MCM
C:\Windows\system32>NET STOP MCM
```

Once the service is installed, the MySQL Cluster Manager agent starts automatically whenever Windows is started. You can verify that the service is running with the Windows **Task Manager**

(which can be searched and then run using the search box in the Windows **Taskbar**). Open the **Task Manager**, and switch to the **Services** tab if it is not already displayed. If the MySQL Cluster Manager agent is running, you can find it in the list of services under `MCM` in the `Name`, column and `MySQL Cluster Manager 9.6.0` in the `Description` column.

You can also verify if the service is running using the Windows **Services** application(which can be searched and then run using the search box in the Windows **Taskbar**). The application also allows you to start, stop, or pause the MySQL Cluster Manager agent service manually using a GUI.

Note

When first installing the MySQL Cluster Manager agent as a service, the service is not started automatically until Windows is restarted. If you do not wish to restart Windows, then you must start the service manually using either `SC START` or `NET START` on the command line or the graphical control provided in the Windows **Services** application.

You can remove the service using the `SC DELETE` command and the name of the service—in this case `MCM`—that was used in the `SC CREATE` command. If the service is running at the time that `SC DELETE` is executed, the removal of the service takes effect the next time the service is stopped. In such a case, you must stop the previous instance of the service manually, and allow it to be removed, before you can reinstall the service.

Once you have installed the MySQL Cluster Manager agent and the service is running correctly, you are ready to connect to it using the MySQL Cluster Manager client. See Section 4.3, “Starting the MySQL Cluster Manager Client”, for information about how to do this.

3.3.3 Setting the MySQL Cluster Manager Agent User Name and Password

Normally it is not necessary to alter the user name or password used by the user agent to administer `mysqld` processes. However, if you should wish to do so, you can change either or both of these, using the procedure outlined here:

1. Stop all agents. (You can use `stop agents` for this purpose.)

2. Update the agent configuration file. Set a new password by uncommenting the line containing `mcmd_password=` and adding the new password as its value; set a new administrative user account name by uncommenting the line containing `mcmd_user=` and setting the value to the new user name. See Section 4.1, “mcmd, the MySQL Cluster Manager Agent”, for more information about these options.

3. For each `mysqld` do the following:

   a. Log in (using the `mysql` client) as the MySQL `root` user

   b. If you are changing the user name, do this first, using the following statement, where *`olduser`* is the current user name and *`newuser`* is the new `manager-user` that you set previously in the agent configuration file:

   ```
   RENAME USER 'olduser'@'127.0.0.1' TO 'newuser'@'127.0.0.1';
   ```

   If you are changing the user name for the first time, use `mcmd` for *`olduser`*. In addition, you should use `127.0.0.1` for the host name (and not `localhost`).

   c. Execute the following statement, where *`newuser`* is the new user name, and *`newpass`* is the new password:

   ```
   SET PASSWORD FOR 'newuser'@'127.0.0.1' = PASSWORD('newpass');
   ```

   Use `mcmd` for the user name if you have not changed it—that is, if `mcmd-user` has been left unset in the agent configuration file. Use `127.0.0.1` for the host name (and not `localhost`).

   d. Issue a `FLUSH PRIVILEGES` statement.

4. Restart the agents. All agents should now be using the new password for the `mcmd` accounts on the MySQL servers (`mysqld` processes).

3.4 MySQL Cluster Manager Configuration File

The `mcmd` configuration file allows you to configure `mcmd` with its application options. A sample configuration file is provided with the MySQL Cluster Manager distribution at `mcmd9.6.0/doc/sample_mcmd.conf`. Edit the `[mcmd]` section of the file and save it as `mcmd.conf` at a desired location, which you will provide to `mcmd` with the `--config` option when you start the agent. If the option is not used, `mcmd` looks for the configuration file at these default locations, in the following order of priority:

- *`mcmd-installation-directory`*`/mcmd.conf`
- *`mcmd-installation-directory`*`/mcmd.ini`
- *For Linux systems:* *`OS-user's-home-directory`*`/mcmd.conf`

  For Windows systems: `C:\Users\`*`user`*`\AppData\Roaming/mcmd.conf`
- *For Linux systems:* *`OS-user's-home-directory`*`/mcmd.ini`

  For Windows systems: `C:\Users\`*`user`*`\AppData\Roaming/mcmd.ini`

Notes

- *For Linux platforms:* The configuration file must have proper file permissions: `mcmd` refuses to start if the configuration file has permissions more open than just read and write for just the owner and the group owner (that is, more open than 660 in numeric notation for permissions).
- *For Windows platforms:* It is recommended that you save the configuration file to a convenient location for which the path does not contain any spaces, such as `C:\mcm\data`.

The format of the configuration file follows that of the MySQL Router configuration file; see Configuration File Syntax for an explanation of the file format.

The following is the sample configuration file; uncomment and adjust the values of the listed options, or add other options:

```
# Copyright (c) 2018, 2021, Oracle and/or its affiliates.
#
# MySQL Cluster Manager sample configuration
#

#[DEFAULT]
#logging_folder = /var/log
#data_folder = /var/lib/mcm_data
#pid_file = mcmd.pid

[logger]
#level = INFO
#filename = mcmd.log

# The MySQL Cluster Manager plugin
[mcmd]
#mcmd_user = mcmd
#mcmd_password = super
#bind_address = ::
#bind_port = 1862
#xcom_port = 18620
```

```
#copy_port = 0
```

For more information about the options that can be set in the agent configuration file, see Section 4.1, “mcmd, the MySQL Cluster Manager Agent”.

3.5 Upgrading MySQL Cluster Manager

This section discusses upgrading MySQL Cluster Manager from a previous release to the latest release (currently 9.6.0), as well as providing basic guidance on upgrading the bundled MySQL NDB Cluster software.

Notes

- Only upgrades from MySQL Cluster Manager 8.0 and later are supported.
- You cannot upgrade from MySQL Cluster Manager 1.4.x or earlier to release 9.6 or later directly; upgrade to release 8.0.34 or later first and then upgrade to 9.6, using the method outlined below.

The basic steps for upgrading a MySQL Cluster Manager installation are listed here:

1. Install the new version of the MySQL Cluster Manager software in the desired location.
2. Create a configuration for the new installation such that it uses the previous installation's data.
3. Stop all running MySQL Cluster Manager agent processes on all hosts.
4. Start the new agent processes, ensuring that they use the new configuration just created.

A more detailed explanation is provided of each of these steps in the next few paragraphs. For illustrative purposes, we assume an upgrade from an existing installation of MySQL Cluster Manager 8.0.44 to a new installation of MySQL Cluster Manager 9.6.0. For a Linux or a Unix-like system, we assume that the installation directories are `/opt/mcm-8.0.44` and `/opt/mcm-9.6.0`, respectively; on Windows, we assume the installation directories are `C:\Program Files\MySQL\MySQL Cluster Manager 8.0.44\` and `C:\Program Files\MySQL\MySQL Cluster Manager 9.6.0\`, respectively.

Step 1: Install new MySQL Cluster Manager version. You can obtain and install a new version of MySQL Cluster Manager in the same way as for a new installation (see Section 3.1, “Obtaining MySQL Cluster Manager”, and Section 3.3, “MySQL Cluster Manager Installation”), with the additional requirement that you should not attempt to install the new version in the same location as the version that you are currently using.

Step 2: Configure new installation. In order for the new MySQL Cluster Manager agent binaries to manage the same MySQL NDB Cluster instances, they must be able to find the data contained in the agent repository used by the old installation's binaries, which is `mcm_data` in the parent directory of the MySQL Cluster Manager installation directory by default, but can be set using the `data_folder` option in the agent configuration file.

It is simplest for MySQL Cluster Manager software upgrades if the agent repository and the agent configuration file are located outside of the agent installation directory. Suppose the old version of the agent is installed to `/opt/mcm-8.0.44`, and that it uses the directory `/var/opt/mcm` for its agent repository and `/etc/mcm/mcmd.ini` for its configuration file. In this case, to make the new binaries use the same configuration and repository, create first the new configuration file (in the format for MySQL Cluster Manager 9.6), at a desired location. You can do this by making a copy of the sample configuration file to the described location.

```
cp /mcm-install-dir/doc/sample_mcmd.conf /etc/mcm/mcmd.conf
```

You should then copy over the old configuration settings from the 8.0 configuration file to the new configuration file, paying special attention to any differences in the configuration file format and the

agent options (see Chapter 2, *What Is New in MySQL Cluster Manager 9.6* and Section 3.4, “MySQL Cluster Manager Configuration File” for details; for upgrade from the 8.1 series and later, you can just reuse the old configuration file, making adjustments only for options that have changed from version to version.)

To use the old data repository, add the following line to the new copy of the `mcmd.conf` file:

```
data_folder=/var/opt/mcm
```

After this, you can save and close the file. See also Section 3.4, “MySQL Cluster Manager Configuration File”.

Step 3: Stop all agents. Stop the agent processes using the old binaries on all hosts making up the management installation. You can stop all agents for a given site, for example `mysite`, using the `stop agents` command in the MySQL Cluster Manager client, as shown here:

```
mcm> stop agents mysite;
```

You should execute a `stop agents` command, similar to the one just shown, for each site listed in the output of `list sites`.

Step 4: Start new MySQL Cluster Manager binaries. Start the new `mcmd` agent binaries with the `--config` option so that it uses the correct configuration file, like this:

```
$> mcmd --config=/etc/mcm/mcmd.conf &
```

Note

A majority of the agents (i.e., at least half of the total number plus one) should be started within a period of 10 seconds; otherwise, the lack of a quorum of nodes for decision making might cause the communication among the nodes to break down.

You should now be able to start the `mcm` client from the new installation and perform management tasks as usual. Once the client successfully starts and connects to the agent, you can verify that it is running the correct version of the MySQL Cluster Manager software using the `version` command, as shown here:

```
mcm> version;
+------------------------------+
| Version                      |
+------------------------------+
| MySQL Cluster Manager 9.6.0  |
+------------------------------+
1 row in set (0.00 sec)
```

Next, check that all hosts, clusters, and processes on all sites are visible to the new `mcm` client, and are operational; for example:

```
mcm> list hosts mysite;
+-----------+-----------+----------+
| Host      | Status    | Version  |
+-----------+-----------+----------+
| tonfisk   | Available | 9.6.0    |
| flundra   | Available | 9.6.0    |
| alpha     | Available | 9.6.0    |
| beta      | Available | 9.6.0    |
| gamma     | Available | 9.6.0    |
+-----------+-----------+----------+

5 rows in set (0.16 sec)

mcmc> list clusters mysite;
+------------------+-----------+
| Cluster          | Package   |
+------------------+-----------+
```

```
| mycluster        | mypackage |
| yourcluster      | mypackage |
+------------------+-----------+
2 rows in set (2.07 sec)

mcm> show status --cluster mycluster;
+-----------+-------------------+---------+
| Cluster   | Status            | Comment |
+-----------+-------------------+---------+
| mycluster | fully operational |         |
+-----------+-------------------+---------+
1 row in set (0.01 sec)

mcm> show status --cluster yourcluster;
+-------------+-------------------+---------+
| Cluster     | Status            | Comment |
+-------------+-------------------+---------+
| yourcluster | fully operational |         |
+-------------+-------------------+---------+
1 row in set (0.01 sec)

mcm> show status -r mycluster;
+--------+----------+----------+---------+-----------+-----------+
| NodeId | Process  | Host     | Status  | Nodegroup | Package   |
+--------+----------+----------+---------+-----------+-----------+
| 145    | ndb_mgmd | tonfisk  | running |           | mypackage |
| 1      | ndbdt    | tonfisk  | running | 0         | mypackage |
| 2      | ndbdt    | flundra  | running | 0         | mypackage |
| 146    | mysqld   | tonfisk  | running |           | mypackage |
| 147    | mysqld   | flundra  | running |           | mypackage |
| 148    | ndbapi   | *tonfisk | added   |           |           |
| 149    | ndbapi   | *flundra | added   |           |           |
+--------+----------+----------+---------+-----------+-----------+
7 rows in set (0.08 sec)

mcm> show status -r yourcluster;
+--------+----------+-------+---------+-----------+-----------+
| NodeId | Process  | Host  | Status  | Nodegroup | Package   |
+--------+----------+-------+---------+-----------+-----------+
| 50     | ndb_mgmd | alpha | running |           | mypackage |
| 1      | ndbd     | beta  | running | n/a       | mypackage |
| 2      | ndbd     | gamma | running | n/a       | mypackage |
+--------+----------+-------+---------+-----------+-----------+
3 rows in set (0.01 sec)
```

See Chapter 4, *Using MySQL Cluster Manager*, for more information about performing common cluster management tasks with the `mcm` client.

Upgrading MySQL NDB Cluster. Although the MySQL NDB Cluster software typically comes bundled with the MySQL Cluster Manager distribution, it is important to keep in mind that upgrading the MySQL Cluster Manager software does *not* upgrade any existing MySQL NDB Cluster installations. Since the new MySQL Cluster Manager installation uses the same configuration as the old one, the clusters under its control remain unchanged. If you wish to upgrade a cluster using the bundled MySQL NDB Cluster software, you should move the `cluster` directory (see Contents of the MySQL Cluster Manager Unix Distribution Archive) and all of its contents to a location outside the MySQL Cluster Manager installation directory. After this, you can use `add package` and `upgrade cluster` to upgrade one or more clusters to the new version of the MySQL NDB Cluster software.

Chapter 4 Using MySQL Cluster Manager

Table of Contents

This chapter discusses starting and stopping the MySQL Cluster Manager agent and client, and setting up, backing up, and restoring MySQL NDB Clusters using the MySQL Cluster Manager.

4.1 mcmd, the MySQL Cluster Manager Agent

`mcmd` is the MySQL Cluster Manager agent program. Invoking this executable starts the MySQL Cluster Manager Agent, to which you can connect using the `mcm` client (see Section 4.3, “Starting the MySQL Cluster Manager Client” and Chapter 5, *MySQL Cluster Manager Client Commands* for more information).

You can modify the behavior of the agent by specifying one or more of the options discussed in this section. Depending on the option of interest, there are up to three ways to set it:

1. Include the option directly on the command line when invoking `mcmd`. This can be done for all options that has **Yes** under the **Cmd-Line** column in **Table 4.1 MySQL Cluster Manager Agent (mcmd) Option Summary**. Some options can only be specified by this method on the command line (for example, `--config` and `--bootstrap`).

   When specifying an agent configuration option on the command line using this method, the name of the option is prefixed with two leading dash characters (`--`); for example, `--mcmd-user=johndoe`. See the *Command-Line Format* for each option given in the option description table.

2. Include the option in the agent configuration file. This can be done for all options that has **Yes** under the **Option File** column in **Table 4.1 MySQL Cluster Manager Agent (mcmd) Option Summary**. These rules apply:

   - The name of the option should *not* be prefixed with dashes, or any other characters.
   - Any hyphens in the option names should be changed to underscores.
   - The format of the configuration file follows that of the MySQL Router configuration file; see Configuration File Syntax for an explanation of the file format. An option must be specified in the correct section in the configuration file; see Table 4.1, “MySQL Cluster Manager Agent (mcmd) Option Summary” for which section each option belongs to.

- Each option must be specified on a separate line. You can comment out a line by inserting a leading hash character (#).

3. Any options configurable in the configuration file (i.e., by method 2 above), with the exception of `mcmd_password`, can be set or overridden at the `mcmd` command line by specifying the option in the format of `--section_name.option=value`, where *`section_name`* is name of the section in the configuration file that the option belongs to. For example, `--mcmd.bind_port=12345` and `--logger.level=DEBUG`.

The following table contains a summary of agent options that are read on startup by `mcmd`. More detailed information about each of these options can be found in the option descriptions.

Table 4.1 MySQL Cluster Manager Agent (mcmd) Option Summary

Name	Cmd-Line	Option File	Configuration File Section
bind_address	-	Yes	`mcmd`
bind-port	-	Yes	`mcmd`
bootstrap	Yes	-	-
config	Yes	-	-
copy-port	-	Yes	`mcmd`
core-file	Yes	-	-
data-folder	Yes	Yes	`DEFAULT`
extra-config	Yes	-	-
filename	-	Yes	`logger`, `syslog`, `filelog`, `eventlog`
help	Yes	-	-
initial	Yes	-	-
level	-	Yes	`logger`
logging_folder	-	Yes	`DEFAULT`
max_total_connections	-	Yes	`DEFAULT`
mcmd_password	-	Yes	`mcmd`
mcmd-user	-	Yes	`mcmd`
pid-file	Yes	Yes	`DEFAULT`
sinks	-	Yes	`DEFAULT`
ssl_ca	-	Yes	`mcmd`
ssl_cert	-	Yes	`mcmd`
ssl_cipher	-	Yes	`mcmd`
ssl_key	-	Yes	`mcmd`
ssl_mode	-	Yes	`mcmd`
unknown_config_option	-	Yes	`DEFAULT`
version	Yes	-	-
xcom-port	-	Yes	`mcmd`

MySQL Cluster Manager Agent (`mcmd`) Option Descriptions

The following list contains descriptions of each configuration option available for use with `mcmd`, including allowed and default values. Options with their **Type** unmentioned need only be specified in order to take effect— you should not try to set a value for them.

- `bind_address`

Type	String
Default Value	`127.0.0.1`

Specify the address for MySQL Cluster Manager client connections. Binding to a specific IPv4 or IPv6 address ensures that `mcmd` is not starting on a network interface controller (NIC) on which nothing is allowed to execute.

It is not possible to specify more than one bind address. Using `::` binds all network interfaces (IPs) on the host.

When set in the configuration file, the option should be put inside the `[mcmd]` section.

- `bind-port=#`

Type	Numeric
Default Value	`1862`
Minimum Value	`1`
Maximum Value	`65535`

Specify the port used by MySQL Cluster Manager client connections. Any valid TC/IP port number can be used. Normally, there is no need to change it from the default value (1862).

- `--bootstrap`, `-B`

Command-Line Format	`--bootstrap`

Start the agent with default configuration values, create a default one-machine cluster named `mycluster` in a site named `mysite`, and start it. This option works only if no sites have been created yet.

- `--config=`*`filename`*, `-c`

Command-Line Format	`--config=file_name`
Type	File name
Default Value	`mcmd.conf, in the MCM installation directory`

Set the file from which to read configuration options. `mcmd.conf` in the installation directory of MySQL Cluster Manager is the first default location `mcmd` looks for the file. See Section 3.4, “MySQL Cluster Manager Configuration File” for more information.

- `copy-port`

Type	Numeric
Default Value	`0`
Minimum Value	`0`
Maximum Value	`65535`

Allows you to specify the port for file copy operations. The default is 0.

- `--core-file`

Command-Line Format	`--core-file[=value]`

Type	Boolean
Default Value	1
Valid Values	1 (Write a core file) 0 (Do not write a core file)

Write a core file to the location where mcmd was started (or to another core dump location as specified by the operating system) when mcmd quits unexpectedly. The default value is 1.

- --data-folder=*dir_name*, -d *dir_name*

Command-Line Format	--data-folder=dir_name
Type	Directory name
Default Value	mcm_data in the parent directory of the MCM installation directory

Set the location of the agent repository, which contains collections of MySQL Cluster Manager data files and MySQL NDB Cluster configuration and data files. The value must be a valid absolute path.

The default location is mcm_data in the parent directory of the MySQL Cluster Manager installation directory. If you change the default, you should use a standard location external to the MySQL Cluster Manager installation directory, such as /var/opt/mcm on Linux.

In addition to the data files for all the clusters under the control of MySQL Cluster Manager, the data-folder also contains a rep directory in which mcmd configuration and metadata are kept. Normally, there is no need to interact with these directories beyond specifying the --data-folder option; see exceptions in, for example, Section 4.8, “Restoring a MySQL Cluster Manager Agent with Data from Other Agents” and Section 4.7, “Backing Up and Restoring MySQL Cluster Manager Agents”.

- --extra-config=*filename* , -a

Command-Line Format	--extra-config=file_name
Type	File name

Read this file after configuration files are read from either default locations or from the location specified by the --config option. Multiple --extra-config options can be passed, and the files are loaded in the order they are specified.

- filename=*filename*

Type	File name
Default Value	mcmd.log

Set the name of the file to write the log to. The default is mcmd.log, located in the same directory where the MySQL Cluster Manager installation directory is found. The location of this file can be overridden with the logging_folder option.

- --help, -?

Command-Line Format	--help

mcmd help output provides information on some file paths and all the program options; it also provides some usage examples of the mcmd command:

```
$> mcmd --help
MySQL Cluster Manager  v9.6.0 on linux-glibc2.17 (64-bit) (MySQL Enterprise - Commercial)
Copyright (c) 2007, 2026, Oracle and/or its affiliates.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Configuration read from the following files in the given order (enclosed
in parentheses means not available for reading):
  (/opt/mcm9.6.0/bin/.././mcmd.conf)
  (/opt/mcm9.6.0/bin/.././mcmd.ini)
  (/home/johndoe/.mcmd.conf)
  (/home/johndoe/.mcmd.ini)
Plugins Path:
  /opt/mcm9.6.0/lib/mysqlrouter

Default Log Directory:
  /opt

Default Persistent Data Directory:
  /opt/mcm_data

Default Runtime State Directory:
  /opt


# Usage

mcmd (-?|--help)

mcmd (-V|--version)

mcmd [-B|--bootstrap] [-c|--config=<path>] [--core-file=[<0|1>]]
     [-d|--data-folder=<directory>] [-a|--extra-config=<path>]
     [-i|--initial] [--pid-file=<pidfile>]

mcmd [--bind-port=<portnumber>] [--copy-port=<portnumber>]
     [--mcmd-user=<username>] [--xcom-port=<portnumber>]

# Options

  -B, --bootstrap
        Bootstrap a MySQL Cluster using MySQL Cluster Manager
  -c <path>, --config <path>
        Only read configuration from given file.
  --core-file [ <0|1>]
        Write a core file if mcmd dies.
  -d <directory>, --data-folder <directory>
        Data directory for MySQL Cluster Manager.
  -a <path>, --extra-config <path>
        Read this file after configuration files are read from either
        default locations or from files specified by the --config
        option.
  -?, --help
        Display this help and exit.
  -i, --initial
        Reinitialize configuration metadata directory
  --pid-file <pidfile>
        Path and filename of pid file
  -V, --version
        Display version information and exit.
  --bind-port <portnumber>
        Portnumber to use for mcmd. [DEPRECATED]
  --copy-port <portnumber>
        Portnumber to use for file copying. [DEPRECATED]
  --mcmd-user <username>
        The username used to access MySQL Cluster Manager [DEPRECATED]
  --xcom-port <portnumber>
        Portnumber to use for XCOM. [DEPRECATED]

# Examples

Bootstrap a NDB cluster on localhost
```

```
    mcmd --bootstrap

Bootstrap a NDB cluster with a specified mcm_data directory

    mcmd --bootstrap -d my_mcm_data

Start mcmd with additional config file options on cmdline

    mcmd --logger.level=NOTE --mcmd.copy_port=12345
```

- `--initial`, `-i`

Command-Line Format	`--initial`

After making a backup of the agent's configuration store (`mcm_data/rep/`) like the `backup agents` command would do for the local host, wipe the configuration store's contents before starting `mcmd`. The agent's configuration is then recovered from other agents. This is useful when an agent has fallen into an inconsistent state and cannot be properly restarted.

- `level=`*`level`*

Type	Enumeration
Default Value	`info`

Valid Values	`fatal`
	`system`
	`error`
	`warning`
	`info`
	`note`
	`debug`

Sets the `mcmd` log severity level. Possible values for this option and their descriptions are listed in Table 4.2, “MySQL Cluster Manager Agent Log Levels” in descending level of severity. When the option is set to a certain severity level, all events of that or higher levels are logged. `info` is the default log level, and is the recommended setting for a production environment; running on a more severe log level produces fewer messages and makes it harder to trace a problem when it occurs.

Table 4.2 MySQL Cluster Manager Agent Log Levels

Level of Severity	Description
`fatal`	Conditions that should be corrected immediately, such as a corrupted MySQL Cluster Manager data repository
`system`	Informational messages about the product
`error`	Conditions that should be corrected, such as configuration errors
`warning`	Conditions that do not fail executions, but may require user attention
`info`	Messages on main events of the site and from command execution (default)
`note`	Informational messages to provide users with some execution details

Level of Severity	Description
debug	Debugging messages that give execution details useful for developers. This causes large log files if used over a long period of time.

While the setting of the level option is applied only to the host whose mcmd agent uses the option, the change log-level client command can be used to apply the logging level to an entire management site or to specific hosts.

- logging_folder=*dir_name*

Type	Directory name
Default Value	The parent directory of the MCM installation directory

Path to the mcmd log file directory. The default is the parent directory of the MySQL Cluster Manager installation directory.

- max_total_connections

Type	Integer
Default Value	512
Minimum Value	1
Maximum Value	9223372036854775807

The maximum number of client connections to mcmd. Setting this option helps prevent mcmd from running out of file descriptors. This is similar to MySQL Server's max_connections system variable.

The default value is 512, and the option is set in the [DEFAULT] section of the configuration file.

- mcmd_password=*password*

The option serves the following two purposes:

- Sets the user password for an mcm client to connect to the mcmd agent with the user name set by --mcmd-user. The client must supply the same value using the --password client option when trying to connect to the agent.
- Sets a password for the MySQL account to be used by the mcmd agent to access the SQL nodes. When an SQL node is initialized, the mcmd agent creates a new MySQL user account on it using the user name set by the --mcmd-user option and the password set by this option. See descriptions of --mcmd-user for more details about the account.
- When using TLS connections for NDB clusters, the password is also used as the certificate passphrase. If you changed the password, you must also update manually the certificate passphrase, or the TLS connections will fail. Alternatively, if the situation allows, you can disable TLS connections for the cluster, recreate the certificates, and then reanable TLS connections.

The option can only be set in the [mcmd] section of configuration file, not on the command line.

This option must be specified, including for bootstrapping, or mcmd fails to start.

- `mcmd-user=user_name`

  The option serves the following two purposes:

  - Sets the user name for an `mcm` client to connect to the `mcmd` agent. If the option is not specified, the default value of `mcmd` is used. If the option is specified with another value, the client must supply it using the `--user` client option when trying to connect to the agent.

    The password for using this user name to connect to the agent is set with the `mcmd_password` option.

  - Sets a user name for the MySQL account to be used by the `mcmd` agent to access the SQL nodes. When an SQL node is initialized, the `mcmd` agent creates a new MySQL user account on it using the user name set by the option and the password set by the `mcmd_password` option. *This account is created with all privileges on the MySQL server including the granting of privileges*. In other words, it is created as if you had executed `GRANT ALL PRIVILEGES ON *.* ... WITH GRANT OPTION` in the `mysql` client. The existing MySQL `root` account is not altered in such cases, and the default `test` database is preserved.

  If the option is not specified, the default value of `mcmd` is used.

- `--pid-file=file`

Command-Line Format	`--pid-file=file_name`
Type	File name
Default Value	`mcmd.pid`

  Set the name and path to a process ID (`.pid`) file. Not normally used or needed. This option is not supported on Windows systems.

- `sinks`

Type	String

Valid Values (Windows)	`consolelog`
	`filelog`
	`eventlog`

Valid Values (Other)	`consolelog`
	`filelog`
	`syslog`

  The different logging methods used by `mcmd`.

  Supported sink values are: `consolelog`, `filelog`, `eventlog` (on Windows only), and `syslog` (on Unix-based systems only). Use a comma-separated list to define multiple values. If you have multiple sinks defined, they can be customized under the corresponding sections of `[syslog]`, `[filelog]`, and `[eventlog]` in the configuration file using the `filename` and `level` options.

  Default value: `filelog` if the `logging_folder` option is not empty in the `[DEFAULT]` section, and `consolelog` otherwise.

- `ssl_ca`

Type	File name

Default Value	NULL

The path name of the Certificate Authority (CA) certificate file in PEM format. The file contains a list of trusted SSL Certificate Authorities.

- ssl_cert

Type	File name
Default Value	NULL

The path name of the SSL public key certificate file in PEM format.

If ssl_cert is set to a certificate that uses any restricted cipher or cipher category, mcmd starts with support for encrypted connections disabled. For information about cipher restrictions, see Connection Cipher Configuration.

- ssl_cipher

Type	String
Default Value	NULL

The list of permissible encryption ciphers for connections that use TLS protocol TLSv1.2. If no cipher in the list is supported, encrypted connections that use these TLS protocols do not work.

For greatest portability, the cipher list should be a list of one or more cipher names, separated by colons. The following example shows two cipher names separated by a colon:

```
[mcmd]
ssl_cipher="DHE-RSA-AES128-GCM-SHA256:AES128-SHA"
```

OpenSSL supports the syntax for specifying ciphers described in the OpenSSL documentation at https://www.openssl.org/docs/manmaster/man1/ciphers.html.

For information about which encryption ciphers MySQL supports, see Encrypted Connection TLS Protocols and Ciphers.

- ssl_key

Type	File name
Default Value	NULL

The path name of the server SSL private key file in PEM format. For better security, use a certificate with an RSA key size of at least 2048 bits.

- ssl_mode

Type	Enumeration
Default Value	PREFERRED
Valid Values	DISABLED REQUIRED

	PREFERRED

ssl_mode sets the security state of the connections. The possible values are as follows:

DISABLED	Establish an unencrypted connection (the default if certificate and key have not been set).
REQUIRED	Establish a secure connection if the secure connections are supported by the target of connection.
PREFERRED	Establish an encrypted connection if the target of connection supports encrypted connections, falling back to an unencrypted connection if an encrypted connection cannot be established. This is the default if ssl_mode is not specified.

- unknown_config_option=*string*

Type	String
Default Value	error
Valid Values	error warning

Determines the behavior for handling unknown configuration options, such as those containing typos. Here are the values this option takes:

- error (default): mcmd returns an error when an unknown option is encountered
- warning: mcmd prints a warning when an unknown option is encountered and continues with starting

MySQL Cluster Manager versions before 8.0.29 ignore any unknown configuration options.

- --version, -V

Command-Line Format	--version

Display version information and exit. Output may vary according to the MySQL Cluster Manager software version, operating platform, and versions of libraries used on your system, but should closely resemble what is shown here, with the first line of output containing the MySQL Cluster Manager release number:

```
$> mcmd  --version
MySQL Cluster Manager  v9.6.0 on linux-glibc2.17 (64-bit) (MySQL Enterprise - Commercial)
```

- xcom-port

Type	Numeric
Default Value	18620
Minimum Value	1
Maximum Value	65535

Specify the XCOM port. The default in 18620.

4.2 Starting and Stopping the MySQL Cluster Manager Agent

Before you can start using MySQL Cluster Manager to create and manage a MySQL NDB Cluster, the MySQL Cluster Manager agent must be started on each computer that is intended to host one or more nodes in the MySQL NDB Cluster to be managed.

The MySQL Cluster Manager agent employs a MySQL user account for administrative access to `mysqld` processes. It is possible, but not a requirement, to change the default user name, the default password used for this account, or both. For more information, see Section 3.3.3, “Setting the MySQL Cluster Manager Agent User Name and Password”.

4.2.1 Starting and Stopping the Agent on Linux

To start the MySQL Cluster Manager agent on a given host running a Linux or similar operating system, you should run `mcmd`, found in the `bin` directory within the manager installation directory on that host. Typical options used with `mcmd` are shown here:

```
mcmd [--config=filename | --bootstrap]
```

See Section 4.1, “mcmd, the MySQL Cluster Manager Agent”, for information about additional options that can be used when invoking `mcmd` from the command line, or in a configuration file.

`mcmd` normally runs in the foreground. If you wish, you can use your platform's usual mechanism for backgrounding a process. On a Linux system, you can do this by appending an ampersand character (`&`), like this (not including any options that might be required):

```
$> ./bin/mcmd &
```

By default, the agent assumes that the agent configuration file is `mcmd.conf` in the MySQL Cluster Manager installation directory (see Section 3.4, “MySQL Cluster Manager Configuration File” for more details). You can tell the agent to use a different configuration file by passing the path to this file to the `--config` option, as shown here:

```
$> ./bin/mcmd --config=/home/mcm/mcm-agent.conf
```

The `--bootstrap` option causes the agent to start with default configuration values, create a default one-machine cluster named `mycluster`, and start it. This option works only if no sites have been created yet.

The use of the `--bootstrap` option with `mcmd` is shown here on a system having the host name `torsk`, where MySQL Cluster Manager has been installed to `/home/jon/mcm`:

```
$> ./mcmd --bootstrap
logging facility initialized, switching logging to loggers specified in configuration
MySQL Cluster Manager 9.6.0 (64bit) started
Connect to MySQL Cluster Manager by running "/home/clusteradmin/mcm9.6.0/bin/mcm" -h torsk -P 1862
Configuring default cluster 'mycluster'...
Setting default_storage_engine to ndbcluster...
Starting default cluster 'mycluster' version '9.6.0-cluster'...
Cluster 'mycluster' started successfully
 ndb_mgmd           torsk:1186
 ndbmtd  torsk
 ndbmtd  torsk
 mysqld  torsk:3306
 mysqld  torsk:3307
 ndbapi  *
Connect to the database by running "/home/clusteradmin/cluster/bin/mysql" -h 127.0.0.1 -P 3306 -u root
```

You can then connect to the agent using the `mcm` client (see Section 4.3, “Starting the MySQL Cluster Manager Client”), and to either of the MySQL Servers running on ports 3306 and 3307 using `mysql` or another MySQL client application.

See Section 4.1, “mcmd, the MySQL Cluster Manager Agent”, for more information about options that can be used with `mcmd`.

The MySQL Cluster Manager agent must be started on each host in the MySQL NDB Cluster to be managed.

To stop one or more instances of the MySQL Cluster Manager agent, use the `stop agents` command in the MySQL Cluster Manager client. If the client is unavailable, you can stop each agent process using the system's standard method for doing so, such as `^C` or `kill`.

You can also set the agent up as a daemon or service on Linux and other Unix-like systems. (See Section 3.3.1, "Installing MySQL Cluster Manager on Unix-like Platforms".) If you also want data node failed processes from a running MySQL NDB Cluster to be started when the agent fails and restarts in such cases, you must make sure that `StopOnError` is set to 0 on each data node (and not to 1, the default).

4.2.2 Starting and Stopping the MySQL Cluster Manager Agent on Windows

To start the MySQL Cluster Manager agent manually on a Windows host, you should invoke `mcmd.exe`, found in the `bin` directory under the manager installation directory on that host. If the configuration file's location is not specified with the `--config` option, mcmd looks for the file at its default locations (see Section 3.4, "MySQL Cluster Manager Configuration File" for details).

Typical options for `mcmd` are shown here:

```
mcmd[.exe] [--config=filename | --bootstrap]
```

For information about additional options that can be used with `mcmd` on the command line or in an option file, see Section 4.1, "`mcmd`, the MySQL Cluster Manager Agent".

By default, the agent assumes that the agent configuration file is `mcmd.conf` in the MySQL Cluster Manager installation directory (see Section 3.4, "MySQL Cluster Manager Configuration File" for more details). You can tell the agent to use a different configuration file by passing the path to this file to the `--config` option, as shown here:

```
C:\Program Files (x86)\MySQL\MySQL Cluster Manager 9.6.0\bin>
  mcmd --config="C:\Program Files (x86)\MySQL\MySQL Cluster Manager 9.6.0\etc\mcmd.ini"
```

The `--bootstrap` option causes the agent to start with default configuration values, create a default one-machine cluster named `mycluster`, and start it. The use of this option with `mcmd` is shown here on a system having the host name `torsk`, where MySQL Cluster Manager has been installed to the default location:

```
C:\Program Files\MySQL\MySQL Cluster Manager 9.6.0\bin>mcmd --bootstrap
MySQL Cluster Manager 9.6.0 started
Connect to MySQL Cluster Manager by running "C:\Program Files\MySQL\MySQL Cluster Manager 9.6.0\bin\mcm" -h
Configuring default cluster 'mycluster'...
Starting default cluster 'mycluster'...
Cluster 'mycluster' started successfully
        ndb_mgmd        TORSK:1186
        ndbd            TORSK
        ndbd            TORSK
        mysqld          TORSK:3306
        mysqld          TORSK:3307
        ndbapi          *
Connect to the database by running "C:\Program Files\MySQL\MySQL Cluster Manager 9.6.0\cluster\bin\mysql" -
```

You can then connect to the agent using the `mcm` client (see Section 4.3, "Starting the MySQL Cluster Manager Client"), and to either of the MySQL Servers running on ports 3306 and 3307 using `mysql` or another MySQL client application.

When starting the MySQL Cluster Manager agent for the first time, you may see one or more Windows **Security Alert** dialogs. You should grant permission to connect to private networks for any of the programs `mcmd.exe`, `ndb_mgmd.exe`, `ndbd.exe`, `ndbmtd.exe`, or `mysqld.exe`. To do so, check the **Private Networks...** box and then click the **Allow access** button. It is generally not necessary to grant MySQL Cluster Manager or MySQL NDB Cluster access to public networks such as the Internet.

See Section 4.1, "`mcmd`, the MySQL Cluster Manager Agent", for more information about options that can be used with `mcmd`.

The MySQL Cluster Manager agent must be started on each host in the MySQL NDB Cluster to be managed.

It is possible to install MySQL Cluster Manager as a Windows service, so that it is started automatically each time Windows starts. See Section 3.3.2.1, “Installing the MySQL Cluster Manager Agent as a Windows Service”.

To stop one or more instances of the MySQL Cluster Manager agent, use the `stop agents` command in the MySQL Cluster Manager client. You can also stop an agent process using the Windows Task Manager. In addition, if you have installed MySQL Cluster Manager as a Windows service, you can stop (and start) the agent using the Windows Service Manager, `CTRL-C`, or the appropriate `SC STOP` (or `SC START`) or `NET STOP` (or `NET START`) command. See Starting and stopping the MySQL Cluster Manager agent Windows service, for more information about each of these options.

4.3 Starting the MySQL Cluster Manager Client

This section covers starting the MySQL Cluster Manager client and connecting to the MySQL Cluster Manager agent.

MySQL Cluster Manager 9.6.0 includes a command-line client `mcm`, located in the installation `bin` directory. `mcm` can be invoked with any one of the options shown in the following table (see Connecting to the agent with the `mcm` client for detailed descriptions of some of the options):

Table 4.3 mcm options

Long form	Short form	Description
`--help`	`-?`	Display `mcm` client options
`--host=<hostname>`	`-h <hostname>`	Host to use when connecting to `mcmd`
`--user=<username>`	`-u <username>`	The user name for connecting to the agent
`--password[=<password>]`	`-p[<password>]`	The password for connecting to the agent
`--port=<portnum>`	`-P <portnum>`	Optional port to use when connecting to `mcmd`
`--version`	`-V`	Shows MySQL Cluster Manager agent/client version

The client/server protocol used by MySQL Cluster Manager is platform-independent. You can connect to any MySQL Cluster Manager agent with an `mcm` client on any platform where it is available. This means, for example, that you can use an `mcm` client on Microsoft Windows to connect to a MySQL Cluster Manager agent that is running on a Linux host.

You can also use the `mysql` client to run MySQL Cluster Manager client sessions on platforms where `mcm` itself (or even `mcmd`) is not available. For more information, see Connecting to the agent using the `mysql` client.

If you experience problems starting an MySQL Cluster Manager client session because the client fails to connect, see Can't connect to [local] MySQL server, for some reasons why this might occur, as well as suggestions for some possible solutions.

To end a client session, use the `exit` or `quit` command (short form: `\q`). Neither of these commands requires a separator or terminator character.

For more information, see Chapter 5, *MySQL Cluster Manager Client Commands*.

Connecting to the agent with the `mcm` client. You can connect to the MySQL Cluster Manager agent by invoking `mcm` (or, on Windows, `mcm.exe`). You may also need to specify one or more of the following command-line options:

- `--host`=*`hostname`* or `-h`[ ]*`hostname`*

  This option takes the name or IP address of the host to connect to. The default is `localhost` (which may not be recognized on all platforms when starting a `mcm` client session even if it works for starting `mysql` client sessions).

  You should keep in mind that the `mcm` client does not perform host name resolution; any name resolution information comes from the operating system on the host where the client is run. For this reason, it is usually best to use a numeric IP address rather than a hostname for this option.

- `--port`=*`portnumber`* or `-P`[ ]*`portnumber`*

  This option specifies the TCP/IP port for the client to use. This must be the same port that is used by the MySQL Cluster Manager agent. As mentioned elsewhere, if no agent port is specified in the MySQL Cluster Manager agent configuration file (`mcmd.ini`), the default number of the port used by the MySQL Cluster Manager agent is 1862, which is also used by default by `mcm`.

- `--user`=*`username`* or `-u`[ ]*`username`*

  The option specifies the user name for connecting to the agent. The default value of "`mcmd`" is used if the option is not specified. To connect successfully, the value of the option must match that specified by the `mcmd` configuration option `mcmd-user` of the agent you are connecting to, which is also "`mcmd`" by default.

- `--password`[=*`password`*] or `-p`[*`password`*]

  The option specifies the password for connecting to the agent. To connect successfully, the value of the option must match that specified by the `mcmd` configuration option `mcmd_password` of the agent you are connecting to.

  If you use the short option form (`-p`), you *must not* leave a space between this option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, the `mcm` client prompts you for one.

  Specifying a password on the command line should be considered insecure. It is preferable that you either omit the password when invoking the client and then supply it when prompted, or put the password in a startup script or configuration file.

  This option must be specified for the client to connect to the agent.

`mcm` accepts additional `mysql` client options, some of which may possibly be of use for MySQL Cluster Manager client sessions. For example, the `--pager` option might prove helpful when the output of `get` contains too many rows to fit in a single screen. The `--prompt` option can be used to provide a distinctive prompt to help avoid confusion between multiple client sessions. However, options not shown in the current manual have not been extensively tested with `mcm` and so cannot be guaranteed to work correctly (or even at all). See mysql Client Options, for a complete listing and descriptions of all `mysql` client options.

Note

Like the `mysql` client, `mcm` also supports `\G` as a statement terminator, which causes the output to be formatted vertically. This can be helpful when using a terminal whose width is restricted to some number of (typically 80) characters. See Chapter 5, *MySQL Cluster Manager Client Commands*, for examples.

Connecting to the agent using the `mysql` client. A `mysql` client from any MySQL distribution should work without any issues for connecting to `mcmd`. In addition, since the client/server protocol

used by MySQL Cluster Manager is platform-independent, you can use a `mysql` client on any platform supported by MySQL. (This means, for example, that you can use a `mysql` client on Microsoft Windows to connect to a MySQL Cluster Manager agent that is running on a Linux host.) Connecting to the MySQL Cluster Manager agent using the `mysql` client is accomplished by invoking `mysql` and specifying a hostname, port number, username and password using the following command-line options:

- `--host=`*`hostname`* or `-h` *`hostname`*

  This option takes the name or IP address of the host to connect to. The default is `localhost`. Like the `mcm` client, the `mysql` client does not perform host name resolution, and relies on the host operating system for this task. For this reason, it is usually best to use a numeric IP address rather than a hostname for this option.

- `--port=`*`portnumber`* or `-P` *`portnumber`*

  This option specifies the TCP/IP port for the client to use. This must be the same port that is used by the MySQL Cluster Manager agent. Although the default number of the port used by the MySQL Cluster Manager agent is 1862 (which is also used by default by `mcm`), *this default value is not known to the `mysql` client*, which uses port 3306 (the default port for the MySQL server) if this option is not specified when `mysql` is invoked.

  Thus, you *must* use the `--port` or `-P` option to connect to the MySQL Cluster Manager agent using the `mysql` client, *even if the agent process is using the MySQL Cluster Manager default port*, and even if the agent process is running on the same host as the `mysql` client. Unless the correct agent port number is supplied to it on startup, `mysql` is unable to connect to the agent.

- `--user=`*`username`* or `-u` *`username`*

  The option specifies the user name for connecting to the agent. By default, the `mysql` client tries to use the name of the current system user on Unix systems and “ODBC” on Windows, so you *must* supply this option and the username when trying to access the MySQL Cluster Manager agent with the `mysql` client; otherwise, `mysql` cannot connect to the agent.

  To connect successfully, the value of the option must match that specified by the `mcmd` configuration option `--mcmd-user` of the agent you are connecting to, which is “`mcmd`” by default.

- `--password`[=*`password`*] or `-p`[*`password`*]

  The option specifies the password for connecting to the agent. If you do not include the `--password` or `-p` option when invoking `mysql`, it cannot connect to the agent. To connect successfully, the value of the option must match that specified by the `mcmd` configuration option `mcmd_password` of the agent you are connecting to, which is “`super`” by default.

  If you use the short option form (`-p`), you *must not* leave a space between this option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, the `mysql` client prompts you for one.

  Specifying a password on the command line should be considered insecure. It is preferable that you either omit the password when invoking the client and then supply it when prompted, or put the password in a startup script or configuration file.

In addition, you can use the `--prompt` option to set the `mysql` client's prompt. This is recommended, since allowing the default prompt (`mysql>`) to be used could lead to confusion between a MySQL Cluster Manager client session and a MySQL client session.

Thus, you can connect to a MySQL Cluster Manager agent by invoking the `mysql` client on the same machine from the system shell in a manner similar to what is shown here.

```
$> mysql -h127.0.0.1 -P1862 -umcmd -p --prompt='mcm> '
```

For convenience, on systems where `mcm` itself is not available, you might even want to put this invocation in a startup script. On a Linux or similar system, this script might be named `mcm-client.sh`, with contents similar to what is shown here:

```
#!/bin/sh
/usr/local/mysql/bin/mysql -h127.0.0.1 -P1862 -umcmd -p --prompt='mcm> '
```

In this case, you could then start up a MySQL Cluster Manager client session using something like this in the system shell:

```
$> ./mcm-client
```

On Windows, you can create a batch file with a name such as `mcm-client.bat` containing something like this:

```
C:\mysql\bin\mysql.exe -umcmd -psuper -h localhost -P 1862 --prompt="mcm> "
```

(Adjust the path to the `mysql.exe` client executable as necessary to match its location on your system.)

If you save this file to a convenient location such as the Windows desktop, you can start a MySQL Cluster Manager client session merely by double-clicking the corresponding file icon on the desktop (or in Windows Explorer); the client session opens in a new `cmd.exe` (DOS) window.

4.4 Setting Up MySQL NDB Clusters with MySQL Cluster Manager

This section provides basic information about setting up a new MySQL NDB Cluster with MySQL Cluster Manager. It also supplies guidance on migration of an existing MySQL NDB Cluster to MySQL Cluster Manager.

For more information about obtaining and installing the MySQL Cluster Manager agent and client software, see Chapter 3, *MySQL Cluster Manager Installation, Configuration, Cluster Setup*.

See Chapter 5, *MySQL Cluster Manager Client Commands*, for detailed information on the MySQL Cluster Manager client commands shown in this chapter.

4.4.1 Creating a MySQL NDB Cluster with MySQL Cluster Manager

In this section, we discuss the procedure for using MySQL Cluster Manager to create and start a new MySQL NDB Cluster. We assume that you have already obtained the MySQL Cluster Manager and MySQL NDB Cluster software, and that you are already familiar with installing MySQL Cluster Manager (see Chapter 3, *MySQL Cluster Manager Installation, Configuration, Cluster Setup*).

MySQL Cluster Manager also supports importing existing, standalone MySQL NDB Clusters; for more information, see Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”.

We also assume that you have identified the hosts on which you plan to run the cluster and have decided on the types and distributions of the different types of nodes among these hosts, as well as basic configuration requirements based on these factors and the hardware characteristics of the host machines.

Note

You can create and start a MySQL NDB Cluster on a single host for testing or similar purposes, simply by invoking `mcmd` with the `--bootstrap` option. See Section 4.2, “Starting and Stopping the MySQL Cluster Manager Agent”.

Creating a new cluster consists of the following tasks:

- **MySQL Cluster Manager agent installation and startup.** Install the MySQL Cluster Manager software distribution, make any necessary edits of the agent configuration files, and start the agent processes as explained in Chapter 3, *MySQL Cluster Manager Installation, Configuration, Cluster Setup*. Agent processes must be running on all cluster hosts before you can create a cluster. This

means that you need to place a complete copy of the MySQL Cluster Manager software distribution on every host. The MySQL Cluster Manager software does not have to be in a specific location, or even the same location on all hosts, but it must be present; you cannot manage any cluster processes hosted on a computer where `mcmd` is not also running.

- **MySQL Cluster Manager client session startup.** Starting the MySQL Cluster Manager client and connect to the MySQL Cluster Manager agent. You can connect to an agent process running on any of the cluster hosts, using the `mcm` client on any computer that can establish a network connection to the desired host. See Section 4.3, "Starting the MySQL Cluster Manager Client", for details.

  On systems where `mcm` is not available, you can use the `mysql` client for this purpose. See Connecting to the agent using the `mysql` client.

- **MySQL NDB Cluster software deployment.** The simplest and easiest way to do this is to copy the complete MySQL NDB Cluster distribution to the same location on every host in the cluster. (If you have installed MySQL Cluster Manager 9.6.0 on each host, the MySQL NDB Cluster 9.6.0 distribution is already included, in *`mcm_installation_dir`*`/cluster`.) If you do not use the same location on every host, be sure to note it for each host. Do not yet start any MySQL NDB Cluster processes or edit any configuration files; when creating a new cluster, MySQL Cluster Manager takes care of these tasks automatically.

  On Windows hosts, you should *not* install as services any of the MySQL NDB Cluster node process programs, including `ndb_mgmd.exe`, `ndbd.exe`, `ndbmtd.exe`, and `mysqld.exe`. MySQL Cluster Manager manages MySQL NDB Cluster processes independently of the Windows Service Manager and does not interact with the Service Manager or any Windows services when doing so.

  Note

  You can actually perform this step at any time up to the point where the software package is registered (using `add package`). However, we recommend that you have all required software—including the MySQL NDB Cluster software—in place before executing any MySQL Cluster Manager client commands.

- **Management site definition.** Using the `create site` command in the MySQL Cluster Manager client, define a MySQL Cluster Manager management site—that is, the set of hosts to be managed. This command provides a name for the site, and must reference all hosts in the cluster. Section 5.2.6, "The `create site` Command", provides syntax and other information about this command. To verify that the site was created correctly, use the MySQL Cluster Manager client commands `list sites` and `list hosts`.

- **MySQL NDB Cluster software package registration.** In this step, you provide the location of the MySQL NDB Cluster software on all hosts in the cluster using one or more `add package` commands. To verify that the package was created correctly, use the `list packages` and `list processes` commands.

- **Cluster definition.** Execute a `create cluster` command to define the set of MySQL NDB Cluster nodes (processes) and hosts on which each cluster process runs, making up a the MySQL NDB Cluster. This command also uses the name of the package registered in the previous step so that MySQL Cluster Manager knows the location of the binary running each cluster process. You can use the `list clusters` and `list processes` commands to determine whether the cluster has been defined as desired.

  If you wish to use SQL node connection pooling, see Setting up `mysqld` connection pooling before creating the cluster.

- **Initial configuration.** Perform any configuration of the cluster that is required or desired prior to starting it. You can set values for MySQL Cluster Manager configuration attributes (MySQL NDB Cluster parameters and MySQL Server options) using the MySQL Cluster Manager client `set`

command. You do not need to edit any configuration files directly—in fact, you should *not* do so. Keep in mind that certain attributes are read-only, and that some others cannot be reset after the cluster has been started for the first time. You can use the `get` command to verify that attributes have been set to the correct values.

- **Cluster startup.** Once you have completed the previous steps, including necessary or desired initial configuration, you are ready to start the cluster. The `start cluster` command starts all cluster processes in the correct order. You can verify that the cluster has started and is running normally after this command has completed, using the MySQL Cluster Manager client command `show status`. At this point, the cluster is ready for use by MySQL NDB Cluster applications.

4.5 Importing MySQL NDB Clusters into MySQL Cluster Manager

It is possible to bring a “wild” MySQL NDB Cluster—that is, a cluster not created using MySQL Cluster Manager—under the control of MySQL Cluster Manager. The following sections provide an outline of the procedure required to import such a cluster into MySQL Cluster Manager, followed by a more detailed example.

4.5.1 Importing a Cluster Into MySQL Cluster Manager: Basic Procedure

The importing process consists generally of the steps listed here:

1. Prepare the “wild” cluster for migration.
2. Verify PID files for cluster processes.
3. Create and configure in MySQL Cluster Manager a “target” cluster whose configuration matches that of the “wild” cluster.
4. Perform a test run, and then execute the `import cluster` command.

This expanded listing breaks down each of the tasks just mentioned into smaller steps:

1. *Prepare the “wild” cluster for migration*

   a. It is highly recommended that you take a complete backup of the “wild” cluster before you make changes to it, using the `ndb_mgm` client. For more information, see Using The NDB Cluster Management Client to Create a Backup.

   b. Any cluster processes that are under the control of the system's boot-time process management facility, such as `/etc/init.d` on Linux systems or the Services Manager on Windows platforms, should be removed from its control.

   c. The wild cluster's configuration must meet the following requirements, and it should be reconfigured and restarted if it does not:

      - `NodeID` must be assigned for every node.
      - `DataDir` must be specified for each management and data node, and the data directories for different nodes cannot overlap with each other.
      - A “free” API node not bounded to any host must be provisioned, through which the `mcmd` agent can communicate with the cluster.

   d. Create a MySQL user named `mcmd` on each SQL node, and grant root privileges to the user.

   e. Make sure that the configuration cache is disabled for each management node. Since the configuration cache is enabled by default, unless the management node has been started with the `--config-cache=false` option, you will need to stop and restart it with that option, in addition to other options that it has been started with previously.

2. *Verify cluster process PID files*.

a. Verify that each process in the “wild” cluster has a valid PID file.

b. If a given process does not have a valid PID file, you must create one for it.

See Section 4.5.2.2, “Verify All Cluster Process PID Files”, for a more detailed explanation and examples.

3. *Create and configure “target” cluster under MySQL Cluster Manager control*

a. Install MySQL Cluster Manager and start `mcmd` on all hosts with the same system user who started the wild cluster processes.

b. Create a MySQL Cluster Manager site encompassing these hosts, using the `create site` command.

c. Add a MySQL Cluster Manager package referencing the MySQL NDB Cluster binaries, using the `add package` command. Use this command's `--basedir` option to point to the location of the MySQL NDB Cluster installation directory.

d. Create the target cluster using the `create cluster` command, including the same processes and hosts used by the wild cluster. Use the command's `--import` option to specify that the cluster is a target for import.

If the wild cluster adheres to the recommendation for node ID assignments given in the description for the `create cluster` command, you need not specify the node IDs for the processes in the `create cluster` command.

Also, this step may be split into a `create cluster` command followed by one or more `add process` commands (see Section 4.5.2.3, “Creating and Configuring the Target Cluster”).

e. For importing a cluster that uses TLS connections, perform the following steps (see TLS Link Encryption for NDB Cluster and Section 4.11, “Using TLS Connections for NDB Clusters” for details):

- Copy a set of API certificates and its private key to the default certificate directory on every host for `mcmd` to use.
- Set `--ndb-tls-search-path` for all the processes to the correct folders.
- Set `RequireTls`, `RequireCertificate`, and `--ndb-mgm-tls` for the processes to their respective values

f. Use `import config` to copy the wild cluster's configuration data into the target cluster. Use this command's `--dryrun` option (short form: `-y`) to perform a test run that merely logs the configuration information the command copies when it is executed without the option.

If any `ndb_mgmd` or `mysqld` processes in the wild cluster are running on ports other than the default, you must first perform `set` commands to assign the correct port numbers for them in the target cluster. When all such processes are running on the correct ports and the dry run is successful, you can execute `import config` (without the `--dryrun` option) to copy the wild cluster's configuration data. Following this step, you should check the log as well as the configuration of the target cluster to ensure that all configuration attributes were copied correctly and with the correct scope. Correct any inconsistencies with the wild cluster's configuration using the appropriate `set` commands.

4. *Test and perform migration of wild cluster*.

   a. Perform a test run of the proposed migration using `import cluster` with the `--dryrun` option, which causes MySQL Cluster Manager to check for errors, but not actually migrate any processes or data.

   b. Correct any errors found using `--dryrun`. Repeat the dry run from the previous step to ensure that no errors were missed.

   c. When the dry run no longer reports any errors, you can perform the migration using `import cluster`, but without the `--dryrun` option.

4.5.2 Importing a Cluster Into MySQL Cluster Manager: Example

As discussed previously (see Section 4.5.1, “Importing a Cluster Into MySQL Cluster Manager: Basic Procedure”), importing a standalone or “wild” cluster that was not created with MySQL Cluster Manager into the manager requires the completion of four major tasks. The example provided over the next few sections shows all the steps required to perform those tasks.

Sample cluster used in example. The “wild” cluster used in this example consists of four nodes—one management node, two data nodes, and one SQL node. Each of these nodes resides on one of three hosts, the IP address for each is shown in the following table:

Table 4.4 Nodes in the example cluster

Node type (executable)	Host name
Management node (`ndb_mgmd`)	`198.51.100.102`
Data node (`ndbd`)	`198.51.100.103`
Data node (`ndbd`)	`198.51.100.104`
SQL node (`mysqld`)	`198.51.100.102`

We assume that these hosts are on a dedicated network or subnet, and that each of them is running only the MySQL NDB Cluster binaries and applications providing required system and network services. We assume on each host that the MySQL NDB Cluster software has been installed from a release binary archive (see Installing an NDB Cluster Binary Release on Linux). We also assume that management node is using `/home/ari/bin/cluster/wild-cluster/config.ini` as the cluster's global configuration file, which is shown here:

```
[ndbd default]
NoOfReplicas= 2

[ndb_mgmd]
HostName= 198.51.100.102
DataDir= /home/ari/bin/cluster/wild-cluster/50/data
NodeId= 50


[ndbd]
HostName= 198.51.100.103
DataDir= /home/ari/bin/cluster/wild-cluster/2/data
NodeId=2

[ndbd]
HostName= 198.51.100.104
DataDir= /home/ari/bin/cluster/wild-cluster/3/data
NodeId=3

[mysqld]
HostName= 198.51.100.102
NodeId= 51

[api]
NodeId= 52
```

Notice that for the import into MySQL Cluster Manager to be successful, the following must be true for the cluster's configuration:

- `NodeID` must be explicitly assigned for every node.
- `DataDir` must be specified for each management and data node, and the data directories for different nodes cannot overlap with each other.
- A “free” API node not bounded to any host must be provisioned, through which the `mcmd` agent can communicate with the cluster.

4.5.2.1 Preparing the Standalone Cluster for Migration

The next step in the import process is to prepare the wild cluster for migration. This requires, among other things, removing cluster processes from control by any system service management facility, and making sure all management nodes are running with configuration caching disabled. More detailed information about performing these tasks is provided in the remainder of this section.

a. Before proceeding with any migration, the taking of a backup using the `ndb_mgm` client's `START BACKUP` command is strongly recommended.

b. Any cluster processes that are under the control of a system boot process management facility such as `/etc/init.d` on Linux systems or the Services Manager on Windows platforms should be removed from this facility's control. Consult your operating system's documentation for information about how to do this. Be sure not to stop any running cluster processes in the course of doing so.

c. Create a MySQL user account on each of the wild cluster's SQL nodes for MySQL Cluster Manager to execute the `import config` and `import cluster` commands in the steps to follow. The account name and password MySQL Cluster Manager uses to access MySQL nodes are specified by the `mcmd` agent options `mcmd-user` and `mcmd_password` (the default values are `mcmd` and `super`, respectively); use those credentials when creating the account on the wild cluster's SQL nodes, and grant the user all privileges on the server, including the privilege to grant privileges. For example, log in to each of the wild cluster's SQL nodes with the `mysql` client as `root` and execute the SQL statements shown here:

```
CREATE USER 'mcmd'@'localhost' IDENTIFIED BY 'super';

GRANT ALL PRIVILEGES ON *.* TO 'mcmd'@'localhost' WITH GRANT OPTION;
```

Keep in mind that this must be done on all the SQL nodes, unless distributed privileges are enabled on the wild cluster.

d. Make sure every node of the wild cluster has been started with its node ID specified with the `--ndb-nodeid` option at the command line, not just in the cluster configuration file. That is required for each process to be correctly identified by `mcmd` during the import. You can check if the requirement is fulfilled by the `ps -ef | grep` command, which shows the options the process has been started with:

```
$> ps -ef | grep ndb_mgmd
ari       8118     1  0 20:51 ?        00:00:04 /home/ari/bin/cluster/bin/ndb_mgmd --config-file=/ho
--configdir=/home/ari/bin/cluster/wild-cluster --initial --ndb-nodeid=50
```

(For clarity's sake, in the command output for the `ps -ef | grep` command in this and the upcoming sections, we are skipping the line of output for the `grep` process itself.)

If the requirement is not fulfilled, restart the process with the `--ndb-nodeid` option; the restart can also be performed in step (e) or (f) below for any nodes you are restarting in those steps.

e. Make sure that the configuration cache is disabled for each management node. Since the configuration cache is enabled by default, unless the management node has been started with the `--config-cache=false` option, you will need to stop and restart it with that option, in addition to other options that it has been started with previously.

On Linux, we can once again use `ps` to obtain the information we need to accomplish this step. In a shell on host `198.51.100.102`, on which the management node resides:

```
$> ps -ef | grep ndb_mgmd
ari       8118     1  0 20:51 ?        00:00:04 /home/ari/bin/cluster/bin/ndb_mgmd --config-file=/home/a
--configdir=/home/ari/bin/cluster/wild-cluster --initial --ndb-nodeid=50
```

The process ID is 8118. The configuration cache is turned on by default, and a configuration directory has been specified using the `--configdir` option. First, terminate the management node using `kill` as shown here, with the process ID obtained from `ps` previously:

```
$> kill -15 8118
```

Verify that the management node process was stopped—it should no longer appear in the output of another `ps` command.

Now, restart the management node as described previously, with the configuration cache disabled and with the options that it was started with previously. Also, as already stated in step (d) above, make sure that the `--ndb-nodeid` option is specified at the restart:

```
$> /home/ari/bin/cluster/bin/ndb_mgmd --config-file=/home/ari/bin/cluster/wild-cluster/config.ini --con
MySQL Cluster Management Server mysql-9.6.0
2020-11-08 21:29:43 [MgmtSrvr] INFO     -- Skipping check of config directory since config cache is dis
```

Caution

Do not use `0` or `OFF` for the value of the `--config-cache` option when restarting `ndb_mgmd` in this step. Using either of these values instead of `false` at this time causes the migration of the management node process to fail at later point in the import process.

Verify that the process is running as expected, using `ps`:

```
$> ps -ef | grep ndb_mgmd
ari      10221     1  0 19:38 ?        00:00:09 /home/ari/bin/cluster/bin/ndb_mgmd --config-file=/home/a
```

The management node is now ready for migration.

Important

While our example cluster has only a single management node, it is possible for a MySQL NDB Cluster to have more than one. In such cases, you must make sure the configuration cache is disabled for *each* management with the steps described in this step.

4.5.2.2 Verify All Cluster Process PID Files

You must verify that each process in the wild cluster has a valid PID file. For purposes of this discussion, a valid PID file has the following characteristics:

- The file name is in the format of `ndb_node_id.pid`, where *`node_id`* is the node ID used for the process.
- The file is located in the data directory used by the process.
- The first line of the file contains the process ID of the node process, and only that process ID.

a. To check the PID file for the management node process, log in to a system shell on host `198.51.100.102`, change to the management node's data directory as specified by the `Datadir` parameter in the cluster's configuration file, then check to see whether the PID file is present. On Linux, you can use the command shown here:

```
$> ls ndb_*.pid
```

```
ndb_50.pid
```

Check the content of the matching `.pid` file using a pager or text editor. We use `more` for this purpose here:

```
$> more ndb_50.pid
10221
```

The number shown should match the `ndb_mgmd` process ID. We can check this on Linux using the `ps` command:

```
$> ps -ef | grep ndb_mgmd
ari      10221     1  0 19:38 ?        00:00:09 /home/ari/bin/cluster/bin/ndb_mgmd --config-file=/ho
```

The management node PID file satisfies the requirements listed at the beginning of this section.

b. Next, we check the PID files for the data nodes, on hosts `198.51.100.103` and `198.51.100.104`. Log in to a system shell on `198.51.100.103`, then obtain the process ID of the `ndbd` process on this host, as shown here:

```
$> ps -ef | grep ndbd
ari      12838     1  0 Nov08 ?        00:10:12 ./bin/ndbd --initial --ndb-nodeid=2 --ndb-connectstr
```

As specified in the cluster's configuration file, the node's `DataDir` is `/home/ari/bin/cluster/wild-cluster/2/data`. Go to that directory to look for a file named `ndb_2.pid`:

```
$> ls ndb_*.pid
ndb_2.pid
```

Now check the content of this file, and you are going to see the process ID for angel process for the data node:

```
$> more ndb_2.pid
12838
```

There should be no need to adjust the PID files to contain the data node processes' own PIDs, as that should have been taken care of by the `--remove-angel` option used with the `import cluster` command at the last step of the import process. The data nodes are ready for import as long as they have valid PID files containing the PIDs for their angel processes.

We are ready to proceed to the `mysqld` node running on host `198.51.100.102`.

c. To check the PID file for the mysqld node: the default location for it is the data directory of the node, specified by the `datadir` option in either a configuration file or at the command line at the start of the `mysqld` process. Let's go to the data directory `/home/ari/bin/cluster/wild-cluster/51/data` on host `198.51.100.104` and look for the PID file.

```
$> ls *.pid
localhost.pid
```

Notice that the MySQL Server could have been started with the `--pid-file` option, which puts a PID file at a specified location. In the following case, the same `mysqld` node has been started with the `mysqld_safe` script, and the `ps` command reveals the value for the `--pid-file` used:

```
$>  ps -ef | grep mysqld
ari      11999  5667  0 13:15 pts/1    00:00:00 /bin/sh ./bin/mysqld_safe --defaults-file=/home/ari/
ari      12136 11999  1 13:15 pts/1    00:00:00 /home/ari/bin/cluster/bin/mysqld --defaults-file=/ho
--basedir=/home/ari/bin/cluster/ --datadir=/home/ari/bin/cluster/wild-cluster/51/data/ --plugin-dir=
--ndb-nodeid=51 --log-error=/home/ari/bin/cluster/wild-cluster/51/data//localhost.localdomain.err
--pid-file=/home/ari/bin/cluster/wild-cluster/51/data//localhost.localdomain.pid
```

As in the example, it is likely that you have a PID file that is not named in the required format for cluster import (`ndb_node_id.pid`); and if the `--pid-file` option was used, the PID file might not be at the required location (the data directory). Let us look into the PID file being referred to in the last example:

```
$> more /home/ari/bin/cluster/wild-cluster/51/data//localhost.localdomain.pid
12136
```

The PID file for the SQL node is at an acceptable location (inside the data directory) and has the correct contents (the right PID), but has the wrong name. Let us just copy the PID file into a correctly named file in the same directory, like this

```
$> cd /home/ari/bin/cluster/wild-cluster/51/data/
$> cp localhost.localdomain.pid ndb_51.pid
```

4.5.2.3 Creating and Configuring the Target Cluster

The next task is to create a “target” cluster. Once this is done, we modify the target cluster's configuration until it matches that of the wild cluster that we want to import. At a later point in the example, we also show how to test the configuration in a dry run before attempting to perform the actual configuration import.

To create and then configure the target cluster, follow these steps:

a. Install MySQL Cluster Manager and start `mcmd` on all hosts *with the same system user who started the wild cluster processes*. Once you have done this, you can start the `mcm` client (see Section 4.3, “Starting the MySQL Cluster Manager Client”) on any one of these hosts to perform the next few steps.

Important

The cluster import is going to fail due to insufficient rights for the `mcmd` agents to perform their tasks if the `mcmd` agents are not started by the same system user who started the wild cluster processes.

b. Create a MySQL Cluster Manager site encompassing all four of the wild cluster's hosts, using the `create site` command, as shown here:

```
mcm> create site --hosts=198.51.100.102,198.51.100.103,198.51.100.104 newsite;
+---------------------------+
| Command result            |
+---------------------------+
| Site created successfully |
+---------------------------+
1 row in set (0.15 sec)
```

We have named this site `newsite`. You should be able to see it listed in the output of the `list sites` command, similar to what is shown here:

```
mcm> list sites;
+---------+------+-------+---------------------------------------------+
| Site    | Port | Local | Hosts                                       |
+---------+------+-------+---------------------------------------------+
| newsite | 1862 | Local | 198.51.100.102,198.51.100.103,198.51.100.104 |
+---------+------+-------+---------------------------------------------+
1 row in set (0.01 sec)
```

c. Add a MySQL Cluster Manager package referencing the MySQL NDB Cluster binaries using the `add package` command; use the command's `--basedir` option to point to the correct location of the MySQL NDB Cluster executables. The command shown here creates such a package, named `newpackage`:

```
mcm> add package --basedir=/home/ari/bin/cluster newpackage;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (0.70 sec)
```

You do not need to include the `bin` directory containing the MySQL NDB Cluster executables in the `--basedir` path. If the executables are in `/home/ari/bin/cluster/bin`, it is sufficient to specify `/home/ari/bin/cluster`; MySQL Cluster Manager automatically checks for the binaries in a `bin` directory within the directory specified by `--basedir`.

d. Create the target cluster including at least some of the same processes and hosts used by the standalone cluster. *Do not include any processes or hosts that are not part of this cluster.* In order to prevent potentially disruptive process or cluster operations from interfering by accident with the import process, it is strongly recommended that you create the cluster for import using the `--import` option for the `create cluster` command.

   You must also take care to preserve the correct node ID (as listed in the `config.ini` file shown previously) for each node.

   The following command creates the cluster `newcluster` for import and includes the management and data nodes, but not the SQL or “free” API node (which we add in the next step):

```
mcm> create cluster --import --package=newpackage \
      --processhosts=ndb_mgmd:50@198.51.100.102,ndbd:2@198.51.100.103,ndbd:3@198.51.100.104 \
       newcluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster created successfully |
+-----------------------------+
1 row in set (0.96 sec)
```

   You can verify that the cluster was created correctly by checking the output of `show status` with the `--process` (`-r`) option, like this:

```
mcm> show status -r newcluster;
+--------+----------+----------------+--------+-----------+------------+
| NodeId | Process  | Host           | Status | Nodegroup | Package    |
+--------+----------+----------------+--------+-----------+------------+
| 50     | ndb_mgmd | 198.51.100.102 | import |           | newpackage |
| 2      | ndbd     | 198.51.100.103 | import | n/a       | newpackage |
| 3      | ndbd     | 198.51.100.104 | import | n/a       | newpackage |
+--------+----------+----------------+--------+-----------+------------+
3 rows in set (0.05 sec)
```

e. If necessary, add any remaining processes and hosts from the wild cluster not included in the previous step using one or more `add process` commands. We have not yet accounted for two of the nodes from the wild cluster: the SQL node with node ID 51, on host `198.51.100.102`, and the API node with node ID 52, which is not bound to any specific host. You can use the following command to add both of these processes to `newcluster`:

```
mcm> add process --processhosts=mysqld:51@198.51.100.102,ndbapi:52@* newcluster;
+---------------------------+
| Command result            |
+---------------------------+
| Process added successfully |
+---------------------------+
1 row in set (0.41 sec)
```

   Once again checking the output from `show status -r`, we see that the `mysqld` and `ndbapi` processes were added as expected:

```
mcm> show status -r newcluster;
+--------+----------+----------------+--------+-----------+------------+
| NodeId | Process  | Host           | Status | Nodegroup | Package    |
+--------+----------+----------------+--------+-----------+------------+
| 50     | ndb_mgmd | 198.51.100.102 | import |           | newpackage |
| 2      | ndbd     | 198.51.100.103 | import | n/a       | newpackage |
| 3      | ndbd     | 198.51.100.104 | import | n/a       | newpackage |
| 51     | mysqld   | 198.51.100.102 | import |           | newpackage |
| 52     | ndbapi   | *              | import |           |            |
```

```
+--------+---------+---------------+--------+-----------+------------+
5 rows in set (0.06 sec)
```

You can also see that since `newcluster` was created using the `create cluster` command's `--import` option, the status of all processes in this cluster—including those we just added—is `import`. This means we cannot yet start `newcluster` or any of its processes. The `import` status and its effects on `newcluster` and its cluster processes persist until we have completed importing another cluster into `newcluster`.

The target `newcluster` cluster now has the same processes, with the same node IDs, and on the same hosts as the original standalone cluster. We are ready to proceed to the next step.

f. Duplicate the wild cluster's configuration attributes in the target cluster using the `import config` command. Test out first the effects of the command by running it with the `--dryrun` option (the step only works if you have created the mcmd user on the cluster's mysqld nodes):

Important

Before executing this command it is necessary to set any non-default ports for `ndb_mgmd` and `mysqld` processes using the `set` command in the `mcm` client.

```
mcm> import config --dryrun newcluster;
+-------------------------------------------------------------------------------------------------------------
| Command result
+-------------------------------------------------------------------------------------------------------------
| Import checks passed. Please check /home/ari/bin/mcm_data/clusters/newcluster/tmp/import_config.49d54
+-------------------------------------------------------------------------------------------------------------
1 row in set (6.87 sec)
```

As indicated by the output from `import config --dryrun`, you can see the configuration attributes and values that would be copied to `newcluster` by the command without the `--dryrun` option in the file `/path-to-mcm-data-repository/clusters/clustername/tmp/import_config.message_id.mcm`. If you open this file in a text editor, you will see a series of `set` commands that would accomplish this task, similar to what is shown here:

```
# The following will be applied to the current cluster config:
set NoOfReplicas:ndbd=2 newcluster;
set DataDir:ndb_mgmd:50=/home/ari/bin/cluster/wild-cluster/50/data newcluster;
set DataDir:ndbd:2=/home/ari/bin/cluster/wild-cluster/2/data newcluster;
set DataDir:ndbd:3=/home/ari/bin/cluster/wild-cluster/3/data newcluster;
set basedir:mysqld:51=/home/ari/bin/cluster/ newcluster;
set datadir:mysqld:51=/home/ari/bin/cluster/wild-cluster/51/data/ newcluster;
set sql_mode:mysqld:51="NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES" newcluster;
set ndb_connectstring:mysqld:51=198.51.100.102 newcluster;
```

Options used at the command line instead of in a configuration file to start a node of the standalone cluster are not imported into the target cluster by the `import config` command; moreover, they will cause one of the following to happen when the `import config --dryrun` is run:

i. For some options, MySQL Cluster Manager will issue a warning that “Option *<param>* may be removed on next restart of process *<type><nodeid>*,” meaning that those options will not be imported into the target cluster, and thus will not be applied when those nodes are restarted after the import. Here are the lists of such options for each node type:

- For `ndb_mgmd` nodes: `--configdir`, `--initial`, `--log-name`, `--reload`, `--verbose`
- For `ndbd` and `ndbmtd` nodes: `--connect-retries`, `--connect-delay`, `--daemon=false`, `--nodaemon`, `--verbose`, `--core-file`
- For `mysqld` nodes: `--ndbcluster`, the `--ndbinfo-*` options, `--verbose`, `--datadir`, `--defaults-group-suffix`, `--core-file`

ii. For some other options, while their values will also not be imported into the target cluster, no warnings will be issued for them. Here are lists of such options for each node type:

- For `ndb_mgmd` nodes: `--config-cache`, `--daemon`, `--ndb-nodeid`, `--nodaemon=false`, `--config-file`, `--skip-config-cache`
- For `ndbd` and `ndbmtd` nodes: `--daemon`, `--foreground`, `--initial`, `--ndb-connectstring`, `--connect-string`, `--ndb-mgmd-host`, `--ndb-nodeid`, `--nodaemon=false`
- For `mysqld` nodes: `--ndb-connectstring`, `--ndb-mgmd-host`, `--ndb-nodeid`, `--defaults-file`, `--no-defaults`, `--basedir`

iii. For options that belong to neither of the two groups described above, having started the standalone cluster's nodes with them at the command line will cause the `import config --dryrun` command to fail with an error, complaining that the options are unsupported.

When you run into the first or third case described above, you have to do one of the following:

- If the options are required for the target cluster and they can be set using the `set` command (see Command-line-only attributes), set them for the target cluster using the `set` command, and then retry the `import config --dryrun` command.
- If the options are not needed for the target cluster, or it cannot be set using the `set` command, restart the wild cluster's node without those options, and then retry the `import config --dryrun` command.

After the successful dry run, you are now ready to import the wild cluster's configuration into `newcluster`, with the command shown here:

```
mcm> import config newcluster;
+-------------------------------------------------------------------------------------------------------------
| Command result
+-------------------------------------------------------------------------------------------------------------
| Configuration imported successfully. Please manually verify plugin options, abstraction level and
+-------------------------------------------------------------------------------------------------------------
```

As an alternative, instead of importing all the settings using the `import config` command, you can make changes to the `/path-to-mcm-data-repository/clusters/clustername/tmp/import_config.message_id.mcm` file generated by the dry run as you wish, and then import the settings by executing the file with the `mcm` agent:

```
mcm> source /path-to-mcm-data-repository/clusters/clustername/tmp/import_config.message_id.mcm
```

You should check the resulting configuration of `newcluster` carefully against the configuration of the wild cluster. If you find any inconsistencies, you must correct these in `newcluster` using the appropriate `set` commands.

4.5.2.4 Testing and Migrating the Standalone Cluster

Testing and performing the migration of a standalone MySQL NDB Cluster into MySQL Cluster Manager consists of the following steps:

1. Perform a test run of the proposed import using `import cluster` with the `--dryrun` option. When this option is used, MySQL Cluster Manager checks for mismatched configuration attributes, missing or invalid processes or hosts, missing or invalid PID files, and other errors, and warns of any it finds, without actually performing any migration of processes or data (the step only works if you have created the mcmd user on the cluster's mysqld nodes):

```
mcm> import cluster --dryrun newcluster;
```

2. If errors occur, correct them, and repeat the dry run shown in the previous step until it returns no more errors. The following list contains some common errors you may encounter, and their likely causes:

   - MySQL Cluster Manager requires a specific MySQL user and privileges to manage SQL nodes. If the `mcmd` MySQL user account is not set up properly, you may see `No access for user...`, `Incorrect grants for user...`, or possibly other errors. Follow the instructions given in this step in Section 4.5.2.1, “Preparing the Standalone Cluster for Migration” to remedy the issue.

   - As described previously, each cluster process (other than a process whose type is `ndbapi`) being brought under MySQL Cluster Manager control must have a valid PID file. Missing, misnamed, or invalid PID files can produce errors such as `PID file does not exist for process...`, `PID ... is not running ...`, and `PID ... is type ...`. See Section 4.5.2.2, “Verify All Cluster Process PID Files”.

   - Process version mismatches can also produce seemingly random errors whose cause can sometime prove difficult to track down. Ensure that all nodes are supplied with the correct release of the MySQL NDB Cluster software, and that it is the same release and version of the software.

   - Each data node angel process in the standalone cluster must be killed prior to import. A running angel process can cause errors such as `Angel process pid exists ...` or `Process pid is an angel process for ...`. If you see such errors, proceed to the next step if these are the only errors you get. The angel processes and the data node PIDs will be taken care of by the `--remove-angel` option used with the `import cluster` command at the last step of the import process.

   - The number of processes, their types, and the hosts where they reside in the standalone cluster must be reflected accurately when creating the target site, package, and cluster for import. Otherwise, you may get errors such as `Process id reported # processes ...`, `Process id ... does not match configured process ...`, `Process id not configured ...`, and `Process id does not match configured process ...`. See Section 4.5.2.3, “Creating and Configuring the Target Cluster”.

   - Other factors that can cause specific errors include processes in the wrong state, processes that were started with unsupported command-line options (see Section 4.5.2.3, “Creating and Configuring the Target Cluster” for details) or without required options, and processes having the wrong process ID, or using the wrong node ID.

3. When `import cluster --dryrun` no longer warns of any errors, you can perform the import with the `import cluster` command, this time omitting the `--dryrun` option. Use the `--remove-angel` option for the `import cluster` command, which kills the angel processes for the data nodes and adjusts the data nodes' PID files to contain the data node processes' own PIDs before importing the cluster:

```
mcm> import cluster --remove-angel newcluster;
+------------------------------+
| Command result               |
+------------------------------+
| Cluster imported successfully |
+------------------------------+
1 row in set (5.58 sec)
```

   You can check that the wild cluster has now been imported, and is now under management of MySQL Cluster Manager:

```
mcm> show status -r newcluster;
+--------+----------+----------------+---------+-----------+------------+
| NodeId | Process  | Host           | Status  | Nodegroup | Package    |
+--------+----------+----------------+---------+-----------+------------+
| 50     | ndb_mgmd | 198.51.100.102 | running |           | newpackage |
| 2      | ndbd     | 198.51.100.103 | running | 0         | newpackage |
| 3      | ndbd     | 198.51.100.104 | running | 0         | newpackage |
| 51     | mysqld   | 198.51.100.102 | running |           | newpackage |
```

```
| 52     | ndbapi   | *              | added   |           |            |
+--------+----------+----------------+---------+-----------+------------+
5 rows in set (0.01 sec)
```

4.6 MySQL NDB Cluster Backup and Restore Using MySQL Cluster Manager

This section describes usage of the NDB native backup and restore functionality implemented in MySQL Cluster Manager, to perform a number of common tasks.

4.6.1 Requirements for Backup and Restore

This section provides information about basic requirements for performing backup and restore operations using MySQL Cluster Manager.

Requirements for MySQL NDB Cluster backup. Basic requirements for performing MySQL backups using MySQL Cluster Manager are minimal. At least one data node in each node group must be running, and there must be sufficient disk space on the node file systems. Partial backups are not supported.

Requirements for MySQL NDB Cluster restore. In general, the following requirements apply when you try to restore a MySQL NDB Cluster using MySQL Cluster Manager:

- A complete restore requires that all data nodes are up and running, and that all files belonging to a given backup are available.
- A partial restore is possible, but must be specified as such. This can be accomplished using the `restore cluster` client command with its `--skip-nodeid` option. See Section 4.6.2.3, “Partial restore—missing images” for details.
- In the event that data nodes have been added to the cluster since the backup was taken, only those data nodes for which backup files exist are restored. In such cases data is not automatically distributed to the new nodes, and, following the restore, you must redistribute the data manually by issuing an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement in the `mysql` client for each `NDB` table in the cluster. See Adding NDB Cluster Data Nodes Online: Basic procedure, for details.
- To restore a backup created by MySQL Cluster Manager to a cluster with fewer data nodes, you need to restore first the logical backup of the metadata of the NDB tables using the `mysqldump` utility and then restore the table data using the `ndb_restore` program. See Section 4.6.2.5, “Restoring a Backup to a Cluster with Fewer Data Nodes” for details.

4.6.2 Basic MySQL NDB Cluster Backup and Restore Using MySQL Cluster Manager

This section describes backing up and restoring a MySQL NDB Cluster, with examples of complete and partial restore operations. Note that the `backup cluster` and `restore cluster` commands work with `NDB` tables only; tables using other MySQL storage engines (such as `InnoDB` or `MyISAM`) are ignored.

For purposes of example, we use a MySQL NDB Cluster named `mycluster` whose processes and status can be seen here:

```
mcm> show status -r mycluster;
+--------+----------+----------+---------+-----------+-----------+
| NodeId | Process  | Host     | Status  | Nodegroup | Package   |
+--------+----------+----------+---------+-----------+-----------+
| 49     | ndb_mgmd | tonfisk  | running |           | mypackage |
| 1      | ndbd     | tonfisk  | running | 0         | mypackage |
| 2      | ndbd     | tonfisk  | running | 0         | mypackage |
| 50     | mysqld   | tonfisk  | running |           | mypackage |
```

```
| 51     | mysqld   | tonfisk  | running |           | mypackage |
| 52     | ndbapi   | *tonfisk | added   |           |           |
| 53     | ndbapi   | *tonfisk | added   |           |           |
+--------+----------+----------+---------+-----------+-----------+
7 rows in set (0.08 sec)
```

You can see whether there are any existing backups of `mycluster` using the `list backups` command, as shown here:

```
mcm> list backups mycluster;
+----------+--------+---------+---------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment |
+----------+--------+---------+---------------------+-------+---------+
| 1        | 1      | tonfisk | 2020-12-04 12:03:52 | 1     |         |
| 1        | 2      | tonfisk | 2020-12-04 12:03:52 | 1     |         |
| 2        | 2      | tonfisk | 2020-12-04 12:04:15 | 1     |         |
| 3        | 1      | tonfisk | 2020-12-04 12:17:41 | 1     |         |
| 3        | 2      | tonfisk | 2020-12-04 12:17:41 | 1     |         |
+----------+--------+---------+---------------------+-------+---------+
6 rows in set (0.12 sec)
```

4.6.2.1 Simple backup

To create a backup, use the `backup cluster` command with the name of the cluster as an argument, similar to what is shown here:

```
mcm> backup cluster mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Backup completed successfully |
+------------------------------+
1 row in set (3.31 sec)
```

`backup cluster` requires only the name of the cluster to be backed up as an argument; for information about additional options supported by this command, see Section 5.8.2, “The `backup cluster` Command”. To verify that a new backup of `mycluster` was created with a unique ID, check the output of `list backups`, as shown here (where the rows corresponding to the new backup files are indicated with emphasized text):

```
mcm> list backups mycluster;
+----------+--------+---------+---------------------+--------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts  | Comment |
+----------+--------+---------+---------------------+--------+---------+
| 1        | 1      | tonfisk | 2020-12-04 12:03:52 | 1      |         |
| 1        | 2      | tonfisk | 2020-12-04 12:03:52 | 1      |         |
| 2        | 1      | tonfisk | 2020-12-04 12:04:15 | 1      |         |
| 2        | 2      | tonfisk | 2020-12-04 12:04:15 | 1      |         |
| 3        | 1      | tonfisk | 2020-12-04 12:17:41 | 1      |         |
| 3        | 2      | tonfisk | 2020-12-04 12:17:41 | 1      |         |
| 4        | 1      | tonfisk | 2020-12-12 14:24:35 | 1      |         |
| 4        | 2      | tonfisk | 2020-12-12 14:24:35 | 1      |         |
+----------+--------+---------+---------------------+--------+---------+
8 rows in set (0.04 sec)
```

If you attempt to create a backup of a MySQL NDB Cluster in which each node group does not have at least one data node running, `backup cluster` fails with the error `Backup cannot be performed as processes are stopped in cluster` *`cluster_name`*.

4.6.2.2 Simple complete restore

To perform a complete restore of a MySQL NDB Cluster from a backup with a given ID, follow the steps listed here:

1. Identify the backup to be used.

   In this example, we use the backup having the ID 4, that was created for `mycluster` previously in this section.

2. Wipe the MySQL NDB Cluster data.

   The simplest way to do this is to stop and then perform an initial start of the cluster as shown here, using `mycluster`:

```
mcm> stop cluster mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster stopped successfully |
+-----------------------------+
1 row in set (15.24 sec)

mcm> start cluster --initial mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster started successfully |
+-----------------------------+
1 row in set (34.47 sec)
```

3. Restore the backup.

   This is done using the `restore cluster` command, which requires the backup ID and the name of the cluster as arguments. Thus, you can restore backup 4 to `mycluster` as shown here:

```
mcm> restore cluster --backupid=4 mycluster;
+--------------------------------+
| Command result                 |
+--------------------------------+
| Restore completed successfully |
+--------------------------------+
1 row in set (16.78 sec)
```

4.6.2.3 Partial restore—missing images

It is possible using MySQL Cluster Manager to perform a partial restore of a MySQL NDB Cluster—that is, to restore from a backup in which backup images from one or more data nodes are not available. This is required if we wish to restore `mycluster` to backup number 6, since an image for this backup is available only for node 1, as can be seen in the output of `list backups` in the `mcm` client :

```
mcm> list backups mycluster;
+----------+--------+---------+---------------------+--------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts  | Comment |
+----------+--------+---------+---------------------+--------+---------+
| 1        | 1      | tonfisk | 2020-12-04 12:03:52 | 1      |         |
| 1        | 2      | tonfisk | 2020-12-04 12:03:52 | 1      |         |
| 2        | 1      | tonfisk | 2020-12-04 12:04:15 | 1      |         |
| 2        | 2      | tonfisk | 2020-12-04 12:04:15 | 1      |         |
| 3        | 1      | tonfisk | 2020-12-04 12:17:41 | 1      |         |
| 3        | 2      | tonfisk | 2020-12-04 12:17:41 | 1      |         |
| 4        | 1      | tonfisk | 2020-12-12 14:24:35 | 1      |         |
| 4        | 2      | tonfisk | 2020-12-12 14:24:35 | 1      |         |
| 5        | 1      | tonfisk | 2020-12-12 14:31:31 | 1      |         |
| 5        | 2      | tonfisk | 2020-12-12 14:31:31 | 1      |         |
| 6        | 1      | tonfisk | 2020-12-12 14:32:09 | 1      |         |
+----------+--------+---------+---------------------+--------+---------+
11 rows in set (0.08 sec)
```

To perform a restore of only those nodes for which we have images (in this case, node 1 only), we can use the `--skip-nodeid` option when executing a `restore cluster` command. This option causes one or more nodes to be skipped when performing the restore. Assuming that `mycluster` has been cleared of data (as described earlier in this section), we can perform a restore that skips node 2 as shown here:

```
mcm> restore cluster --backupid=6 --skip-nodeid=2 mycluster;
+--------------------------------+
| Command result                 |
```

```
+-------------------------------+
| Restore completed successfully |
+-------------------------------+
1 row in set (17.06 sec)
```

Because we excluded node 2 from the restore process, no data has been distributed to it. To cause MySQL NDB Cluster data to be distributed to any such excluded or skipped nodes following a partial restore, it is necessary to redistribute the data manually by executing an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement in the `mysql` client for each `NDB` table in the cluster. To obtain a list of `NDB` tables from the `mysql` client, you can use multiple `SHOW TABLES` statements or a query such as this one:

```
SELECT CONCAT('' TABLE_SCHEMA, '.', TABLE_NAME)
    FROM INFORMATION_SCHEMA.TABLES
    WHERE ENGINE='ndbcluster';
```

You can generate the necessary SQL statements using a more elaborate version of the query just shown, such the one employed here:

```
mysql> SELECT
    ->     CONCAT('ALTER ONLINE TABLE `', TABLE_SCHEMA,
    ->            '`.`', TABLE_NAME, '` REORGANIZE PARTITION;')
    ->     AS Statement
    -> FROM INFORMATION_SCHEMA.TABLES
    -> WHERE ENGINE='ndbcluster';
+-----------------------------------------------------------------------------+
| Statement                                                                   |
+-----------------------------------------------------------------------------+
| ALTER ONLINE TABLE `mysql`.`ndb_apply_status` REORGANIZE PARTITION;          |
| ALTER ONLINE TABLE `mysql`.`ndb_index_stat_head` REORGANIZE PARTITION;      |
| ALTER ONLINE TABLE `mysql`.`ndb_index_stat_sample` REORGANIZE PARTITION;    |
| ALTER ONLINE TABLE `db1`.`n1` REORGANIZE PARTITION;                         |
| ALTER ONLINE TABLE `db1`.`n2` REORGANIZE PARTITION;                         |
| ALTER ONLINE TABLE `db1`.`n3` REORGANIZE PARTITION;                         |
| ALTER ONLINE TABLE `test`.`n1` REORGANIZE PARTITION;                        |
| ALTER ONLINE TABLE `test`.`n2` REORGANIZE PARTITION;                        |
| ALTER ONLINE TABLE `test`.`n3` REORGANIZE PARTITION;                        |
| ALTER ONLINE TABLE `test`.`n4` REORGANIZE PARTITION;                        |
+-----------------------------------------------------------------------------+
10 rows in set (0.09 sec)
```

4.6.2.4 Partial restore—data nodes added

A partial restore can also be performed when new data nodes have been added to a MySQL NDB Cluster following a backup. In this case, you can exclude the new nodes using `--skip-nodeid` when executing the `restore cluster` command. Consider the MySQL NDB Cluster named `mycluster` as shown in the output of the following `show status` command:

```
mcm> show status -r mycluster;
+--------+----------+----------+---------+-----------+-----------+
| NodeId | Process  | Host     | Status  | Nodegroup | Package   |
+--------+----------+----------+---------+-----------+-----------+
| 49     | ndb_mgmd | tonfisk  | stopped |           | mypackage |
| 1      | ndbd     | tonfisk  | stopped | 0         | mypackage |
| 2      | ndbd     | tonfisk  | stopped | 0         | mypackage |
| 50     | mysqld   | tonfisk  | stopped |           | mypackage |
| 51     | mysqld   | tonfisk  | stopped |           | mypackage |
| 52     | ndbapi   | *tonfisk | added   |           |           |
| 53     | ndbapi   | *tonfisk | added   |           |           |
+--------+----------+----------+---------+-----------+-----------+
7 rows in set (0.03 sec)
```

The output of `list backups` shows us the available backup images for this cluster:

```
mcm> list backups mycluster;
+----------+--------+---------+---------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment |
+----------+--------+---------+---------------------+-------+---------+
| 1        | 1      | tonfisk | 2020-12-04 12:03:52 | 1     |         |
```

```
| 1        | 2      | tonfisk | 2020-12-04 12:03:52 | 1     |         |
| 2        | 1      | tonfisk | 2020-12-04 12:04:15 | 1     |         |
| 2        | 2      | tonfisk | 2020-12-04 12:04:15 | 1     |         |
| 3        | 1      | tonfisk | 2020-12-04 12:17:41 | 1     |         |
| 3        | 2      | tonfisk | 2020-12-04 12:17:41 | 1     |         |
| 4        | 1      | tonfisk | 2020-12-12 14:24:35 | 1     |         |
| 4        | 2      | tonfisk | 2020-12-12 14:24:35 | 1     |         |
+----------+--------+---------+---------------------+-------+---------+
8 rows in set (0.06 sec)
```

Now suppose that, at a later point in time, 2 data nodes have been added to `mycluster` using an `add process` command. The `show status` output for `mycluster` now looks like this:

```
mcm> show status -r mycluster;
+--------+----------+----------+---------+-----------+-----------+
| NodeId | Process  | Host     | Status  | Nodegroup | Package   |
+--------+----------+----------+---------+-----------+-----------+
| 49     | ndb_mgmd | tonfisk  | running |           | mypackage |
| 1      | ndbd     | tonfisk  | running | 0         | mypackage |
| 2      | ndbd     | tonfisk  | running | 0         | mypackage |
| 50     | mysqld   | tonfisk  | running |           | mypackage |
| 51     | mysqld   | tonfisk  | running |           | mypackage |
| 52     | ndbapi   | *tonfisk | added   |           |           |
| 53     | ndbapi   | *tonfisk | added   |           |           |
| 3      | ndbd     | tonfisk  | running | 1         | mypackage |
| 4      | ndbd     | tonfisk  | running | 1         | mypackage |
+--------+----------+----------+---------+-----------+-----------+
9 rows in set (0.01 sec)
```

Since nodes 3 and 4 were not included in the backup, we need to exclude them when performing the restore. You can cause `restore cluster` to skip multiple data nodes by specifying a comma-separated list of node IDs with the `--skip-nodeid` option. Assume that we have just cleared `mycluster` of MySQL NDB Cluster data using the `mcm` client commands `stop cluster` and `start cluster --initial` as described previously in this section; then we can restore `mycluster` (now having 4 data nodes numbered 1, 2, 3, and 4) from backup number 4 (made when `mycluster` had only 2 data nodes numbered 1 and 2) as shown here:

```
mcm> restore cluster --backupid=4 --skip-nodeid=3,4 mycluster;
+--------------------------------+
| Command result                 |
+--------------------------------+
| Restore completed successfully |
+--------------------------------+
1 row in set (17.61 sec)
```

No data is distributed to the skipped (new) nodes; you must force nodes 3 and 4 to be included in a redistribution of the data using `ALTER ONLINE TABLE ... REORGANIZE PARTITION` as described previously in this section.

An alternative to generating and running the `ALTER ONLINE TABLE ... REORGANIZE PARTITION` steps is to make use of the logical backup of the NDB tables' metadata, which is part of the cluster backup created by MySQL Cluster Manager. To do this, before you run the `restore cluster` step outlined above:

1. Locate the logical backup for the metadata; see Locations of backup files in Section 4.6.2.5, “Restoring a Backup to a Cluster with Fewer Data Nodes” for instructions.

2. Restore the logical backup; see Restore the Logical Backup of NDB Table Metadata [53] in Section 4.6.2.5, “Restoring a Backup to a Cluster with Fewer Data Nodes” for instructions.

You can then run the `restore cluster` step, and the data is going to be redistributed across all the data nodes, without the need for further manual intervention.

4.6.2.5 Restoring a Backup to a Cluster with Fewer Data Nodes

Sometimes, you want to transfer data from your cluster to another one that has fewer data nodes—for example, when you want to scale down your cluster or prepare a smaller replica cluster for a replication

setup. While the methods described in Section 4.6.2, “Basic MySQL NDB Cluster Backup and Restore Using MySQL Cluster Manager” will not work in that case, you can perform the transfer by just using the `backup cluster` command and the `ndb_restore` program.

The process starts with creating a backup for the original cluster using the `backup cluster` command. Next, create a new cluster with fewer data nodes using the `create cluster` command. Before the NDB table data can be transferred, the metadata for the NDB tables must first be restored to the new cluster. The `backup cluster` command also creates a logical backup for the metadata of the NDB tables (see Logical Backup for NDB Table Metadata, for details). Use the `--all` option with the `list backups` command to list all backups, including the logical backups for the NDB tables' metadata, which are marked by the comment “Schema”:

```
mcm> list backups --all mycluster;
+----------+--------+---------+---------------------+------+--------+
| BackupId | NodeId | Host    | Timestamp           | Part |Comment |
+----------+--------+---------+---------------------+------+--------+
| 1        | 1      | tonfisk | 2020-09-21 21:13:09Z | 1    |        |
| 1        | 2      | tonfisk | 2020-09-21 21:13:09Z | 1    |        |
| 1        | 3      | tonfisk | 2020-09-21 21:13:09Z | 1    |        |
| 1        | 4      | tonfisk | 2020-09-21 21:13:09Z | 1    |        |
| 1        | 50     | tonfisk | 2020-09-21 21:13:12Z |      | Schema |
| 2        | 1      | tonfisk | 2020-09-21 21:17:50Z | 1    |        |
| 2        | 2      | tonfisk | 2020-09-21 21:17:50Z | 1    |        |
| 2        | 3      | tonfisk | 2020-09-21 21:17:50Z | 1    |        |
| 2        | 4      | tonfisk | 2020-09-21 21:17:50Z | 1    |        |
| 2        | 50     | tonfisk | 2020-09-21 21:17:52Z |      | Schema |
+----------+--------+---------+---------------------+------+--------+
10 rows in set (0.01 sec)
```

Next, we have to find out the locations of the logical backup file and the backup files for each data node of the original cluster.

Locations of backup files. The backup files for each node are to be found under the folder specified by the cluster parameter `BackupDataDir` for data nodes and the parameter `backupdatadir` for `mysqld` nodes. Because the `get` command is not case sensitive, you can use this single command to check the values of both parameters:

```
mcm> get BackupDataDir mycluster;
+--------------+----------------+----------+---------+----------+---------+---------+----------+
| Name         | Value          | Process1 | NodeId1 | Process2 | NodeId2 | Level   | Comment  |
+--------------+----------------+----------+---------+----------+---------+---------+----------+
| BackupDataDir | /opt/mcmbackup | ndbmtd   | 1       |          |         | Process |          |
| BackupDataDir | /opt/mcmbackup | ndbmtd   | 2       |          |         | Process |          |
| BackupDataDir | /opt/mcmbackup | ndbmtd   | 3       |          |         | Process |          |
| BackupDataDir | /opt/mcmbackup | ndbmtd   | 4       |          |         | Process |          |
| backupdatadir | /opt/mcmbackup | mysqld   | 50      |          |         | Process | MCM only |
+--------------+----------------+----------+---------+----------+---------+---------+----------+
5 rows in set (0.18 sec)
```

The backup files for each backup of a specific BackupID are found under *`BackupDataDir`*`/BACKUP/BACKUP-`*`ID/`* for each data node, and under *`backupdatadir`*`/BACKUP/BACKUP-`*`ID/`* for each `mysqld` node. The `comment` “`MCM only`” in the row returned for the parameter `backupdatadir` indicates that `backupdatadir` is used by MySQL Cluster Manager only, and the directory it specifies contains only backups for the NDB tables' metadata. Notice that If `BackupDataDir` is not specified, the `get` command will return no value for it, and it takes up the value of `DataDir`, so that the image is stored in the directory *`Datadir`*`/BACKUP/BACKUP-`*`backup_id`*. If `backupdatadir` has not been specified, the `get` command will again return no value for it, and the logical backup files for the `mysqld` node are to be found at the default locations of *`/path-to-mcm-data-repository/`*`clusters/`*`clustername`*`/`*`nodeid`*`/BACKUP/BACKUP-`*`Id`*.

The process of restoring the backed-up data from the original cluster to the new one consists of the following steps:

1. Stop the original cluster:

```
mcm> stop cluster mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster stopped successfully |
+-----------------------------+
1 row in set (19.54 sec)

mcm> show status mycluster;
+-----------+---------+---------+
| Cluster   | Status  | Comment |
+-----------+---------+---------+
| mycluster | stopped |         |
+-----------+---------+---------+
1 row in set (0.05 sec)
```

2. Start your new cluster. Make sure the new cluster is operational and it has at least one free `ndbapi` slot for the `ndb_restore` utility to connect to the cluster:

```
mcm> start cluster newcluster2nodes;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster started successfully |
+-----------------------------+
1 row in set (33.68 sec)

mcm> show status -r newcluster2nodes;
+--------+----------+---------+---------+-----------+-----------+
| NodeId | Process  | Host    | Status  | Nodegroup | Package   |
+--------+----------+---------+---------+-----------+-----------+
| 49     | ndb_mgmd | tonfisk | running |           | mypackage |
| 1      | ndbmtd   | tonfisk | running | 0         | mypackage |
| 2      | ndbmtd   | tonfisk | running | 0         | mypackage |
| 50     | mysqld   | tonfisk | running |           | mypackage |
| 51     | ndbapi   | *       | added   |           |           |
+--------+----------+---------+---------+-----------+-----------+
5 rows in set (0.09 sec)
```

3. Restore the logical backup of the metadata of the NDB tables onto the new cluster. See Reloading SQL-Format Backups for different ways to restore a logical backup. One way to do it is to open a `mysql` client, connect it to a `mysqld` node of the cluster, and then source the logical backup file with the `mysql` client:

```
mysql> source path-to-logical-backup-file/BACKUP-BackupID.mysqld_nodeid.schema.sql
```

   See Locations of backup files above on how to find the path of the logical backup file. For our sample clusters, this is how the command looks like for restoring the NDB table metadata from the backup with the BackupID 2:

```
mysql> source /opt/mcmbackup/BACKUP/BACKUP-2/BACKUP-2.50.schema.sql
```

4. Restore one by one the backup for each data node of the original cluster to the new cluster, using the `ndb_restore` program:

```
$> ndb_restore -b BackupID -n nodeID -r --backup_path=backup-folder-for-data_node
```

   See Locations of backup files above on how to find the path of the data node backup files. For our sample clusters, to restore the data from the backup with the BackupID 2 for data node 1 to 4 of `mycluster`, execute the following commands:

```
$> ndb_restore --backupid=2 --nodeid=1 --restore_data --backup_path=/opt/mcmbackup/BACKUP/BACKUP-2/

$> ndb_restore --backupid=2 --nodeid=2 --restore_data --backup_path=/opt/mcmbackup/BACKUP/BACKUP-2/

$> ndb_restore --backupid=2 --nodeid=3 --restore_data --backup_path=/opt/mcmbackup/BACKUP/BACKUP-2/

$> ndb_restore --backupid=2 --nodeid=4 --restore_data --backup_path=/opt/mcmbackup/BACKUP/BACKUP-2/
```

The option `--disable-indexes` is used so indexes are ignored during the restores. This is because if we also try to restore the indexes node by node, they might not be restored in the right order for the foreign keys and unique key constraints to work properly. Therefore, the `--disable-indexes` option is used in the above commands, after the execution of which we rebuild the indexes with the following `ndb_restore` command and the `--rebuild-indexes` option (you only need to run this on one of the data nodes):

```
$> ndb_restore --backupid=2 --nodeid=1 --rebuild-indexes --backup_path=/opt/mcmbackup/BACKUP/BACKUP-2/
```

The data and indexes have now been fully restored to the new cluster.

4.7 Backing Up and Restoring MySQL Cluster Manager Agents

This section explains how to back up configuration data for `mcmd` agents and how to restore the backed-up agent data. Used together with the `backup cluster` command, the `backup agents` command allows you to backup and restore a complete cluster-plus-manager setup.

If no host names are given with the `backup agents` command, backups are created for all agents of the site:

```
mcm> backup agents mysite;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Agent backup created successfully |
+----------------------------------+
1 row in set (0.07 sec)
```

To backup one or more specific agents, specify them with the `--hosts` option:

```
mcm> backup agents --hosts=tonfisk mysite;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Agent backup created successfully |
+----------------------------------+
1 row in set (0.07 sec)
```

If no site name is given, only the agent that the `mcm` client is connected to is backed up.

The backup for each agent includes the following contents from the agent repository (`mcm_data` folder):

- The `rep` subfolder
- The metadata files `high_water_mark` and `repchksum`

The repository is locked while the backup are in progress, to avoid creating an inconsistent backup. The backup for each agent is created in a subfolder named `rep_backup/`*`timestamp`* under the agent's `mcm_data` folder, with *`timestamp`* reflecting the time the backup began. If you want the backup to be at another place, create a soft link from `mcm_data/rep_backup` to your desired storage location.

You can list agent backups using the `list backups` command with the `--agent` option and the site name:

```
mcm> list backups --agent mysite;
+------------+-------+---------+---------------------+-------+--------------+
| BackupId   | Agent | Host    | Timestamp           | Files | Comment      |
+------------+-------+---------+---------------------+-------+--------------+
| 1522914101 | 0     | tonfisk | 2020-04-05 07:41:41Z | 5     | Agent backup |
| 1522914105 | 0     | tonfisk | 2020-04-05 07:41:45Z | 5     | Agent backup |
| 1522914121 | 0     | tonfisk | 2020-04-05 07:42:01Z | 5     | Agent backup |
+------------+-------+---------+---------------------+-------+--------------+
3 rows in set (0.00 sec)
```

To restore the backup for an agent:

- Wipe the contents of the agent's `mcm_data/rep` folder
- Delete the metadata files `high_water_mark` and `repchksum` from the `mcm_data` folder
- Copy the contents in the `mcm_data/rep_backup/timestamp/rep` folder back into the `mcm_data/rep` folder
- Copy the metadata files `high_water_mark` and `repchksum` from the `mcm_data/rep_backup/timestamp` folder back into the `mcm_data` folder
- Restart the agent

The steps are illustrated below:

```
mysql@tonfisk$ cd mcm_data

mysql@tonfisk$ cp mcm_data/rep_backup/timestamp/rep/* ./rep/

mysql@tonfisk$ cp mcm_data/rep_backup/timestamp/high_water_mark ./

mysql@tonfisk$ cp mcm_data/rep_backup/timestamp/repchksum ./

mysql@tonfisk$ mcm9.6.0/bin/mcmd
```

The backup may be manually restored on just one, or more than one agents. If backup is restored for only one agent on, say, host A, host A will contact the other agents of the site to make them recover their repositories from host A using the usual mechanism for agent recovery. If all agents on all hosts are restored and restarted manually, the situation will be similar to the normal restarting all agents after stopping them at slightly different points in time.

If configuration changes has been made to the cluster since the restored backup was created, the same changes must be made again after the agent restores have been completed, to ensure that the agents' configurations match those of the actual running cluster. For example: sometime after a backup was done, a `set MaxNoOfTables:ndbmtd=500 mycluster` command was issued and soon afterward, something happened and corrupted the agent repository; after the agent backup was restored, the same `set` command has to be rerun in order to update the `mcmd` agents' configurations. While the command does not effectively change anything on the cluster itself, after it has been run, a rolling restart of the cluster processes using the `restart cluster` command is still required.

4.8 Restoring a MySQL Cluster Manager Agent with Data from Other Agents

Sometimes, an `mcmd` agent can fail to restart after a failure because its configuration store has been corrupted (for example, by an improper shutdown of the host). If there is at least one other `mcmd` agent that is still functioning properly on another host of the cluster, you can restore the failed agent by the following steps:

- Make sure the `mcmd` agent has really been stopped.
- Restart `mcmd` with the `--initial` option, which wipes the contents of the `rep` folder after backing them up and then starts the agent.

The agent then recovers the configuration store from other agents on the other hosts.

However, if all the `mcmd` agents for the cluster are malfunctioning, you will have to do one of the following:

- Restore one of the agents first using an agent backup (see Section 4.7, “Backing Up and Restoring MySQL Cluster Manager Agents” for details), and then restore the other agents with it .

- Recreate the whole cluster and restore it using a cluster backup (see Section 4.6, “MySQL NDB Cluster Backup and Restore Using MySQL Cluster Manager” for details).

4.9 Setting Up MySQL NDB Cluster Replication with MySQL Cluster Manager

This section provides sample steps for setting up a MySQL NDB Cluster replication with a single replication channel using the MySQL Cluster Manager.

Before trying the following steps, it is recommended that you first read NDB Cluster Replication to familiarize yourself with the concepts, requirements, operations, and limitations of MySQL NDB Cluster replication.

1. Create and start a source cluster:

```
mcm> create site --hosts=tonfisk msite;

mcm> add package --basedir=/usr/local/cluster-mgt/cluster-9.6.0 9.6.0;

mcm> create cluster -P 9.6.0 -R \
       ndb_mgmd@tonfisk,ndbmtd@tonfisk,ndbmtd@tonfisk,mysqld@tonfisk,mysqld@tonfisk,ndbapi@*,ndbapi@* \
       source;

mcm> set portnumber:ndb_mgmd=4000 source;

mcm> set port:mysqld:51=3307 source;

mcm> set port:mysqld:50=3306 source;

mcm> set server_id:mysqld:50=100 source;

mcm> set log_bin:mysqld:50=binlog source;

mcm> set binlog_format:mysqld:50=ROW source;

mcm> set ndb_connectstring:mysqld:50=tonfisk:4000 source;

mcm> start cluster source;
```

2. Create and start a replica cluster (we begin with creating a new site called “ssite” just for the replica cluster; you can also skip that and put the source and replica cluster hosts under the same site instead):

```
mcm> create site --hosts=flundra ssite;

mcm> add package --basedir=/usr/local/cluster-mgt/cluster-9.6.0 9.6.0;

mcm> create cluster -P 9.6.0 -R \
      ndb_mgmd@flundra,ndbmtd@flundra,ndbmtd@flundra,mysqld@flundra,mysqld@flundra,ndbapi@*,ndbapi@* \
      replica;

mcm> set portnumber:ndb_mgmd=4000 replica;

mcm> set port:mysqld:50=3306 replica;

mcm> set port:mysqld:51=3307 replica;

mcm> set server_id:mysqld:50=101 replica;

mcm> set ndb_connectstring:mysqld:50=flundra:4000 replica;

mcm> set replica_skip_errors:mysqld=all replica;

mcm> start cluster replica;
```

3. Create a replica account (with the user name “myreplica” and password “mypw”) on the source cluster with the appropriate privilege by logging into the source replication client (`mysqlM`) and issuing the following statements:

```
mysqlM> CREATE USER 'myreplica'@'flundra' IDENTIFIED BY 'mypw';
mysqlM> GRANT REPLICATION SLAVE ON *.* TO 'myreplica'@'flundra';
```

4. Log in to the replica cluster client (`mysqlS`) and issue the following statements:

```
mysqlS> CHANGE REPLICATION SOURCE TO
    -> SOURCE_HOST='tonfisk',
    -> SOURCE_PORT=3306,
    -> SOURCE_USER='myreplica',
    -> SOURCE_PASSWORD='mypw';
```

5. Start replication by issuing the following statement with the replica cluster client:

```
mysqlS> START REPLICA;
```

The above example assumes that the source and replica clusters are created at about the same time, with no data on both before replication starts. If the source cluster has already been operating and has data on it when the salve cluster is created, after step 3 above, follow these steps to transfer the data from the source cluster to the replica cluster and prepare the replica cluster for replication:

1. Back up your source cluster using the `backup cluster` command of MySQL Cluster Manager:

```
mcm> backup cluster source;
```

Note

Only `NDB` tables are backed up by the command; tables using other MySQL storage engines are ignored.

2. Look up the backup ID of the backup you just made by listing all backups for the source cluster:

```
mcm> list backups source;
+----------+--------+---------+---------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment |
+----------+--------+---------+---------------------+-------+---------+
| 1        | 1      | tonfisk | 2014-10-17 20:03:23 | 1     |         |
| 1        | 2      | tonfisk | 2014-10-17 20:03:23 | 1     |         |
| 2        | 1      | tonfisk | 2014-10-17 20:09:00 | 1     |         |
| 2        | 2      | tonfisk | 2014-10-17 20:09:00 | 1     |         |
+----------+--------+---------+---------------------+-------+---------+
```

From the output, you can see that the latest backup you created has the backup ID “2”, and backup data exists for node “1” and “2”.

3. Using the backup ID and the related node IDs, identify the backup files just created under `/mcm_data/clusters/cluster_name/node_id/data/BACKUP/BACKUP-backup_id/` in the source cluster's installation directory (in this case, the files under the `/mcm_data/clusters/source/1/data/BACKUP/BACKUP-2` and `/mcm_data/clusters/source/2/data/BACKUP/BACKUP-2`), and copy them over to the equivalent places for the replica cluster (in this case, `/mcm_data/clusters/replica/1/data/BACKUP/BACKUP-2` and `/mcm_data/clusters/replica/2/data/BACKUP/BACKUP-2` under the replica cluster's installation directory). After the copying is finished, use the following command to check that the backup is now available for the replica cluster:

```
mcm> list backups replica;
+----------+--------+---------+---------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment |
+----------+--------+---------+---------------------+-------+---------+
| 2        | 1      | flundra | 2014-10-17 21:19:00 | 1     |         |
| 2        | 2      | flundra | 2014-10-17 21:19:00 | 1     |         |
+----------+--------+---------+---------------------+-------+---------+
```

4. Restore the backed up data to the replica cluster (note that you need an unused `ndbapi` slot for the `restore cluster` command to work):

```
mcm> restore cluster --backupid=2 replica;
```

5. On the source cluster client, use the following command to identify the correct binary log file and position for replication to start:

```
mysqlM> SHOW MASTER STATUS\G;
*************************** 1. row ***************************
             File: binlog.000017
         Position: 2857
     Binlog_Do_DB:
 Binlog_Ignore_DB:
Executed_Gtid_Set:
```

6. On the replica cluster client, provide to the replica cluster the information of the source cluster, including the binary log file name (with the `SOURCE_LOG_FILE` option) and position (with the `SOURCE_LOG_POS` option) you just discovered in step 5 above:

```
mysqlS> CHANGE REPLICATION SOURCE TO
    -> SOURCE_HOST='tonfisk',
    -> SOURCE_PORT=3306,
    -> SOURCE_USER='myreplica',
    -> SOURCE_PASSWORD='mypw',
    -> SOURCE_LOG_FILE='binlog.000017',
    -> SOURCE_LOG_POS=2857;
```

7. Start replication by issuing the following statement with the replica cluster client:

```
mysqlS> START REPLICA;
```

As an alternative to these steps, you can also follow the steps described in NDB Cluster Backups With NDB Cluster Replication to copy the data from the source to the replica and to specify the binary log file and position for replication to start.

4.10 Using Encrypted Connections for MySQL Cluster Manager Agents and Clients

MySQL Cluster Manager supports secure connections using TLS for the following:

- `mcm` client and `mcmd` agent connections
- `mcmd` agent and `mysqld` node connections
- `mcmd` agent connections

The following options are used to configure the secure connections (see the option descriptions for details):

- `ssl_ca`
- `ssl_cert`
- `ssl_key`
- `ssl_mode`
- `ssl_cipher`

Enable and Disable Secure Connections. Secure connections can be enabled or disabled by configuring the relevant options in the `mcmd` configuration file in the `mcmd` section:

```
[mcmd]
ssl_key = /absolute/path/to/key
ssl_cert = relative/path/cert
ssl_ca = /path/to/ca_cert
```

The options can also be configured at the command line:

```
$> mcmd --mcmd.ssl_key=/absolute/path/to/key --mcmd.ssl_cert=relative/path/cert --mcmd.ssl_ca=/path/to/
```

`ssl_key` and `ssl_cert` may specify the file name of the TLS key and certificate. Both absolute and relative paths are allowed—relative paths are relative to the current working directory.

To enable or disable secure connections, stop all agents, reconfigure the secure connection options, and restart all agents.

Client Connections. With `ssl_ca` specified, `mcmd` enforces client certificates validation. The mcm client should then provide `ssl_key` and `ssl_cert` information when connecting

```
$> ./bin/mcmd --mcmd.ssl_key=/foo/server-key.pem --mcmd.ssl_cert=/foo/server-cert.pem --mcmd.ssl_ca=/fo
MySQL Cluster Manager 9.6.0 (64bit) started

$> ./bin/mcm --ssl-key=/foo/client-key.pem --ssl-cert=/foo/client-cert.pem
Welcome to the MySQL Cluster Manager client. Commands end with ; or \g.
Your connection id is 0
Agent version: 9.6.0 MySQL Cluster Manager
```

The client may also connect using `--ssl-mode=VERIFY_CA` and `--ssl-ca=cacert.pem` to validate certificates from the client side:

```
$> ./bin/mcm --ssl-mode=VERIFY_CA --ssl-ca=/foo/cacert.pem --ssl-key=/foo/client-key.pem --ssl-cert=/fo
Welcome to the MySQL Cluster Manager client. Commands end with ; or \g.
Your connection id is 0
Agent version: 9.6.0 MySQL Cluster Manager
```

Information on Secure Connections. The `show settings` command has an `--tls` option to show the TLS-specific settings

```
$> ./mcm -e 'show settings --tls'
+---------+------------+----------+
| Section | Key        | Value    |
+---------+------------+----------+
| mcmd    | ssl_ca     |          |
| mcmd    | ssl_cert   |          |
| mcmd    | ssl_cipher |          |
| mcmd    | ssl_key    |          |
| mcmd    | ssl_mode   | DISABLED |
+---------+------------+----------+
```

The `show variables` commands shows the supported tls versions and the supported SSL ciphers:

```
$> ./mcm -e 'show variables'
+--------------------------+--------------------------+
| Variable_name            | Value                    |
+--------------------------+--------------------------+
| auto_increment_increment | 1                        |
| character_set_client     | latin1                   |
| ssl_cipher_list          | LIST-OF-SUPPORTED-CIPHERS |
| tls_version_list         | TLSv1.2,TLSv1.3          |
+--------------------------+--------------------------+
```

The `show status` command, used with no operands, shows runtime information of the connected mcmd, including the TLS version and the cipher in use::

```
$> ./mcm -e 'show status'
+--------------+-----------------------------------+
| Property     | Value                             |
+--------------+-----------------------------------+
| agent number | 1                                 |
```

```
| cwd          | /path/to/current/working/directory |
| max_msg_id   | 234                                |
| max_synode   | {1a2b3c4d 0 234}                   |
| ssl_cipher   | ECDCH-THE-CIPHER-NAME              |
| tls_version  | TLSv1.3                            |
| uptime       | 45                                 |
| version      | 9.6.0                              |
+--------------+------------------------------------+
```

4.11 Using TLS Connections for NDB Clusters

MySQL Cluster Manager 9.6 supports TLS Link Encryption for NDB Cluster, which is available for NDB Cluster 8.3.0 and later. This section describes a few scenarios for using MySQL Cluster Manager to configure or manage TLS connections in an NDB Cluster.

Create a new cluster with TLS enabled on initial startup

Create a site, package, and a cluster with the desired configuration—see Section 4.4, “Setting Up MySQL NDB Clusters with MySQL Cluster Manager” for instructions. Then follow the steps for a basic or a user-defined setup.

Basic setup. Create the CA and certificates for the cluster; this also defines `--ndb-tls-search-path` for any managed process in the cluster:

```
mcm> create certs mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Certificates created successfully |
+----------------------------------+
1 row in set (8.56 sec)
```

Verify `--ndb-tls-search-path` settings:

```
mcm> get -d ndb-tls*: mycluster;
+--------------------+---------------------------------------+----------+---------+----------+---------+
| Name               | Value                                 | Process1 | NodeId1 | Process2 | NodeId2 |
+--------------------+---------------------------------------+----------+---------+----------+---------+
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndb_mgmd | 145     |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndbmtd   | 1       |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndbmtd   | 2       |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | mysqld   | 146     |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | mysqld   | 147     |          |         |
+--------------------+---------------------------------------+----------+---------+----------+---------+
5 rows in set (0.10 sec)
```

User-defined setup. Set `--ndb-tls-search-path` (the command is split into multiple lines for easy reading only; it should be entered in a single line):

```
mcm> set
     ndb_tls_search_path:ndb_mgmd=/foo/mcm_data/clusters/mycluster/certs,
     ndb_tls_search_path:ndbmtd=/foo/mcm_data/clusters/mycluster/certs,
     ndb_tls_search_path:mysqld=/foo/mcm_data/clusters/mycluster/certs
     mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (3.17 sec)
```

Ensure `--ndb-tls-search-path` is set correctly for all managed processes (and issue more `set` commands for corrections, if needed):

```
mcm> get -d ndb-tls*: mycluster;
+--------------------+---------------------------------------+----------+---------+----------+---------+
| Name               | Value                                 | Process1 | NodeId1 | Process2 | NodeId2 |
```

```
+--------------------+---------------------------------------+----------+---------+----------+-------
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndb_mgmd | 145     |          |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndbmtd   | 1       |          |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndbmtd   | 2       |          |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | mysqld   | 146     |          |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | mysqld   | 147     |          |
+--------------------+---------------------------------------+----------+---------+----------+-------
5 rows in set (0.10 sec)
```

For both kinds of setups, create CA and certificates for the cluster (notice that the certificates are only loaded once by the processes at startup):

```
mcm> create certs mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Certificates created successfully |
+----------------------------------+
1 row in set (8.56 sec)
```

Enable `RequireTls` for `ndb_mgmd` and datanodes with another `set` command.

```
mcm> set RequireTls:ndb_mgmd=true,RequireTls:ndbmtd=true mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (3.56 sec)
```

Optionally one may also enable `RequireCertificate`, or set `--ndb-mgm-tls mode` to `strict`:

```
mcm> set RequireCertificate:ndb_mgmd=true,RequireCertificate:ndbmtd=true mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (2.53 sec)

mcm> set ndb_mgm_tls:ndb_mgmd=strict,ndb_mgm_tls:ndbmtd=strict mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (2.39 sec)
```

Start the cluster:

```
mcm> start cluster mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Cluster started successfully |
+------------------------------+
1 row in set (1 min 33.62 sec)
```

Cluster is now running with TLS enabled, required, and (optionally) with certificates required for peers, and in strict mode.

Enable TLS for an Existing Cluster

Assuming you have a cluster already created and started by MySQL Cluster Manager, follow these steps to enable TLS connections for it.

Ensure `--ndb-tls-search-path` is set correctly for all managed processes, and issue the needed `set` commands for corrections if needed:

```
mcm> get -d ndb-tls*: mycluster;
+---------------------+----------------------------------------+----------+---------+----------+---------+
| Name                | Value                                  | Process1 | NodeId1 | Process2 | NodeId2 |
+---------------------+----------------------------------------+----------+---------+----------+---------+
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndb_mgmd | 145     |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndbmtd   | 1       |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | ndbmtd   | 2       |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | mysqld   | 146     |          |         |
| ndb_tls_search_path | /foo/mcm_data/clusters/mycluster/certs | mysqld   | 147     |          |         |
+---------------------+----------------------------------------+----------+---------+----------+---------+
5 rows in set (0.10 sec)
```

Create the CA and certificates for the cluster using MySQL Cluster Manager:

```
mcm> create certs mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Certificates created successfully |
+----------------------------------+
1 row in set (8.57 sec)
```

Restart the cluster so that processes load the certificates created (notice that the certificates are only loaded once by the processes at startup):

```
mcm> restart cluster mycluster;
+--------------------------------+
| Command result                 |
+--------------------------------+
| Cluster restarted successfully |
+--------------------------------+
1 row in set (1 min 38.09 sec)
```

Enable `RequireTls` for `ndb_mgmd` and datanodes with another `set` command:

```
mcm> set RequireTls:ndb_mgmd=true,RequireTls:ndbmtd=true mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (1 min 47.38 sec)
```

Cluster is now running with TLS enabled and required.

Optionally, one may also enable `RequireCertificate`, or set either `--ndb-mgm-tls` mode to `strict`:

```
mcm> set RequireCertificate:ndb_mgmd=true,RequireCertificate:ndbmtd=true mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (1 min 42.53 sec)

mcm> set ndb_mgm_tls:ndb_mgmd=strict,ndb_mgm_tls:ndbmtd=strict mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (1 min 48.13 sec)
```

Cluster is now running with TLS enabled, required, and (optionally) with certificates required for peers, and strict mode.

Chapter 5 MySQL Cluster Manager Client Commands

Table of Contents

Identifiers in client commands

Case-sensitivity rules for client commands

Options common to client commands

The sections in this chapter describe commands used in the MySQL Cluster Manager 9.6.0 client for tasks such as defining sites, packages, and MySQL NDB Cluster instances (“clusters”); configuring a MySQL NDB Cluster; and getting the status of a running MySQL NDB Cluster. These commands are issued to the management agent using the `mysql` client program included with the MySQL NDB Cluster distribution (for information about the `mysql` client not specific to using MySQL Cluster Manager, see mysql — The MySQL Command-Line Client). Each MySQL Cluster Manager client command takes the form shown here:

```
instruction [options] [arguments]

options:
    option [option] [...]

option:
        --option-long-name[=value-list]
    |   -option-short-name [value-list]

value-list:
    value[,value[,...]]

arguments:
    argument [argument] [...]
```

Consider the following MySQL Cluster Manager command, which adds a host named `torsk` to the site `mysite`:

```
add hosts --hosts=torsk mysite;
```

In this example, the command contains a `add hosts` instruction. An instruction consists of one or two keywords, such as `set`, or `show status`.

Most command options have short forms, consisting of single letters, in addition to their long forms. Using the short form of the `--hosts` option, the previous example could also be written like this:

```
add hosts -h torsk mysite;
```

The long form of an option must be preceded by a double dash (`--`), and is not case-sensitive (lower case being the canonical form). The short form of an option must be preceded by a single dash (`-`), and is case-sensitive. In either case, the dash character or characters must come immediately before the option name, and there must be no space characters between them. Otherwise, the MySQL Cluster Manager client cannot parse the command correctly. More information about long and short forms of options is given later in this section.

Important

Do not confuse options given to MySQL Cluster Manager client commands with `mysql` client options. A MySQL Cluster Manager client command option is always employed as part of a MySQL Cluster Manager client command; it is *not* passed to the `mysql` client when invoking it.

> In addition, you cannot issue queries or other SQL statements in the MySQL Cluster Manager client. These are not recognized by the client, and are rejected with an error. The converse of this is also true: MySQL Cluster Manager client commands are not recognized by the standard `mysql` client.

The instruction just shown takes the argument `mysite`. The argument is usually an identifier that names the object to be effected; in this case, the command deletes the site whose name matches the argument. (For more information, see Section 5.2.6, "The `create site` Command".)

An additional `--verbose` option can be used for the `create cluster` ,`add process`, and `list hosts` commands. In both cases, using the option causes the command to return a list of the MySQL NDB Cluster processes affected by the command; this includes their node IDs, process types, and the hosts where they are located.

Identifiers in client commands.
A legal MySQL Cluster Manager identifier consists of any sequence of characters from among the following:

- The letters `a` through `z` and `A` through `Z`
- The digits `0` through `9`
- The dash (`-`), period (`.`), and underscore (_) characters

A MySQL Cluster Manager identifier must begin with a letter or digit.

Case-sensitivity rules for client commands.
The rules for case-sensitivity of MySQL Cluster Manager identifiers, commands, command options, process names, and configuration attributes are as follows:

- *Identifiers are case-sensitive*. For example, `delete site mycluster` cannot be used to delete a site named `myCluster`.
- *Command keywords and the long forms of command options are case-insensitive*. For example, any of the three commands `delete cluster mycluster`, `DELETE CLUSTER mycluster`, and `DeLeTe cLuStEr mycluster` works to delete the MySQL NDB Cluster instance named `mycluster`.

  In this manual, we show command keywords and the long forms of command options in lowercase, but you are not required to follow this convention if you do not wish to do so.
- *The short forms of command options are case-sensitive*. For example, for the `backup cluster` command, the `-w` (lowercase) is the short form of the `--waitstarted` option, but `-W` (uppercase) is the short form of the `--waitcompleted` option.
- *Names of MySQL NDB Cluster processes are case-insensitive*. For example, either of the commands `get --include-defaults DataMemory:ndbd mycluster` or `get --include-defaults datamemory:NDBD mycluster` reports the data memory allocated for each `ndbd` process in the cluster named `mycluster`.

  In this manual, we show names of MySQL NDB Cluster processes in lowercase. You are not required to follow this convention if you do not wish to do so; however, since the corresponding executables are named and must be invoked in lowercase, we suggest that you use lowercase.
- *Configuration attribute names are case-insensitive*. For example, either of the commands `get --include-defaults DataMemory:ndbd mycluster` or `get --include-defaults datamemory:ndbd mycluster` returns the data memory allocated for each `ndbd` process in the cluster named `mycluster`; either of the commands `set engine-condition-pushdown:mysqld:4=0 mycluster` or `set Engine-Condition-Pushdown:mysqld:4=0`

`mycluster` disables the condition pushdown optimization in the `mysqld` process having the node ID `4` in the MySQL NDB Cluster named `mycluster`.

Note

Configuration attributes in the MySQL Cluster Manager derive from two different sources: MySQL NDB Cluster configuration parameters, and MySQL Server options. MySQL NDB Cluster configuration parameters are case-insensitive, but their canonical forms use upper camelcase (that is, medial capitalization including the first letter). This means that whether you set a value for data memory using the MySQL Cluster Manager client or in the `config.ini` file, you can refer to it as `DataMemory`, `datamemory`, or `dATAmEMORY` without any negative impact. However, MySQL Server command-line options are case-sensitive and use only lowercase. This means that, for example, `set Engine-Condition-Pushdown:mysqld:4=0 mycluster` in the MySQL Cluster Manager client works to disable condition pushdown in the indicated `mysqld` process, but if you invoke the `mysqld` executable from a system prompt using `--Engine-Condition-Pushdown=0`, `mysqld` fails to start.

In this manual, for easy recognition, we show configuration attribute names as having the same lettercase used in other MySQL documentation; thus, we always refer to `DataMemory`, rather than `datamemory` or `DATAMEMORY`, and `engine-condition-pushdown`, rather than `Engine-Condition-Pushdown` or `ENGINE-CONDITION-PUSHDOWN`. While you are not required to do this when using MySQL Cluster Manager, we suggest that you also follow this convention.

Note

Values that contain space characters must be quoted using single quote (`'`) characters. For example, if you wish to define a package named `mypackage` for a site named `mysite` using `/usr/local/mysql cluster/9.6` (where a space occurs between `mysql` and `cluster`) as the path to the base directory on all hosts, the correct command would be `add package --basedir='/usr/local/mysql cluster/9.6' mypackage`.

To decrease the possibility of errors in reading and entering MySQL Cluster Manager commands, we recommend avoiding the use of space characters whenever possible.

Each command must end with a terminator character. By default, this is the semicolon (`;`) character. However, the sequences `\g` and `\G` are also supported as command terminators. The `\G` terminator causes the output to be vertically formatted (the same as in the standard `mysql` client), as shown in this example:

```
mcm> get DataMemory mycluster\G
*************************** 1. row ***************************
    Name: DataMemory
   Value: 500M
Process1: ndbd
     Id1: 2
Process2:
     Id2:
   Level: Process
 Comment:
*************************** 2. row ***************************
    Name: DataMemory
   Value: 500M
Process1: ndbd
     Id1: 3
Process2:
     Id2:
   Level: Process
 Comment:
```

```
2 rows in set (0.22 sec)
```

By convention (for reasons of readability), we do not normally include the command terminator when showing the syntax for a command in Backus-Naur format or when including a MySQL Cluster Manager command inline in this text. However, if you do not use a statement terminator when you enter the command in the MySQL Cluster Manager client, the client displays a special “waiting...” prompt `->` until you supply a terminator, as shown here:

```
mcm> list sites
  ->
  ->
  ->
  -> ;
Empty set (1.50 sec)
```

(The is the same as the behavior of the `mysql` client when you fail to end a statement with a terminator.)

A command option can also in many cases accept (or even require) a set of one or more *`values`*. The next example includes such an option, and also demonstrates setting of multiple values in a single option by passing them to the option as a comma-separated list:

```
mcm> create site --hosts=tonfisk,flundra mysite;
+---------------------------+
| Command result            |
+---------------------------+
| Site created successfully |
+---------------------------+
1 row in set (7.41 sec)
```

The command just shown creates a site named `mysite`, consisting of two hosts named `tonfisk` and `flundra`. (See Section 5.2.6, “The `create site` Command”, for more information about this command.) Since we used the long form of the `--hosts` option, we were required to use an equals sign (`=`) to mark the end of the option name and the beginning of the values list. You must not insert any space characters before or after the equal sign; doing so causes an error, as shown here:

```
mcm> create site --hosts =grindval,haj yoursite;
ERROR 7 (00MGR): Option --hosts requires a value
mcm> create site --hosts= grindval,haj yoursite;
ERROR 7 (00MGR): Option --hosts requires a value
```

The short form of an option does not use an equal sign. Instead, the value-list is separated from the option by a space. Using the `-h` option, which is the short form of the `--hosts` option, the previous `create site` command can be entered and executed like this:

```
mcm> create site -h tonfisk,flundra mysite;
+---------------------------+
| Command result            |
+---------------------------+
| Site created successfully |
+---------------------------+
1 row in set (7.41 sec)
```

The short forms of options actually accept multiple spaces between the option name and the values list; however, a single space is sufficient. If you omit the space, or try to use an equal sign, the command fails with an error, as shown here:

```
mcm> create site -htonfisk,flundra mysite;
ERROR 6 (00MGR): Illegal number of operands
mcm> create site -h=tonfisk,flundra mysite;
ERROR 3 (00MGR): Illegal syntax
```

Any option value containing one or more whitespace characters, one or more dash characters (`-`), or both, must be quoted using single quotation marks. Multiple values should be separated by commas only; do not insert spaces before or after any of the commas. Using spaces before or after the commas in a list of values causes the command to fail with an error, as shown here:

```
mcm> create site --hosts=tonfisk, flundra mysite;
ERROR 6 (00MGR): Illegal number of operands
```

As you can see from the examples just shown, a MySQL Cluster Manager client command returns a result set, just as an SQL statement does in the standard `mysql` client. The result set returned by a MySQL Cluster Manager client command consists of one of the following:

- **A single row that contains a message indicating the outcome of the command.** The `create site` command in the last example returned the result `Site created successfully`, to inform the user that the command succeeded.
- **One or more rows listing requested objects or properties.** An example of such a command is `list processes`, as shown here:

  ```
  mcm> list processes mycluster;
  +--------+----------+----------+
  | NodeId | Name     | Host     |
  +--------+----------+----------+
  | 49     | ndb_mgmd | flundra  |
  | 1      | ndbd     | tonfisk  |
  | 2      | ndbd     | grindval |
  | 50     | mysqld   | haj      |
  | 51     | mysqld   | torsk    |
  | 52     | ndbapi   | *        |
  +--------+----------+----------+
  6 rows in set (0.03 sec)
  ```

  In the case of `list processes`, each row in the result contains the ID and type of a node in the MySQL NDB Cluster named `mycluster`, together with the name of the host on which the process is running.
- **An empty result set.** This can occur with one of the `list` commands when there is nothing to report, such as when `list sites` is used before any sites have been created:

  ```
  mcm> list sites;
  Empty set (0.72 sec)
  ```

Each command must be entered separately; it is not possible to combine multiple commands on a single line.

Options common to client commands.
The following three options are common to most MySQL Cluster Manager client commands:

1. `--help` (short form: `-?`): Common to all client commands. Provides help output specific to the given command. See Section 5.1, “Online Help and Information Commands”, for more information about this option.
2. `--force` (short form `-f`): Causes any safety checks to be bypassed when executing the command. For example, `delete cluster` *`mycluster`* normally fails if any of the MySQL NDB Cluster processes in the MySQL NDB Cluster named *`mycluster`* are running; however, `delete cluster --force` *`mycluster`* forces the shutdown of *`mycluster`*, followed by the deletion of *`mycluster`* from MySQL Cluster Manager's inventory.

   The `--force` option is supported for the following MySQL Cluster Manager client commands:

   - `delete site`
   - `start cluster`
   - `restart cluster`
   - `stop cluster`
   - `delete cluster`

- `upgrade cluster`
- `add process`
- `start process`
- `stop process`
- `remove process`
- `set`
- `reset`

5.1 Online Help and Information Commands

Online help is available in the MySQL Cluster Manager client for MySQL Cluster Manager client commands. The client can provide both general and command-specific information. In addition, you can obtain information about `mysql` client commands that are independent of the MySQL server and thus are also available for use when connected to the MySQL Cluster Manager agent.

Listing MySQL Cluster Manager client commands.
For a list of all commands with brief descriptions, use the `list commands` command, as shown here:

```
mcm> list commands;
+--------------------------------------------------------------------------+
| Help                                                                     |
+--------------------------------------------------------------------------+
| COMMANDS                                                                 |
|                                                                          |
| abort backup           Abort an ongoing cluster backup.                  |
| add hosts              Add hosts to site.                                |
| add package            Add a package alias.                              |
| add process            Add cluster process.                              |
| autotune               Autotune a cluster to given use-case template.    |
| backup agents          Backup the agents repository and metadata.        |
| backup cluster         Backup a cluster.                                 |
| change log-level       Change the log-level                              |
| change process         Change process type.                              |
| collect logs           Collect log files.                                |
| create cluster         Create a cluster.                                 |
| create site            Create a site.                                    |
| delete cluster         Delete a cluster.                                 |
| delete package         Delete a package.                                 |
| delete site            Delete a site.                                    |
| get                    Get configuration variables.                      |
| import cluster         Import a running cluster.                         |
| import config          Import the configuration of a running cluster.    |
| list backups           List backup images.                               |
| list clusters          List all clusters.                                |
| list commands          List the help text.                               |
| list hosts             List hosts in site.                               |
| list nextnodeids       List next nodeids to be allocated.                |
| list packages          List all packages.                                |
| list processes         List processes.                                   |
| list sites             List all sites.                                   |
| remove hosts           Remove hosts from site.                           |
| remove process         Remove a cluster process.                         |
| reset                  Reset configuration variables.                    |
| restart cluster        Restart a cluster.                                |
| restore cluster        Restore a cluster.                                |
| rotate log             Rotate the mcmd log.                              |
| set                    Set configuration variables.                      |
| show settings          Show agent settings.                              |
| show status            Show cluster, process, operation or backup status. |
| start cluster          Start a cluster.                                  |
| start process          Start a cluster process.                          |
```

```
| stop agents            Stop agents in site.                                |
| stop cluster           Stop a cluster.                                     |
| stop process           Stop a cluster process.                             |
| upgrade cluster        Upgrade a cluster.                                  |
| version                Print version information.                          |
|                                                                            |
| GLOBAL OPTIONS                                                             |
| Options that can be used with all commands                                 |
|                                                                            |
|     --help|-?             Print detailed help.                             |
|                                                                            |
| Use '<COMMAND> --help' to see verbose help for individual commands.        |
+----------------------------------------------------------------------------+
51 rows in set (0.03 sec)
```

Obtaining information about specific MySQL Cluster Manager client commands.
To obtain more detailed help specific to a given command, invoke the command using the `--help` option, as shown in this example:

```
mcm> create site --help;
+-------------------------------------------------------------------+
| Help                                                              |
+-------------------------------------------------------------------+
|                                                                   |
| create site [options] <sitename>                                  |
|                                                                   |
|     Creates a site from the hosts listed in --hosts.              |
|                                                                   |
|     Required options:                                             |
|     --hosts|-h             Comma separated list of hostnames.     |
|                            Format: --hosts = <host>[,<host>]*.    |
|                                                                   |
+-------------------------------------------------------------------+
9 rows in set (0.00 sec)
```

For any MySQL Cluster Manager client command, the `--help` option may be abbreviated to `-?`:

```
mcm> list processes -?;
+-------------------------------------------------------------+
| Help                                                        |
+-------------------------------------------------------------+
|                                                             |
| list processes <sitename>                                   |
|                                                             |
|       Lists all processes defined in the specified cluster. |
+-------------------------------------------------------------+
4 rows in set (0.00 sec)
```

As mentioned elsewhere in this manual (see Chapter 5, *MySQL Cluster Manager Client Commands*), many other MySQL Cluster Manager command options have short forms as well. These are included in the documentation for each command. You can also find out what these are for a given command by invoking it with the `--help` or `-?` option.

You can obtain the release version of the MySQL Cluster Manager software in use from the output of the `version` command.

`mysql` client commands in the MySQL Cluster Manager client.
You can also use most standard `mysql` client commands in the MySQL Cluster Manager client (but *not* SQL statements, which depend on being connected to a MySQL server), such as `prompt`, `quit`, and `status`. For example, the output of the status command when connected to the MySQL Cluster Manager agent looks something like this (depending on the exact version of the client and agent that you are using and possibly other factors):

```
mcm> status
--------------
./bin/mcm  Ver 9.6.0 for Linux on x86_64 (MySQL Enterprise Server - Commercial)

Connection id:       3
Current database:    <n/a>
```

```
Current user:        mcmd
SSL:      Not in use
Current pager:  stdout
Using outfile:  ''
Using delimiter:     ;
Server version:      9.6.0 MySQL Cluster Manager
Protocol version:    10
Connection:          127.0.0.1 via TCP/IP
Server characterset: latin1
Db     characterset: latin1
Client characterset: latin1
Conn.  characterset: latin1
TCP port:       1862
Binary data as:      Hexadecimal
Uptime:   4 hours 11 min 54 sec

Agent no: 0  Connections: 1  Max msg id: {658358e0 102 0}
--------------
```

Note

You may use the command delimiter with `mysql` client commands, but you are not required to do so. For instance, assuming that the delimiter in use was the default semicolon (`;`) character, we could have executed the `status` command like this:

```
mcm> status;
--------------
/home/jon/bin/mcm/cluster/bin/mysql  Ver 14.14 Distrib 9.6.0,...
```

A particularly useful `mysql` client command that you can also employ with `mcm` is the `source` command (short form: `\.`), which you can use for executing scripts containing MySQL Cluster Manager client commands. On a Linux system, you might have a text file in your home directory named `get-attributes.mcm`, whose contents are shown here:

```
get :ndb_mgmd mycluster\G
get :ndbd mycluster\G
get :mysqld mycluster\G
```

Assuming that you have created a cluster named `mycluster`, you can run this script in the client; the results vary according to how this cluster is actually configured, but should be similar to this:

```
mcm> \. ~/get-attributes.mcm
mcm> get :ndb_mgmd mycluster\G
*************************** 1. row ***************************
    Name: DataDir
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/49/data
Process1: ndb_mgmd
 NodeId1: 49
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 2. row ***************************
    Name: HostName
   Value: flundra
Process1: ndb_mgmd
 NodeId1: 49
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 3. row ***************************
    Name: NodeId
   Value: 49
Process1: ndb_mgmd
 NodeId1: 49
Process2:
```

```
 NodeId2:
   Level:
 Comment: Read only
*************************** 4. row ***************************
    Name: PortNumber
   Value: 1186
Process1: ndb_mgmd
 NodeId1: 49
Process2:
 NodeId2:
   Level: Process
 Comment:
4 rows in set (0.09 sec)

mcm> get :ndbd mycluster\G
*************************** 1. row ***************************
    Name: DataDir
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/1/data
Process1: ndbd
 NodeId1: 1
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 2. row ***************************
    Name: HostName
   Value: tonfisk
Process1: ndbd
 NodeId1: 1
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 3. row ***************************
    Name: NodeId
   Value: 1
Process1: ndbd
 NodeId1: 1
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 4. row ***************************
    Name: DataDir
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/2/data
Process1: ndbd
 NodeId1: 2
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 5. row ***************************
    Name: HostName
   Value: grindval
Process1: ndbd
 NodeId1: 2
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 6. row ***************************
    Name: NodeId
   Value: 2
Process1: ndbd
 NodeId1: 2
Process2:
 NodeId2:
   Level:
 Comment: Read only
6 rows in set (0.10 sec)

mcm> get :mysqld mycluster\G
```

```
*************************** 1. row ***************************
    Name: datadir
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/50/data
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 2. row ***************************
    Name: HostName
   Value: haj
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 3. row ***************************
    Name: log_error
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/50/data/mysqld_50_out.err
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 4. row ***************************
    Name: ndb_nodeid
   Value: 50
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 5. row ***************************
    Name: ndbcluster
   Value:
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 6. row ***************************
    Name: NodeId
   Value: 50
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 7. row ***************************
    Name: port
   Value: 3306
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 8. row ***************************
    Name: socket
   Value: /tmp/mysql.mycluster.50.sock
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
```

```
*************************** 9. row ***************************
    Name: tmpdir
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/50/data/tmp
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 10. row ***************************
    Name: datadir
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/51/data
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 11. row ***************************
    Name: HostName
   Value: torsk
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 12. row ***************************
    Name: log_error
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/51/data/mysqld_51_out.err
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 13. row ***************************
    Name: ndb_nodeid
   Value: 51
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 14. row ***************************
    Name: ndbcluster
   Value:
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 15. row ***************************
    Name: NodeId
   Value: 51
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 16. row ***************************
    Name: port
   Value: 3307
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
```

```
*************************** 17. row ***************************
    Name: socket
   Value: /tmp/mysql.mycluster.51.sock
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 18. row ***************************
    Name: tmpdir
   Value: /home/jon/bin/mcm/mcm_data/clusters/mycluster/51/data/tmp
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
18 rows in set (0.05 sec)

mcm>
```

Note

You are not returned to the client prompt until the script has finished executing.

Similarly, on Windows, you can create a batch file using Notepad or another text editor, copy the same `get` commands as shown previously into it, and save it as `get-attributes.bat` in a convenient location such as the Windows desktop.

You can view a list of available `mysql` client commands using the `help` command. For more information about these, view the `help` output or see mysql Client Commands, in the *MySQL Manual*.

5.2 MySQL Cluster Manager Site and Agent Commands

In this section, we discuss commands used to work with MySQL Cluster Manager management sites. In addition, the `stop agents`, `show settings`, `version`, and `show warnings` commands, which relate to management agents, are also covered in this section.

A *site*, in terms of MySQL NDB Cluster and MySQL Cluster Manager, is a collection of one or more host computers where MySQL Cluster Manager agents are running. Each agent is identified by the combination of two pieces of information:

- The hostname or IP address of the machine where the agent is running
- The number of the port used by the agent for communications

Note

MySQL NDB Cluster makes extremely intensive use of network connections, and DNS lookups can contend with MySQL NDB Cluster and MySQL Cluster Manager for bandwidth, resulting in a negative impact on the performance of MySQL NDB Cluster and the applications using it. For this reason, we recommend that you use numeric IP addresses rather than hostnames for MySQL NDB Cluster and MySQL Cluster Manager host computers whenever feasible.

5.2.1 The `add hosts` Command

```
add hosts {--hosts=|-h }host_list site_name

host_list:
    host[, host[, ...]]
```

This command adds one or more hosts to an existing management site. Agents using the same port as the management site must be running on any hosts added using this command. This command takes two mandatory arguments: a list of hosts (using the `--hosts` option or its short form `-h`), and the name of the site to which the hosts are to be added.

The `--hosts` takes a comma-separated list of one or more hosts to be added to the site.

For example, the following command adds two hosts named `torsk` and `kolja` to management site `mysite`:

```
mcm> add hosts --hosts=torsk,kolja mysite;
+--------------------------+
| Command result           |
+--------------------------+
| Hosts added successfully |
+--------------------------+
1 row in set (0.48 sec)
```

None of the hosts added by this command may already be members of management site *`site_name`*. Do not attempt to add again a host that is already a member of the management site using its secondary (or alternate) IP address—the `mcmd` process on the host is already bound to the IP address that was supplied when the host was first added, and it cannot be bound again to another IP address.

Notes

- This command does not support the `--force` option.
- Do not use `localhost` in the host list, as MySQL Cluster Manager relies on the operating system for host name resolution, and `localhost` might be resolved differently on different systems. Use proper host names for the host list or, preferably, use the IP addresses for the hosts instead.
- When IPv6-enabled Windows systems are used as MySQL NDB Cluster hosts under MySQL Cluster Manager, you must reference these hosts using IPv4 addresses. Otherwise, MySQL Cluster Manager will be unable to connect to the agent processes on those hosts. See Section 6.1, “MySQL Cluster Manager Usage and Design Limitations”.

5.2.2 The `remove hosts` Command

```
remove hosts {--hosts=|-h }host_list site_name

host_list:
    host[, host[, ...]]
```

This command removes one or more hosts from an existing management site. It takes as arguments a required option `--hosts` (or its short form `-h`), whose value is a comma-separated list of one or more hosts to be removed, and the name of the site from which the hosts are to be removed. A number of limitations apply:

- The name of the host to be removed must not be `localhost` or `127.0.0.1`.
- The host to be removed must not have any managed processes from any clusters assigned to them (remove those processes first with the `remove process` command); it can have unmanaged processes assigned to them though (typically, `ndbapi@hostname` or `mysqld@*hostname`).
- There should not be any packages defined with explicit paths pointing to the host to be removed.
- A quorum consists of a majority of hosts (i.e., half of the total number of hosts plus one) must exist for the site both before and after the host's removal, or it will not be possible to execute the `remove host` command.
- You cannot remove the last host from a site; use the `delete site` command instead.

The following command removes two hosts named tonfisk and flundra from the management site mysite:

```
mcm> remove hosts --hosts=tonfisk,flundra mysite;
+---------------------------+
| Command result            |
+---------------------------+
| Hosts removed successfully |
+---------------------------+
1 row in set (0.48 sec)
```

5.2.3 The change log-level Command

```
change log-level [{--hosts=|-h }host_list] log_level site_name

host_list:
    host[,host[,...]]
```

Set the management agent's cluster logging level. This has the same effect as using the logger level option; however, unlike the option, this command can be used at run time and does not require a restart of mcmd. Issuing this command overrides any value for level set in the agent configuration file.

When used with the *log_level* alone without a *host_list* and a *site_name*, this command applies only to the agent the mcm client is connected to. In the following example, the logging level is set to warning only on the host managed directly by the agent to which the mcm client is connected:

```
mcm> change log-level warning;
+--------------------------------+
| Command result                 |
+--------------------------------+
| Log-level changed successfully |
+--------------------------------+
1 row in set (0.00 sec)
```

You can specify the name of a site to be affected by the command. For example, the following invocation of the command applies to the site named mysite:

```
mcm> change log-level debug mysite;
+--------------------------------+
| Command result                 |
+--------------------------------+
| Log-level changed successfully |
+--------------------------------+
1 row in set (0.05 sec)
```

You can also restrict the change to one or more hosts in a given site using the --hosts option (or its short form -h), with multiple host names separated by commas. The following command changes the logging level to debug on the hosts named tonfisk and haj, but not on any other hosts in mysite:

```
mcm> change log-level --hosts=tonfisk,haj debug mysite;
+--------------------------------+
| Command result                 |
+--------------------------------+
| Log-level changed successfully |
+--------------------------------+
1 row in set (0.09 sec)
```

You must specify a site when using the --hosts option; trying to use --hosts alone results in an error.

Accepted values for *log_level* are the same as for the level option: one of debug, note, info, warning, error, system, or fatal. For more detailed information about the meanings and effects of these values, see NDB Cluster Logging Management Commands.

5.2.4 The rotate log Command

```
rotate log [{--hosts=|-h }host_list] [site_name]
```

```
host_list:
    host[,host[,...]]
```

Rotate mcmd logs for the connected MySQL Cluster Manager agent, for agents running on certain hosts, or for agents on all hosts in a management site.

For example, to rotate logs for the agent to which the client session is connected:

```
mcm> rotate log;
+--------------------------+
| Command result           |
+--------------------------+
| Log rotated successfully |
+--------------------------+
1 row in set (0.03 sec)
```

A new log file, with an underscore and then a timestamp inserted into its file name before the file extension, is created as a result:

```
-rw-r-----   1 mcmd cluster    74265 Jul 15 22:45 mcmd.log
-rw-r-----   1 mcmd cluster  1197573 Jul 15 22:45 mcmd_2021-07-15T22-45-28.log
```

To rotate logs for agents on specific hosts like nanna12 and nanna13, use the --hosts option (or its short form -h):

```
mcm> rotate log --hosts=nanna12,nanna13 mysite;
```

To rotate logs on all agents in the management site mysite:

```
mcm> rotate log mysite;
```

5.2.5 The collect logs Command

```
collect logs [cluster_name]
```

This command collects log files and other related files from all hosts. When the name of a cluster (*cluster_name*) is supplied with the command, it collects all the log files (.log, mcmd_eventlog.csv) as well as the configuration files (.ini, .cnf), error files (.err), and trace files (.trace.log.*) used by all processes belonging to the cluster, and also all the agent log files. If *cluster_name* is omitted, only the agent log files are collected.

When an mcmd agent receives the collect logs command from the mcm agent it is connected with, it sets up a TCP server socket using port 0 by default, and lets the operating system assign the actual port number. All agents in the site are then instructed to perform the copying, and each of them spawns a TCP client, which connects to the TCP server socket set up earlier to copy the files over.

To assign a specific port manually for file copying, use the --copy-port option when starting mcmd. Default value for the option is 0. The collect logs command times out if, in 30 seconds, no connections can be established by any of the clients or no incoming connections are detected by the TCP server.

Warning

If a firewall or other networking issues prohibit the TCP clients to connect to the TCP server socket, the collect logs command will never complete.

The collected files are put under the MySQL Cluster Manager data repository (mcm_data in the parent directory of the MySQL Cluster Manager installation directory) by default, or specified by the option --data-folder) inside a folder named collected_files, under which the files are organized under a hierarchy that looks like the following:

```
/mcm_data_repository/collected-files/
|── timestamp/
```

```
|   ├── mcmd/
|   |   ├── mcmd#/
|   |       ├── mcmd.log
|   |       └── mcmd_timestamp.log
|   |       └── mcmd_eventlog.csv
|   |       └── ...
|   ├── cluster_name/
|       ├── node#_nodetype/
|       |   ├── log files (.log)
|       |   ├── configuration files (.ini, .cnf)
|       |   ├── error files (.err)
|       |   └── trace files (.trace.log.*)
|       ├── node#_nodetype/
|       |   ├── log files (.log)
|       |   ├── configuration files (.ini, .cnf)
|       |   ├── error files (.err)
|       |   └── trace files (.trace.log.*)
|       ├── node#_nodetype/
|       |   └── ...
|       └── other/
|           ├── utilities files (.err, .out)
|           └── ...
|
├── timestamp/
|   ├── mcmd/
|   |   ├── mcmd#/
|   |       ├── mcmd.log
|   |       ├── mcmd_timestamp.log
|   |       └── ...
|   ├── cluster_name/
|   |   ├── node#_nodetype/
...
```

For example, the error log for the `mysqld` node of node number 146 is found at:

```
/opt/mcm_data/collected-files/2021-07-31T07:44:05Z/51_mysqld/mysqld_146_out.err
```

5.2.6 The `create site` Command

```
create site {--hosts=|-h }host_list site_name

host_list:
    host[,host[,...]]
```

The `create site` command is used to create a MySQL Cluster Manager management site; that is, a set of MySQL Cluster Manager management agents running on one or more host computers. The command requires a list of one or more hosts where management agents are running and a name for the site. The host list is passed as the value of the `--hosts` option (short form: `-h`).

This is an example of a `create site` command that creates a site named `mysite`, consisting of the hosts `tonfisk` and `flundra`:

```
mcm> create site --hosts=tonfisk,flundra mysite;
+---------------------------+
| Command result            |
+---------------------------+
| Site created successfully |
+---------------------------+
1 row in set (0.31 sec)
```

Tip

You can verify that the site was created as intended, using the `list sites` command, as shown here:

```
mcm> list sites;
```

```
+--------+------+-------+-----------------+
| Site   | Port | Local | Hosts           |
+--------+------+-------+-----------------+
| mysite | 1862 | Local | tonfisk,flundra |
+--------+------+-------+-----------------+
1 row in set (0.06 sec)
```

(See Section 5.2.8, "The list sites Command", for more information about this command.)

Agents must be running on all hosts specified in the --hosts option when create site is executed; otherwise, the command fails with the error Agent on host *host*:*port* is unavailable. The host where the agent used to issue the command is running must be one of the hosts listed. Otherwise, the command fails with the error Host *host_name* is not a member of site *site_name*.

Warning

Moreover, if the client and the agent it is connected to are on the same host, that host must be included in the host list using its host name or its own loopback address (which can be something other than 127.0.0.1 on some systems); otherwise, the cluster might become not restartable in the future.

A given agent may be a member of one site only; if one of the management agents specified in the *host_list* already belongs to a site, the command fails with the error Host *host* is already a member of site *site*.

Notes

- Using localhost as the argument for the --hosts option will result in the creation of a single-host site (consisting of the host on which the command is run) that cannot be scaled up later by the add hosts command. Also notice that you cannot mix localhost with other host names in the host list. Therefore, it is recommended that you use IP addresses (but not any addresses belonging to the localhost subnet 127.*.*.*) or proper host names in the list.
- When IPv6-enabled Windows systems are used as MySQL NDB Cluster hosts under MySQL Cluster Manager, you must reference these hosts using IPv4 addresses. Otherwise, MySQL Cluster Manager will be unable to connect to the agent processes on those hosts. See Section 6.1, "MySQL Cluster Manager Usage and Design Limitations".

5.2.7 The delete site Command

```
delete site site_name
```

The delete site command deletes an existing management site. The command does not stop or remove any agents making up the deleted site; instead, these agents continue to run, and remain available for use in other sites.

The command takes a single argument, the name of the site to be deleted. This example shows the deletion of a management site named mysite:

```
mcm> delete site mysite;
+--------------------------+
| Command result           |
+--------------------------+
| Site deleted successfully |
+--------------------------+
1 row in set (0.38 sec)
```

If the site to be deleted does not exist, the command fails with the error Command requires a site to be defined. If there are any packages referencing hosts belonging to the site, delete site

fails with the error Packages exist in site *site_name*. The command also fails if there are defined any clusters that include hosts belonging to the site.

Note

The management client must be connected to a site in order to be able to delete it.

In addition, if you execute a delete site command with the --force option using one management agent while a different management agent is not running, you must remove the "missing" management agent's site files manually. For more information on site files, see Section 3.4, "MySQL Cluster Manager Configuration File".

5.2.8 The list sites Command

```
list sites
```

This command returns a list of the sites known to the management agent. It does not require any arguments. An example is shown here:

```
mcm> list sites;
+--------+------+-------+-----------------+
| Site   | Port | Local | Hosts           |
+--------+------+-------+-----------------+
| mysite | 1862 | Local | tonfisk,flundra |
+--------+------+-------+-----------------+
1 row in set (0.06 sec)
```

The output of list sites contains the following columns:

- **Site.** The name of the site.
- **Port.** The TCP/IP port used for communications between clients and management agents.
- **Local.** Either one of Local or Remote.
- **Hosts.** A comma-separated list of the hosts making up the site.

5.2.9 The list hosts Command

```
list hosts [--verbose|-v]  site_name
```

The list hosts command is used to obtain a list of the hosts comprising a given management site. The command requires a single argument, the name of the site to be examined. For each host listed, the information returned includes the hostname, status, and version of the management agent software in use, as shown in this example:

```
mcm> list hosts mysite;
+-----------+-----------+---------+
| Host      | Status    | Version |
+-----------+-----------+---------+
| tonfisk   | Available | 9.6.0   |
| flundra   | Available | 9.6.0   |
+-----------+-----------+---------+
2 rows in set (0.16 sec)
```

Status can be one of :

- Available: Agent on the host is active
- Recovery: Agent on the host is in the process of recovering itself
- Unresponsive: Agent on the host rejected an attempt to connect
- Unavailable: Agent on the host is unreachable

If an agent is reported persistently as Unresponsive or Unavailable, you may have to restart it.

If you omit the *site_name* argument, the command fails with an error, as shown here:

```
mcm> list hosts;
ERROR 6 (00MGR): Illegal number of operands
```

Using the --verbose option (short form : -v) causes the command to print additional information on the hosts:

```
mcm> list hosts --verbose mysite;
+---------+-----------+---------+-------+---------+-------------------------+
| Host    | Status    | Version | Cores | Memory  | OS                      |
+---------+-----------+---------+-------+---------+-------------------------+
| tonfisk | Available | 9.6.0   | 1     | 1819 Mb | Linux 4.15.0-147-generic |
| flundra | Available | 9.6.0   | 1     | 1819 Mb | Linux 4.15.0-147-generic |
+---------+-----------+---------+-------+---------+-------------------------+
2 rows in set (0.07 sec)
```

5.2.10 The show settings Command

```
show settings [--hostinfo | --tls]
```

This command lists the current values of a number of mcmd options:

```
mcm> show settings;
+---------+----------------+-----------------------------------------------------+
| Section | Setting        | Value                                               |
+---------+----------------+-----------------------------------------------------+
| DEFAULT | data_folder    | /home/dso/extra3/mcm-9.6.0-cluster-9.6.0/mcm_data |
| DEFAULT | logging_folder | /home/dso/extra3/mcm-9.6.0-cluster-9.6.0          |
| logger  | filename       | mcmd.log                                            |
| logger  | level          | info                                                |
| mcmd    | bind_port      | 1862                                                |
| mcmd    | copy_port      | 0                                                   |
| mcmd    | mcmd_password  | ********                                            |
| mcmd    | mcmd_user      | mcmd                                                |
| mcmd    | xcom_port      | 18062                                               |
+---------+----------------+-----------------------------------------------------+
9 rows in set (0.10 sec)
```

Using the --hostinfo option makes the command print out information on the host that the mcm client is connected to:

```
mcm> show settings --hostinfo;
+-----------------+------------------------------+
| Property        | Value                        |
+-----------------+------------------------------+
| Hostname        | localhost.localdomain        |
| Platform        | Linux 3.13.11-100.fc19.x86_64 |
| Processor cores | 1                            |
| Total memory    | 1819 Mb                      |
+-----------------+------------------------------+
4 rows in set (0.00 sec)
```

Using the --tls option makes the command print out the TLS-specific settings:

```
mcm> show settings --tls;
+---------+------------+----------+
| Section | Key        | Value    |
+---------+------------+----------+
| mcmd    | ssl_ca     |          |
| mcmd    | ssl_cert   |          |
| mcmd    | ssl_cipher |          |
| mcmd    | ssl_key    |          |
| mcmd    | ssl_mode   | DISABLED |
+---------+------------+----------+
5 rows in set (0.00 sec)
```

5.2.11 The stop agents Command

```
stop agents  [[--hosts=host_list] site_name]
```

This command stops one or more MySQL Cluster Manager agents on one or more hosts.

When used without any arguments, stop agents stops the agent to which the client is currently connected.

When used with the name of a management site, the command stops all agents running on hosts making up the site. The following stops all MySQL Cluster Manager agents running on hosts in mysite:

```
mcm> stop agents mysite;
```

You can also stop a subset of the agents in a given management site by listing the hosts where they are running with the --hosts option (short form: -h), along with the name of the site to which they belong. The result of the following command is to stop MySQL Cluster Manager agents running on hosts kolja and torsk, both of which are members of the management site mysite:

```
mcm> stop agents --hosts=kolja,torsk mysite;
```

Multiple host names following the --hosts option should be separated by commas, with no intervening spaces. Invoking stop agents with this option without supplying a *site_name* causes a syntax error. Using an undefined *site_name* or names of hosts not belonging to the site with this command also results in an error.

Note

When IPv6-enabled Windows systems are used as MySQL NDB Cluster hosts under MySQL Cluster Manager, you must reference these hosts using IPv4 addresses. Otherwise, MySQL Cluster Manager will be unable to connect to the agent processes on those hosts. See Section 6.1, “MySQL Cluster Manager Usage and Design Limitations”.

5.2.12 The version Command

```
version
```

This command displays the version of the MySQL Cluster Manager software in use by the MySQL Cluster Manager agent to which this client is connected, as shown here:

```
mcm> version;
+-------------------------------------+
| Version                             |
+-------------------------------------+
| MySQL Cluster Manager 9.6.0 (64bit) |
+-------------------------------------+
1 row in set (0.00 sec)
```

The version command takes no arguments.

5.2.13 The show warnings Command

Using the show warnings command, you can check the warnings (up to the last five) issued to the agent log (mcmd.log). For example:

```
mcm> set delayed_insert_timeout:mysqld=400 mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
```

```
mcm> show warnings;
+---------+------+-----------------------------------------------------------------------------------------------
| Level   | Code | Message
+---------+------+-----------------------------------------------------------------------------------------------
| Warning | 0    | 2021-07-12 21:27:38 Config variable delayed_insert_timeout was deprecated in mysqld 5.6.
| Warning | 1287 | 2021-07-12 21:27:38 '@@delayed_insert_timeout' is deprecated and will be removed in a fu
| Warning | 1287 | 2021-07-12 21:27:38 '@@delayed_insert_timeout' is deprecated and will be removed in a fu
+---------+------+-----------------------------------------------------------------------------------------------
3 rows in set (0.10 sec)
```

5.2.14 The list warnings Command

```
list warnings [--max=n]
```

Using this command, you can check important warnings issued across the whole site during the executions of MySQL Cluster Manager client commands since the agent was started (or restarted). It provides additional information that is not exposed by the show warnings command. Use the --max option to specify the maximum number of warnings to be returned by this command. The default value for the option is 5, and the maximum value is 100.

```
mcm> list warnings --max=4
Timestamp            Host    Message
2022-01-06 06:49:09  torsk   ndb_cluster_connection_pool specified without ndb_cluster_connection_pool_node
2022-01-06 10:17:22  flundra Config variable max_delayed_threads was deprecated in mysqld 5.6.7
2022-01-06 11:19:25  flundra Parameter names use underscore: ndb-use-exact_count rewritten to ndb_use_exact
2022-01-10 12:38:01  torsk   At least one mysqld is not running. Schema dump may be missing
```

5.3 MySQL Cluster Manager Package Commands

This section contains information about MySQL Cluster Manager client commands used to register, extend, unregister, and obtain information about the software packages making up instances of MySQL NDB Cluster that are to be managed using the MySQL Cluster Manager.

5.3.1 The add package Command

```
add package {--basedir=|-b }path
    [{--hosts=|-h }host_list] package_name

host_list:
    host[,host[,...]]
```

This command creates a new package, or, if the package named *package_name* already exists, this command extends the package definition.

The --basedir option (short form: -b) indicates the location of the MySQL NDB Cluster installation directory on the listed hosts, and is required. This must be the path to the top-level directory where the MySQL NDB Cluster software is located (for example, /usr/local/mysql), and should *not* include the MySQL NDB Cluster bin, libexec, or other subdirectory within the installation directory.

Hosts may be specified as a comma-separated list, using the --hosts option (short form: -h); however, this option is not required. If --hosts is omitted, the *path* is assumed to be valid for all hosts in the cluster that is created using this package (see Section 5.4.1, “The create cluster Command”).

Important

- You cannot perform add package if you have not yet defined any sites (each host referenced in an add package command must be associated with a site). See Section 5.2.6, “The create site Command”, for more information about defining sites.
- When a package is first added for a site with the add package command, whenever the --hosts option is used, the host list must contain the host for the mcmd agent to which the mcm client is currently connected, in order to

allow the MySQL Cluster Manager to access the version information of the package.

Suppose we have two Linux hosts named tonfisk and flundra, and the MySQL NDB Cluster software is installed in /usr/local/mysql on both hosts. In this case, you can create a package named mypackage that accounts for both hosts as shown here:

```
mcm> add package --basedir=/usr/local/mysql mypackage;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (0.71 sec)
```

When this package is used to create a cluster, the MySQL Cluster Manager knows that it should find the MySQL NDB Cluster software in the /usr/local/mysql directory on each of the hosts.

For options to MySQL Cluster Manager client command options having Windows paths as values, you must use forward slashes (/) in place of backslashes (\), so if tonfisk and flundra are Windows hosts where MySQL NDB Cluster has been installed to the directory C:\mysql, the corresponding add package command would look like this (with the --basedir option highlighted):

```
mcm> add package --basedir=c:/mysql mypackage;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (0.71 sec)
```

In the example just given, we could also have issued the command as add package --basedir=/usr/local/mysql --hosts=tonfisk,flundra mypackage (or add package --basedir=c:/mysql --hosts=tonfisk,flundra mypackage on Windows) with the same result, but the --hosts option was not required, since the MySQL NDB Cluster software's location is the same on each host. Let us suppose, however, that the software is installed in /usr/local/ndb-host-10 on host tonfisk and in /usr/local/ndb-host-20 on host flundra. In this case, we must issue 2 separate commands, specifying the host as well as the base directory in each case, as shown here:

```
mcm> add package --basedir=/usr/local/ndb-host-10
   >   --hosts=tonfisk yourpackage;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (0.68 sec)

mcm> add package --basedir=/usr/local/ndb-host-20
   >   --hosts=flundra yourpackage;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (0.81 sec)
```

Assuming that both hosts belong to a site called mysite, you can verify that these packages have been created as desired using the list packages command, as shown here:

```
mcm> list packages mysite;
+------------+--------------------------------------+-----------------+
| Package    | Path                                 | Hosts           |
+------------+--------------------------------------+-----------------+
| yourpackage | /usr/local/ndb-host-10               | tonfisk         |
|            | /usr/local/ndb-host-20               | flundra         |
| mypackage  | /usr/local/mysql                     | tonfisk,flundra |
+------------+--------------------------------------+-----------------+
```

```
3 rows in set (1.07 sec)
```

(For more information about this command, see Section 5.3.3, “The list packages Command”.)

It is possible to assign the same base directory (or directories) on the same host (or hosts) to multiple packages, as shown in this example, in which we assume that hosts tonfisk and flundra have previously been assigned to a site named mysite:

```
mcm> add package -b /usr/local/mysql-cluster mypackage;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (1.41 sec)

mcm> add package -b /usr/local/mysql-cluster yourpackage;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (1.58 sec)

mcm> list packages mysite;
+-------------+--------------------------+-----------------+
| Package     | Path                     | Hosts           |
+-------------+--------------------------+-----------------+
| mypackage   | /usr/local/mysql-cluster | tonfisk,flundra |
| yourpackage | /usr/local/mysql-cluster | tonfisk,flundra |
+-------------+--------------------------+-----------------+
2 rows in set (0.50 sec)
```

Note

When IPv6-enabled Windows systems are used as MySQL NDB Cluster hosts under MySQL Cluster Manager, you must reference these hosts using IPv4 addresses. Otherwise, MySQL Cluster Manager will be unable to connect to the agent processes on those hosts. See Section 6.1, “MySQL Cluster Manager Usage and Design Limitations”.

5.3.2 The delete package Command

```
delete package [{--hosts=|-h }host_list] package_name

host_list:
    host[,host[,...]]
```

This command is used to unregister a package. More specifically, it removes any references to MySQL NDB Cluster software installations added to the agent's repository when the package was created. delete package does *not* remove any MySQL NDB Cluster installations; the command removes only references to the installations. Once a package has been unregistered, it can no longer be used for a create cluster command. The MySQL NDB Cluster binaries remain, but cannot be used in a MySQL NDB Cluster administered using the MySQL Cluster Manager unless and until the base directory containing them has been registered with another package. (It is possible to register a base directory with multiple packages; see Section 5.3.1, “The add package Command”, for more information and an example.)

If the --hosts option (short form: -h) is used with this command, the base directory settings for the host or hosts named by the option are removed as well. All hosts given in the *host_list* must be members of the site to which the package is registered. Otherwise, the command fails.

A package that is in use by a cluster cannot be unregistered; the cluster must first be deleted (see Section 5.4.2, “The delete cluster Command”).

Here is an example that demonstrates how to unregister a package named mypackage:

```
mcm> delete package mypackage;
+-----------------------------+
| Command result              |
+-----------------------------+
| Package deleted successfully |
+-----------------------------+
1 row in set (1.23 sec)
```

You can also verify that the package was unregistered using the `list packages` command; the package name should no longer appear in the output of this command. If you attempt to use the unregistered package in a `create cluster` command, the command fails, as shown here:

```
mcm> create cluster --package=mypackage
   > --processhosts=ndb_mgmd@tonfisk,ndbd@grindval,ndbd@flundra,mysqld@tonfisk mycluster;
ERROR 4001 (00MGR): Package mypackage not defined
```

An `upgrade cluster` command that references an unregistered package also fails.

Note

When IPv6-enabled Windows systems are used as MySQL NDB Cluster hosts under MySQL Cluster Manager, you must reference these hosts using IPv4 addresses. Otherwise, MySQL Cluster Manager will be unable to connect to the agent processes on those hosts. See Section 6.1, “MySQL Cluster Manager Usage and Design Limitations”.

5.3.3 The `list packages` Command

```
list packages [package_name] site_name
```

This command lists registered packages. It requires a single argument, that being the name of the site with which the packages are registered, as shown in this example:

```
mcm> list packages mysite;
+-------------+--------------------------------------+-----------------+
| Package     | Path                                 | Hosts           |
+-------------+--------------------------------------+-----------------+
| yourpackage | /usr/local/ndb-host-10               | tonfisk         |
|             | /usr/local/ndb-host-20               | flundra         |
| mypackage   | /usr/local/mysql                     | tonfisk,flundra |
+-------------+--------------------------------------+-----------------+
3 rows in set (1.07 sec)
```

If `tonfisk` and `flundra` are Windows hosts, the list of packages might look something like this:

```
mcm> list packages mysite;
+-------------+--------------------------------------+-----------------+
| Package     | Path                                 | Hosts           |
+-------------+--------------------------------------+-----------------+
| yourpackage | c:/cluster/ndb-host-10               | tonfisk         |
|             | c:/cluster/ndb-host-20               | flundra         |
| mypackage   | c:/mysql                             | tonfisk,flundra |
+-------------+--------------------------------------+-----------------+
3 rows in set (1.07 sec)
```

In the example just shown, `yourpackage` uses the MySQL NDB Cluster binaries installed at `C:\cluster\ndb-host-10` on host `tonfisk`, and at `C:\cluster\ndb-host-20` on `flundra`; `mypackage` uses MySQL NDB Cluster binaries installed at `C:\mysql` on both hosts.

The output contains three columns; these are described in the following list:

- **`Package`.** The name of the package. This can sometimes be empty when a package includes MySQL NDB Cluster installations that are in different locations on different hosts (see next example).
- **`Path`.** The path to the MySQL NDB Cluster installation directory (base directory) on the indicated host or hosts. This is the same as the value given for the `--basedir` option in the `add package` command that was used to create or augment the package.

On Windows, paths shown in this column have any backslash characters converted to forward slashes, just as must be done for the `--basedir` option (see the earlier example in this section).

- **`Hosts`.** The host or hosts where the MySQL NDB Cluster installation or installations are located.

You can filter the results so that information relating to only a single package is displayed by supplying the package name before the site name, as shown here:

```
mcm> list packages yourpackage mysite;
+-------------+--------------------------------------+-----------------+
| Package     | Path                                 | Hosts           |
+-------------+--------------------------------------+-----------------+
| yourpackage | /usr/local/ndb-host-10               | tonfisk         |
|             | /usr/local/ndb-host-20               | flundra         |
+-------------+--------------------------------------+-----------------+
2 rows in set (0.55 sec)
```

(See Section 5.3.1, "The `add package` Command", for the `add package` commands that were used to create `yourpackage`.)

When a package contains MySQL NDB Cluster installations using different base directories on different hosts, each unique combination of path and host is shown in its own row. However, the name of the package is displayed in the first row only; all rows that immediately follow this row and that do not contain the package name also relate to the same package whose name is shown in the first preceding row to display a package name. For example, consider the `list packages` command and output shown here:

```
mcm> list packages mysite;
+-------------+--------------------------------------+---------+
| Package     | Path                                 | Hosts   |
+-------------+--------------------------------------+---------+
| yourpackage | /usr/local/ndb-host-10               | tonfisk |
|             | /usr/local/ndb-host-20               | flundra |
| mypackage   | /usr/local/mysql                     | tonfisk |
|             | /usr/local/bin/mysql                 | flundra |
+-------------+--------------------------------------+---------+
3 rows in set (1.07 sec)
```

This output shows that there are two packages defined for the site named `mysite`; these packages are named `yourpackage` and `mypackage`. The package `yourpackage` consists of the MySQL NDB Cluster binaries in the directory `/usr/local/ndb-host-10` on host `tonfisk`, and in the directory `/usr/local/ndb-host-20` on host `flundra`. The package named `mypackage` consists of the MySQL NDB Cluster binaries in the directory `/usr/local/mysql` on host `tonfisk`, and in the directory `/usr/local/bin/mysql` on host `flundra`.

If you omit the *`site_name`* argument, the command fails with an error, as shown here:

```
mcm> list packages;
ERROR 6 (00MGR): Illegal number of operands
```

5.4 MySQL Cluster Manager Cluster Commands

This section contains descriptions of MySQL Cluster Manager commands used to perform operations on clusters. These include creating and deleting a cluster; starting, stopping, and restarting a cluster; upgrading a cluster (that is, upgrading the MySQL NDB Cluster software used by a given cluster); and listing clusters known to MySQL Cluster Manager.

5.4.1 The `create cluster` Command

```
create cluster {--package=|-P }package_name
    {--processhosts=|-R }process_host_list cluster_name
    [(--import|-m) cluster_name] [--verbose | -v]
```

```
process_host_list:
    process_name[:node_id]@host[,process_name@host[,...]]

process_name:
    {ndb_mgmd|ndbd|ndbmtd|mysqld|ndbapi}
```

This command creates a cluster to be managed by the MySQL Cluster Manager. However, it does not start the cluster (see Section 5.4.7, “The start cluster Command”).

This command can also be used to create a cluster earmarked specifically as a target for importing another cluster that is not already under MySQL Cluster Manager control, as described later in this section, by employing the --import option. See also Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”.

create cluster requires the following arguments:

- A *package_name*, supplied as the value of the --package option (short form: -P). This must be the name of a package previously registered using add package.

- A list (*process_host_list*) of MySQL NDB Cluster processes, the hosts on which they are to run, and—optionally—their node IDs, supplied as the value of the --processhosts option (short form: -R), with list items separated by commas. As with other lists passed as option values in MySQL Cluster Manager commands, you must not use spaces before or after the commas.

  Each item in the *process_host_list* consists of the name of a MySQL NDB Cluster process—possibly suffixed with a colon (:) character followed by the process node ID—joined with the name of the host on which it is located using an amphora (@) sign (also sometimes known as the “at” sign). Permitted values for processes are ndb_mgmd, ndbd, and mysqld. You can also use ndbmtd as process name; in other words, a valid process name is the name of a MySQL NDB Cluster process daemon binary. If node IDs are specified, they must be within the allowed range for the type of node defined.

  To support running your own NDB API applications with a cluster under MySQL Cluster Manager, it is also possible to use ndbapi as a process type. Such applications can be connected to a managed cluster. Currently, MySQL Cluster Manager recognises only that an NDB API application is connected to the cluster; the NDB API application itself must be started, stopped, and configured manually.

  It is also possible to specify one or more “free” mysqld and ndbapi processes without any hosts. To do this, simply use the wildcard * (asterisk character) in place of the hostname or IP address, as shown below:

  - “Free” mysqld process: mysqld@*
  - “Free” ndbapi process: ndbapi@*

  It is also possible to specify a node ID for a “free” process. (If this is not specified, MySQL Cluster Manager assigns a suitable node ID automatically.)

  A mysqld process or ndbapi process that is specified without a host in this fashion is permitted to connect to the cluster from any host that can access the cluster over the network. Otherwise, the process may connect to the cluster only from the specified host.

  By convention, items in the *process_host_list* are listed according to the process type, in the following order:

  1. Management node processes (ndb_mgmd)
  2. Data node processes (ndbd, ndbmtd)

3. SQL node processes (mysqld)
4. Custom NDB API applications (ndbapi)

   For information about writing your own NDB API applications, see The NDB API, in the *MySQL NDB Cluster API Developer Guide*.

While the order in which the items are listed does not affect whether the create cluster command succeeds, we suggest that you follow this convention for readability, as well as compatibility with other MySQL NDB Cluster management tools such as ndb_mgm.

create cluster causes cluster node IDs to be assigned consecutively, in the order that the nodes are specified in the *process_host_list*, with node IDs for data node processes starting with 1, and node IDs for processes other than data node processes starting with 145. You are recommended to follow the best practice of reserving node ID 1 to 144 for data nodes.

Each host referenced in the list must be part of the site for which the package used in create cluster is defined.

For processes of types mysqld and ndbapi, the hostname is required, but not enforced in the running cluster. In other words, an [api] section is created in the cluster config.ini file, but no HostName parameter is specified; thus, the mysqld or ndbapi can connect from any host. (Currently, there is no way using MySQL Cluster Manager to specify that a mysqld or ndbapi process is restricted to connecting from a single host.)

- A name for the cluster. Once the cluster has been created, this name is used to refer to it in other cluster management commands such as delete cluster, start cluster, and stop cluster. Like other object names used with MySQL Cluster Manager, the *cluster_name* must be valid according to the rules given elsewhere in this document for identifiers (see Chapter 5, *MySQL Cluster Manager Client Commands*).

An additional --verbose option for this command causes create cluster to output extra information as it is executed, as shown later in this section.

The --import option flags the cluster as being created as a target for importing a cluster created outside MySQL Cluster Manager. This option causes the cluster's status to appear as import in the output of show status, as shown here:

```
mcm> show status --process newcluster;
+--------+----------+-------+--------+-----------+--------------+
| NodeId | Process  | Host  | Status | Nodegroup | Package      |
+--------+----------+-------+--------+-----------+--------------+
| 1      | ndb_mgmd | alpha | import |           | newpackage   |
| 5      | ndbd     | beta  | import | n/a       | newpackage   |
| 6      | ndbd     | gamma | import | n/a       | newpackage   |
| 10     | mysqld   | delta | import |           | newpackage   |
| 11     | ndbapi   | *     | import |           |              |
+--------+----------+-------+--------+-----------+--------------+
6 rows in set (0.04 sec)
```

Having the import status causes any of the commands start cluster, restart cluster, start process, and stop process to fail if they are executed before an import cluster command has been executed against this cluster. It is also not possible to execute upgrade cluster on a cluster having processes with import status. Other operations on this cluster continue to be performed normally.

Caution

While it is possible to import into a cluster that was created without this option, it is not advisable, since the cluster is not protected against accidentally performing any of the operations listed previously, which may result in confusing

or misleading errors, and possibly other problems. For this reason, it is strongly recommended that you always use the --import option for creating the cluster in such cases.

For more information about importing clusters into MySQL Cluster Manager, including examples, see Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”.

Example

Consider the following command issued in the MySQL Cluster Manager client, which creates a cluster named mycluster:

```
mcm> create cluster --package=mypackage
  -> --processhosts=ndb_mgmd@flundra,ndbd@tonfisk,ndbd@grindval,mysqld@flundra
  -> mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster created successfully |
+-----------------------------+
1 row in set (7.71 sec)
```

As defined by the command just shown, mycluster consists of four nodes: a management node on host flundra; two data nodes—one on each of the hosts tonfisk and grindval; and one SQL node, also on host flundra.

Using the --verbose option causes the command to print output similar to that produced by the list processes command, as shown here:

```
mcm> create cluster --verbose --package=mypackage
  -> --processhosts=ndb_mgmd@flundra,ndbd@tonfisk,ndbd@grindval,mysqld@flundra
  -> mycluster;
+--------+----------+----------+
| NodeId | Name     | Host     |
+--------+----------+----------+
| 49     | ndb_mgmd | flundra  |
| 1      | ndbd     | tonfisk  |
| 2      | ndbd     | grindval |
| 50     | mysqld   | flundra  |
+--------+----------+----------+
4 rows in set (0.32 sec)
```

You can also create this cluster in such a way that the mysqld process is permitted to connect to the cluster from any host able to reach the other cluster hosts over the network as shown here:

```
mcm> create cluster --package=mypackage
  -> --processhosts=ndb_mgmd@flundra,ndbd@tonfisk,ndbd@grindval,mysqld@*
  -> mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster created successfully |
+-----------------------------+
1 row in set (7.71 sec)
```

Note

In the case of a “free” ndbapi process, it is not necessary to have the MySQL Cluster Manager software installed on the host where the ndbapi process is running.

Configuration changes to the newly-created cluster can be made using the set command prior to starting the cluster. This is often preferable to doing after the cluster has been started, since set commands used to make configuration changes in a running cluster can require a rolling restart, and rolling restarts of clusters having many nodes or large quantities of data (or both) may take a great deal of time to complete.

Note

When creating a cluster having more than one mysqld process on the same host machine, MySQL Cluster Manager assigns the MySQL default port (3306) to each of them. Therefore, you must assign a unique port for each mysqld process in the cluster.

5.4.2 The delete cluster Command

```
delete cluster [--removedirs] cluster_name
```

This command deletes the cluster named *cluster_name*, removing it from the list of clusters managed by MySQL Cluster Manager.

delete cluster does *not* remove any MySQL NDB Cluster binaries from hosts. However, it *does* remove the cluster configuration, data, and log files that reside in the MySQL Cluster Manager data repository.

This example demonstrates how to delete a cluster named mycluster:

```
mcm> delete cluster mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster deleted successfully |
+-----------------------------+
1 row in set (1.22 sec)
```

A look at the MySQL Cluster Manager data repository (at /opt/mcm_data/ in this case) shows that the folder that used to host the configuration, data, and log files for mycluster (/opt/mcm_data/clusters/mycluster) no longer exists:

```
$> ls -l /opt/mcm_data/clusters
total 0
```

To remove the configuration and data files outside of the MySQL Cluster Manager data repository, delete cluster must be invoked with the --removedirs option, like this:

```
mcm> delete cluster --removedirs mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster deleted successfully |
+-----------------------------+
1 row in set (1.22 sec)
```

For example, if one of the data node on mycluster has its data directory outside of the MySQL Cluster Manager data repository:

```
mcm> get Datadir mycluster;
+---------+--------------------------+----------+---------+----------+---------+---------+---------+
| Name    | Value                    | Process1 | NodeId1 | Process2 | NodeId2 | Level   | Comment |
+---------+--------------------------+----------+---------+----------+---------+---------+---------+
| DataDir | /home/dso/mycluster/cdata | ndbd     | 1       |          |         | Process |         |
...
```

Deleting mycluster without using --removedirs does *not* remove the data directory for node 1:

```
$> ls -l /home/dso/mycluster
total 4 drwxr-xr-x. 3 dso dso 4096 Sep 10 18:00 cdata
```

However, if the --removedirs option is used, the data directory for node 1 also gets removed:

```
$> ls -l /home/dso/mycluster
total 0
```

delete cluster fails if the cluster to be deleted is running, as shown here:

```
mcm> delete cluster mycluster;
ERROR 5010 (00MGR): All processes must be stopped to delete cluster mycluster
```

You must shut down the cluster first, using `stop cluster`.

The `delete cluster` command also fails if a backup of the cluster exists somewhere under */path-to-mcm-data-repository/clusters/clustername* (the default arrangement). The backup should be moved to another other location or be deleted first before the `delete cluster` command is executed. This is to prevent an unexpected loss of the cluster backup.

5.4.3 The `list clusters` Command

```
list clusters site_name
```

This command lists all clusters defined for a given management site named *`site_name`*, together with the package used by each cluster. For example, the command shown here displays a list of all clusters defined for the site named `mysite`:

```
mcm> list clusters mysite;
+------------------+----------+
| Cluster          | Package  |
+------------------+----------+
| mycluster        | m-7.1.26 |
| yourcluster      | y-7.1.26 |
| someothercluster | s-7.2.9  |
+------------------+----------+
3 rows in set (2.07 sec)
```

If *`site_name`* is omitted, the command fails with an error, as shown here:

```
mcm> list clusters;
ERROR 6 (00MGR): Illegal number of operands
```

5.4.4 The `list nextnodeids` Command

```
list nextnodeids cluster_name
```

MySQL Cluster Manager normally assigns IDs to new node processes automatically (although this can be overridden when issuing the `create cluster` or `add process` command). The `list nextnodeids` command can be used to see the next node ID that MySQL Cluster Manager has reserved for the next new process (of each possible process type) to be added to the cluster named *`cluster_name`*, which is required.

```
mcm> list nextnodeids mycluster;
+-----------+--------------+-------------+--------------------------+
| Category  | NodeId Range | Next NodeId | Processes                |
+-----------+--------------+-------------+--------------------------+
| Datanodes | 1  - 144     | 3           | ndbd, ndbmtd             |
| Others    | 145 - 255    | 149         | ndb_mgmd, mysqld, ndbapi |
+-----------+--------------+-------------+--------------------------+
2 rows in set (0.07 sec)
```

5.4.5 The `restart cluster` Command

```
restart cluster [--sequential-restart]  cluster_name
```

This command performs a rolling restart (see Performing a Rolling Restart of an NDB Cluster) of the cluster named *`cluster_name`*. The cluster must already be running in order for this command to succeed. (For information about how to determine the operation state of the cluster, see Section 5.4.6, “The `show status` Command”.)

For example, the command shown here performs a rolling restart of the cluster named `mycluster`:

```
mcm> restart cluster mycluster;
+--------------------------------+
```

```
| Command result                 |
+--------------------------------+
| Cluster restarted successfully |
+--------------------------------+
1 row in set (1 min 22.53 sec)
```

If the cluster is not already running, `restart cluster` fails with an error, as shown here:

```
mcm> show status --cluster mycluster;
+-----------+---------+---------+
| Cluster   | Status  | Comment |
+-----------+---------+---------+
| mycluster | stopped |         |
+-----------+---------+---------+
1 row in set (1.49 sec)

mcm> restart cluster mycluster;
ERROR 5009 (00MGR): Restart can not be performed as processes are
stopped in cluster mycluster
```

By default, a rolling restart is performed on the nodes in a *parallel* manner (that is, half of the nodes are stopped and restarted together, followed by the second half of the nodes). In some situations, you might want to have a rolling restart performed in a *sequential* manner for the data nodes by adding the `--sequential-restart` option, in which case the data nodes ares stopped and restarted one after another.

Note

Depending on the number of nodes and the amount of data stored in the cluster, a rolling restart can take a considerable amount of time, up to several hours for a cluster with a great many data nodes and a large amount of data.

Currently, there is no mechanism in MySQL Cluster Manager for performing system initial restarts of a cluster. This means that attributes that require an initial restart to be changed must be set before the cluster is started for the first time.

5.4.6 The `show status` Command

```
show status
show status --cluster|-c cluster_name
show status --operation|-o cluster_name
show status --backup|-b cluster_name
show status --process|-r cluster_name
show status --progress cluster_name
show status --progressbar cluster_name
```

This command is used to check the status of clusters, cluster processes, backups, and commands issued in the MySQL Cluster Manager client. The type of status returned depends on whether an option is used with the command and, if so, which of the four options of `--cluster` (short form: `-c`), `--operation` (short form: `-o`), `--backup` (short form: `-b`), or `--process` (short form `-r`) is used.

When no option is used, `show status` reports runtime information from the `mcmd` to which the client is connected. For example:

```
mcm> show status;
+--------------+-----------------------------------------------------------------------------------+
| Property     | Value                                                                             |
+--------------+-----------------------------------------------------------------------------------+
| agent_number | 0                                                                                 |
| cwd          | /opt/mcm9.6.0/                                                                    |
| connections  | 1                                                                                 |
| max_msg_id   | 105                                                                               |
| max_synode   | {4c0f56d8 105 0}                                                                  |
| ssl_cipher   |                                                                                   |
| tls_version  |                                                                                   |
| uptime       | 90                                                                                |
| version      | 9.6.0                                                                             |
```

```
+-------------+-------------------------------------------------------------------------------------+
9 rows in set (0.00 sec)
```

--cluster option

When this option is used, show status reports on the status of the cluster named *cluster_name*, as shown in this example:

```
mcm> show status --cluster mycluster;
+-----------+-------------------+---------+
| Cluster   | Status            | Comment |
+-----------+-------------------+---------+
| mycluster | fully operational |         |
+-----------+-------------------+---------+
1 row in set (0.01 sec)
```

When used with the --cluster option (short form: -c), the output of this command consist of two columns. The Cluster column contains the name of the cluster. The Status column contains a description of the cluster's status; possible values and their meanings are shown in the following table:

Table 5.1 Status values shown by show status --cluster

Status Value	Meaning
fully operational	All cluster processes are running.
operational	All node groups are up and running, but at least one data node process (ndbd or ndbmtd) is not running. The cluster is online, but you should determine why any “missing” data nodes are not running and correct the problem as soon as possible.
non-operational	The cluster is not operational, because at least one node group is offline. You must investigate and fix the problem or problems, then restart the cluster, before the cluster can be used for data storage and retrieval operations.
failed	All processes in the cluster have exited, but, unlike stopped, some nodes did not exit cleanly.
stopped	The cluster is not running, because it has been stopped by the user. This normally does not indicate any problem as such, but you must restart the cluster before it can be used by any applications.
created	The cluster has been created successfully using the create cluster command, but has never been started. You must start the cluster using the start cluster command before you can make use of it.
unknown	The MySQL Cluster Manager was unable to determine the cluster's status. This may or may not indicate a problem with the cluster; it is possible that the problem lies with one or more MySQL Cluster Manager agents or the MySQL Cluster Manager client. You should attempt to determine the status of the cluster by other means, such as using show status --process in the MySQL Cluster Manager client (described later in this section), or employing one of the commands available in the ndb_mgm client (see

Status Value	Meaning
	ndb_mgm — The NDB Cluster Management Client) such as SHOW or ALL STATUS.

--operation option

When the --operation option (short form: -o) is used, it causes SHOW STATUS to display the status of the latest command to be executed. An example of this command is shown here:

```
mcm> show status --operation mycluster;
+--------------+----------+-----------------------+
| Command      | Status   | Description           |
+--------------+----------+-----------------------+
| start cluster | finished | Completed successfully |
+--------------+----------+-----------------------+
1 row in set (0.10 sec)
```

The output contains 3 columns, described in the following list:

- **Command.** The text of the command last issued (previous to the show status --operation command), less any options or arguments.
- **Status.** The current state of the command. Possible values and their meanings are listed later in this section.
- **Description.** Depending on the command and its status, this column may contain additional information. Otherwise, No information available is displayed here.

Possible values for the Status column, together with descriptions of these values, are shown in the following table:

Table 5.2 Status values shown by show status --operation

Status Value	Description
executing	MySQL Cluster Manager is executing the command, but has not yet completed doing so.
finished	The command has executed (and completed) successfully.
failed	The command failed to execute. The Description column may contain information about the reason for the failure.
unknown	MySQL Cluster Manager was unable to determine the status of this command.

--backup option

When this option is used, show status reports on the status of the backup process for the cluster named *cluster_name*, as shown in the following examples:

```
mcm> show status --backup mycluster;
+----------------------------------------+
| Command result                         |
+----------------------------------------+
| No backup currently active in mycluster |
+----------------------------------------+
1 row in set (0.05 sec)
```

```
mcm> show status --backup mycluster;
+----------------------------------------+
| Command result                         |
+----------------------------------------+
| BackupId 5 currently active in mycluster|
```

```
+----------------------------------------+
1 row in set (0.09 sec)
```

--process option

When run with this option, show status returns information about each process in the cluster named *cluster_name*, as shown in this example:

```
mcm> show status --process mycluster;
+------+----------+----------+---------+-----------+
| Id   | Process  | Host     | Status  | Nodegroup |
+------+----------+----------+---------+-----------+
| 1    | ndb_mgmd | tonfisk  | running |           |
| 2    | ndbd     | flundra  | running | 0         |
| 3    | ndbd     | grindval | running | 0         |
| 4    | mysqld   | lax      | running |           |
+------+----------+----------+---------+-----------+
4 rows in set (1.67 sec)
```

When the --process option (short form: -r) is used with show status, the output contains 5 columns, described in the following list:

- **Id.** This is the node ID of the process as a node in cluster *cluster_name*.
- **Process.** The type of process, that is, the name of the corresponding MySQL NDB Cluster executable. Allowed values are ndb_mgmd, ndbd, ndbmtd, and mysqld.
- **Host.** The hostname or IP address of the computer where the process is running.
- **Status.** The state or condition of this process. Possible values for this column are given later in this section.
- **Nodegroup.** If the Process is ndbd or ndbmtd—that is, if the process is a data node process—then this column shows the ID of the node group to which the process belongs. For any other value of Process, this column is empty.

Possible values for the Status column are shown in the following table, together with a description of what this value represents:

Table 5.3 Status values shown by show status --process

Status Value	Meaning
running	The process is running normally.
stopped	The process has been stopped by the user.
added	The process has been added to the cluster, but not yet started.
connected	The ndbapi or mysqld process is connected to the cluster.
starting	The process has been started, but is not yet fully running. (For data nodes, you can determine which start phase the node is currently in by using the status command in the ndb_mgm client.)
stopping	The process has received a command to stop, and is now shutting down.
failed	The process has shut down unexpectedly (likely to have crashed). You should determine the cause for this unplanned shutdown, fix the problem, and restart the process as soon as possible.
import	The process is part of a cluster that was created for import, but the actual migration of processes

Status Value	Meaning
	and data from the original cluster has not yet taken place. start process and stop process commands fail for this process until this migration has occurred.
unknown	MySQL Cluster Manager is unable to establish the current status of this process. You should try to determine its status using other means.

--progress option

When run with this option, show status returns, when available, progress on the current action of mcmd on the cluster named *cluster_name*, in terms of the percentage of the total number of steps completed:

```
mcm> show status --progress mycluster;
+-----------------+-----------+----------+
| Command         | Status    | Progress |
+-----------------+-----------+----------+
| restore cluster | executing | 47%      |
+-----------------+-----------+----------+
1 row in set (0.02 sec)
```

--progressbar option

The option provides the same function as the --progress option, but also adds an ASCII-art progress bar:

```
mcm> show status --progressbar mycluster;
+-----------------+-----------+-----------------------------+
| Command         | Status    | Progress                    |
+-----------------+-----------+-----------------------------+
| restore cluster | executing | 47% [#########           ]  |
+-----------------+-----------+-----------------------------+
1 row in set (0.02 sec)
```

You must supply the name of an existing cluster with this command, or else show status fails with an error, as shown here:

```
mcm> show status;
ERROR 6 (00MGR): Illegal number of operands

mcm> show status -c nosuchcluster;
ERROR 5001 (00MGR): Cluster nosuchcluster not defined
```

Important

Do not confuse this command with the MySQL SHOW STATUS statement, which has a different syntax and can be used only in the standard mysql client. The MySQL Cluster Manager client command accepts only those options shown at the beginning of this section, and does not accept a LIKE or WHERE clause.

5.4.7 The start cluster Command

```
start cluster [--initial|-i] [--skip-init=process_id_list] cluster_name
```

This command starts the cluster named *cluster_name*, as shown in this example:

```
mcm> start cluster mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster started successfully |
+-----------------------------+
1 row in set (45.37 sec)
```

In order for the command to succeed, the cluster named in the command must already exist; otherwise the command fails with the error Cluster *cluster_name* not defined, as shown here:

```
mcm> list sites;
+--------+------+-------+----------------------------+
| Site   | Port | Local | Hosts                      |
+--------+------+-------+----------------------------+
| mysite | 1862 | Local | tonfisk,flundra,grindval,haj |
+--------+------+-------+----------------------------+
1 row in set (1.72 sec)

mcm> list clusters mysite;
+-----------+-----------+
| Cluster   | Package   |
+-----------+-----------+
| mycluster | mypackage |
+-----------+-----------+
1 row in set (1.70 sec)

mcm> start cluster yourcluster;
ERROR 5001 (00MGR): Cluster yourcluster not defined
```

In addition, the cluster must not already be running, as shown here:

```
mcm> show status --cluster mycluster;
+-----------+-------------------+---------+
| Cluster   | Status            | Comment |
+-----------+-------------------+---------+
| mycluster | fully operational |         |
+-----------+-------------------+---------+
1 row in set (0.01 sec)

mcm> start cluster mycluster;
ERROR 5005 (00MGR): Cluster mycluster is running
```

A cluster created for import cannot be started until the import has been completed. See Section 5.4.1, “The create cluster Command”, and Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”, for more information.

--initial option

The --initial option (short form: -i) causes the following to happen:

- All cluster data node are started as if start process --initial had been used on them, which means that all data nodes wipe their data and start with clean data node file systems. NDB tables that were previously stored in the cluster are lost.
- All cluster SQL nodes are started as if start process --initial have been used on them, which means MySQL Cluster Manager rebuilds the mysqld data directory with the mysqld --initialize-insecure command . However, the node's data directory must be empty, or the reinitialization will not be attempted.

  To skip reinitialization for any SQL nodes, list their process IDs (separated by commas if there are more than one) using the --skip-init=*process_id_list* option, for example:

  ```
  mcm> start cluster --initial --skip-init=50,51 mycluster;
  ```

  The --skip-init option only accepts SQL node IDs as its argument; it cannot be used to skip the initialization of data nodes.

Under normal circumstances, you should use this option to start a cluster only when either you do not wish to preserve any of its data (and want to make a clean start), or you intend to restore the cluster from backup to a known good state (see Section 5.8.5, “The restore cluster Command”). You should also be aware that no special warnings are printed by the mcm client when --initial is used with start cluster; the command is immediately executed.

For information about creating cluster backups, see Section 5.8.2, “The backup cluster Command”. If you need to know which backups are available (if any), use list backups.

5.4.8 The stop cluster Command

```
stop cluster cluster_name
```

This command stops the cluster named *cluster_name*, if it is running, as shown in this example:

```
mcm> stop cluster mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Cluster stopped successfully |
+------------------------------+
1 row in set (21.31 sec)
```

stop cluster fails if the cluster is not in an operational state (see Section 5.4.6, “The show status Command”, for information about obtaining the cluster's status):

```
mcm> show status --cluster mycluster;
+-----------+---------+---------+
| Cluster   | Status  | Comment |
+-----------+---------+---------+
| mycluster | stopped |         |
+-----------+---------+---------+
1 row in set (0.01 sec)

mcm> stop cluster mycluster;
ERROR 5006 (00MGR): Cluster mycluster is stopped
```

stop cluster cannot be used on a cluster created for import until the import has been completed. See Section 5.4.1, “The create cluster Command”, and Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”, for more information.

5.4.9 The autotune Command

```
autotune [--dryrun] [--sequential-restart] [--writeload=writeload] template cluster_name

writeload:
    {low|medium|high}

template:
    {web|realtime|test}
```

The command autotune a number of parameters for the cluster according to the specified values for the template argument and the [optional] writeload option, in order to optimize the cluster's performance.

The valid values for template are:

- web: Maximize performance for the given hardware.
- realtime: Maximize performance while maximizing sensitivity to timeouts in order to minimize the time needed to detect failed cluster processes.
- test: Minimal resource usage for small-scale testing. Not intended for production environments.

The valid values for --writeload are:

- low: The expected load includes fewer than 100 write transactions for second.
- medium: The expected load includes 100 to 1000 write transactions per second. This is the default value used when --writeload is not specified.

- high: The expected load includes more than 1000 write transactions per second.

The cluster must be in the created or fully operational status for this command to work, or an error will result. The command tunes the cluster by issuing a number of set commands to adjust different parameters, and then performs a rolling restart for the cluster. Use the --sequential-restart option to make the rolling restart a sequential one.

When the --dryrun option is used, the command does not make any actual changes to the cluster, but writes the set commands that it will issue for tuning into the file **/path-to-mcm-data-repository**/clusters/**clustername**/tmp/autotune.**message_id**.mcm.

```
mcm> autotune --dryrun --writeload=high realtime mycluster;
+-------------------------------------------------------------------------------------------------------------
| Command result
+-------------------------------------------------------------------------------------------------------------
| Autotuning calculation complete. Please check /opt/mcm_data/clusters/mycluster/tmp/autotune.30fcce24_
+-------------------------------------------------------------------------------------------------------------
1 row in set (0.62 sec)
```

```
$> cat /opt/mcm_data/clusters/mycluster/tmp/autotune.30fcce24_2184_0.mcm
# The following will be applied to the current cluster config:
set HeartbeatIntervalDbDb:ndbmtd=1500 mycluster;
set HeartbeatIntervalDbApi:ndbmtd=1500 mycluster;
set RedoBuffer:ndbmtd=64M mycluster;
set SharedGlobalMemory:ndbmtd=20M mycluster;
set DataMemory:ndbmtd=83886080 mycluster;
set IndexMemory:ndbmtd=18874368 mycluster;
set MaxNoOfExecutionThreads:ndbmtd=2 mycluster;
set FragmentLogFileSize:ndbmtd=256M mycluster;
set NoOfFragmentLogFiles:ndbmtd=3 mycluster;
```

After checking out those changes in the .mcm file, if you do not want to apply all of them to your cluster, you can edit the .mcm file as desired, and then execute it at the mcm client (see Section 4.5.2.3, “Creating and Configuring the Target Cluster” for how to do that). If you are happy with all the changes described in the file, issue the autotune command again without the --dryrun option, to perform the tuning:

```
mcm> autotune --writeload=high realtime mycluster;
+-----------------------------------------------------+
| Command result                                      |
+-----------------------------------------------------+
| Cluster successfully autotuned to template realtime |
+-----------------------------------------------------+
1 row in set (2 min 58.09 sec)
```

5.4.10 The upgrade cluster Command

```
upgrade cluster {--package=|-P }package_name
 [{--nodeid|-n }node_id_list] [--force|-f]
 [--retry|-L] [--set=attribute_assignment_list] cluster_name

node_id_list:
    node_id[, node_id[, ...]]

attribute_assignment_list:
    attribute_assignment[,attribute_assignment][,...]

attribute_assignment:
    attribute_name:process_name[=value]
```

This command upgrades the cluster named *cluster_name* to the software package *package_name* specified with the --package. It finishes an upgrade by performing a rolling restart for the cluster, in which data nodes are restarted with the --initial option to have their data file systems rebuilt.

The new package must be registered using add package before you can use it for an upgrade; otherwise, upgrade cluster fails with an error.

To use the command to perform an upgrade on a cluster, unless some special options are used (see explanations on the --force, --retry, and --nodeid options below), the cluster must be in the fully operational status (you can check that using the command show status --cluster *cluster_name*). A cluster created for import cannot be upgraded until the import has been completed. See Section 5.4.1, "The create cluster Command", and Section 4.5, "Importing MySQL NDB Clusters into MySQL Cluster Manager", for more information.

Suppose mycluster is using MySQL NDB Cluster 8.0.34, and the binaries are registered with a package named 8.0.34, as shown by this list clusters command:

```
mcm> list clusters mysite;
+-----------+---------+
| Cluster   | Package |
+-----------+---------+
| mycluster | 8.0.34  |
+-----------+---------+
1 row in set (1.80 sec)
```

Now you wish to upgrade mycluster to MySQL NDB Cluster. Assuming that you have placed the NDB 9.6.0 binaries in the same directory on each host, the add package command to create a new package named 9.6.0 that contains these binaries might look something like this:

```
mcm> add package --basedir=/usr/local/ndb-9.6.0 9.6.0;
+---------------------------+
| Command result            |
+---------------------------+
| Package added successfully |
+---------------------------+
1 row in set (0.88 sec)
```

Note

On Windows, you must replace any backslash (\) characters in the path used for the add package command's --basedir option with forward slashes (/). See Section 5.3.1, "The add package Command", for additional information and examples.

Both packages should now be listed in the output of the command list packages mysite. To perform the upgrade to the 9.6.0 package, use the upgrade cluster command as shown here:

```
mcm> upgrade cluster --package=9.6.0 mycluster;
+-------------------------------------+
| Command result                      |
+-------------------------------------+
| Cluster version changed successfully |
+-------------------------------------+
1 row in set (3 min 17.00 sec)
```

Once the upgrade cluster command has been successfully executed, you can verify that mycluster is now using the 9.6.0 package from the output of the appropriate list clusters command:

```
mcm> list clusters mysite;
+-----------+---------+
| Cluster   | Package |
+-----------+---------+
| mycluster | 9.6.0   |
+-----------+---------+
1 row in set (1.80 sec)
```

The command can perform major as well as minor series upgrades. Despite the name of this command, upgrade cluster can also be used to perform MySQL NDB Cluster downgrades.

Not all upgrades and downgrades between different versions of MySQL NDB Cluster are supported by the command. These criteria must be met:

- The upgrade or downgrade must be supported by the MySQL NDB Cluster versions involved. See Upgrading and Downgrading NDB Cluster for lists of allowed upgrades and downgrades.
- Both the versions you upgrade or downgrade to and from must be supported by the version of MySQL Cluster Manager you are using.

When using the upgrade cluster command, you can use the --set option to reconfigure your MySQL NDB Cluster at the same time. This is particularly helpful when the upgrade requires configuration changes to your cluster. This option takes as its argument an attribute assignment list similar in format to that used with the get and set commands; see description of the set command on the proper way to formulate an attribute assignment list. For example: if you want to change the memory assigned to each data node for storing database records to 750M, specify that with the --set option in your upgrade cluster command:

```
mcm> upgrade cluster --package=9.6.0 --set=DataMemory:ndbd=750M  mycluster;
+-------------------------------------+
| Command result                      |
+-------------------------------------+
| Cluster version changed successfully |
+-------------------------------------+
1 row in set (3 min 17.04 sec)
```

Note

Unlike the way you use the set command, an equal sign (=) immediately following the --set option is required.

Options for dealing with failed upgrades

The --force option (-f for short) should be used when you want to run the upgrade cluster command again after a failed upgrade attempt that ends up with any failed management or data nodes. Without the --force option, the upgrade cluster command only runs when the cluster is in the fully operational status.

The --retry option (-L for short) should be used when you want to retry the upgrade cluster command after a failed attempt that ends up with some nodes being upgraded, and some not. Without the --retry option, the upgrade cluster command cannot be run on the same cluster twice using the same package.

In the case of a failed or incomplete upgrade, instead of using the --force and --retry option, you can also choose to retry the upgrade only on the failed nodes by specifying them using the --nodeid option (-n in short). Check for any failed nodes after a failed upgrade:

```
mcm> upgrade cluster -P next mycluster;
ERROR 7006 (00MGR): Process error: <reason of failure>
mcm> show status --process mycluster;
+--------+----------+----------+---------+-----------+-----------+
| NodeId | Process  | Host     | Status  | Nodegroup | Package   |
+--------+----------+----------+---------+-----------+-----------+
| 49     | ndb_mgmd | thinkpad | running |           | next      |
| 1      | ndbmtd   | thinkpad | running | 0         | next      |
| 2      | ndbmtd   | thinkpad | running | 0         | next      |
| 50     | mysqld   | thinkpad | running |           | next      |
| 51     | mysqld   | thinkpad | failed  |           | next      |
| 52     | ndbapi   | *        | added   |           |           |
+--------+----------+----------+---------+-----------+-----------+
6 rows in set (0.03 sec)
```

Then, issue the command again, specifying the failed node with the --nodeid option:

```
mcm> upgrade cluster --nodeid=51 -P next mycluster;
+-------------------------------------+
| Command result                      |
```

```
+-------------------------------------+
| Cluster version changed successfully |
+-------------------------------------+
1 row in set (26.03 sec)
```

Warning

Using the `--nodeid` option inappropriately with the `upgrade cluster` command may result in a partial upgrade. Use it only when a previous attempt to upgrade failed, and only with guidance from the proper support personnel.

5.5 MySQL Cluster Manager Configuration Commands

This section covers the commands used in the MySQL Cluster Manager for getting and setting values of various types used in MySQL NDB Cluster configuration. We begin with a discussion of what we mean by the term “configuration attribute”, and how this relates to the manual configuration of a MySQL NDB Cluster using MySQL NDB Cluster configuration parameters and MySQL Server options and variables that you may already be familiar with.

Configuration attributes.
Traditionally, when administering MySQL NDB Cluster, it has been necessary to distinguish between 3 types of configuration data:

- *Configuration parameters* set in the MySQL NDB Cluster global configuration file read by the management server (or servers), by convention named `config.ini`
- *Configuration variables* set in a running MySQL server (SQL node) by using the SQL `SET` statement in the `mysql` command-line client (or in another MySQL client application)
- *Configuration options* passed to MySQL NDB Cluster executable programs when invoking them

Note

Configuration options passed to `mysqld` often have the effect of setting values for configuration variables, many—but not all—of which can be overridden in a running MySQL server using a MySQL client application such as `mysql`.

MySQL Cluster Manager simplifies this configuration scheme by treating all 3 types of configuration data as *attributes*, where the term “attribute” refers to a MySQL NDB Cluster configuration parameter, a MySQL Server variable, or a command-line option used with one or more MySQL NDB Cluster binary programs. It does this transparently, handling all necessary changes in a unified interface.

Suppose that you wish to know how much data memory is allocated to the data nodes in a given MySQL NDB Cluster. Rather than having to determine that this is controlled using the `DataMemory` configuration parameter that is written in the `config.ini` file and then reading that file to find the value, you merely invoke the MySQL Cluster Manager `get` command, and MySQL Cluster Manager handles reading from the file for you, and displays the value without the necessity of opening the file in a separate application such as `more` or `less`. If you wish to change the amount of data memory allocated to the data nodes, you can issue a MySQL Cluster Manager `set` (or `reset`) command; MySQL Cluster Manager then writes the desired value to `config.ini`. If—as is the case with `DataMemory`—updating a configuration value in a running MySQL NDB Cluster requires a rolling restart to be performed, MySQL Cluster Manager can perform this operation automatically so that the configuration change takes effect without further intervention required on the part of the operator.

Configuration attribute levels.
A configuration attribute value applies at one of the three levels, described here:

- *Default*: This value is always used by any MySQL NDB Cluster process of the type or types (such as `ndbd` or `mysqld`) to which the attribute applies, unless this value is overridden by the user.

- *Process*: This value is used for all instances of a given type of MySQL NDB Cluster process.
- *Instance*: This value is used for a specific instance of a MySQL NDB Cluster process, the instance being identified by its MySQL NDB Cluster node ID.

Default values are hard-coded into MySQL NDB Cluster; you can override a default value for a given configuration attribute (using the `set` command) or reset a given attribute value to its default (using the `reset` command), but you cannot change a default value itself. You can set or reset an configuration attribute's value on either the process level or the instance level using a single `set` or `reset` command. Once you have set or reset the value of a configuration attribute, this value persists until it is changed by executing another `set` or `reset` command.

Note

When setting or resetting a configuration attribute value, you must specify the level at which the setting applies.

MySQL Cluster Manager determines what value to use for a configuration attribute relating to a given process by following these steps for each MySQL NDB Cluster process:

(For each configuration attribute:)

1. Is an attribute value defined for the node ID of this process?

   Yes: Use the value that was defined for this node ID, and exit.

   No: Proceed to the next step.

2. Is an attribute value specified on the process level, that is, for all processes of this type?

   Yes: Use the value that was specified for all processes of this type, and exit.

   No: Use the default value that applies to processes of this type, and exit.

(In the steps just shown, “exit” can be taken to mean “If there are more configuration attributes applicable to this process that have not yet been set, proceed to the next attribute until there are no more attributes to be set for this process”.)

Note

The most recently specified value takes precedence. This means that if you set a configuration attribute for a specific process, then later specify a process-level value for this attribute, the process-level value is used for all processes of that type, including the instance for which you earlier set an instance-specific value.

Mandatory attributes.
Some attributes must be defined in the MySQL Cluster Manager at the process type or instance level for all processes of the applicable type or types for the cluster configuration to be valid. Such *mandatory attributes* may be changed, but not reset; in other words, the definition can be changed, but the definition itself cannot be removed entirely. Another way of stating this is that a mandatory attribute has no default value.

An example of a mandatory attribute is `NodeId`. If you try to reset a mandatory attribute, the attempt fails with an error, as shown here:

```
mcm> reset NodeId:ndb_mgmd:1 mycluster;
ERROR 6007 (00MGR): Config attribute NodeId is mandatory and cannot be reset
mcm> reset NodeId:ndbd:2 mycluster;
ERROR 6007 (00MGR): Config attribute NodeId is mandatory and cannot be reset
mcm> reset NodeId:mysqld:4 mycluster;
ERROR 6007 (00MGR): Config attribute NodeId is mandatory and cannot be reset
```

Read-only attributes.
A *read-only attribute* is an attribute that must be defined by the MySQL Cluster Manager when a cluster is created. A read-only attribute can be neither changed nor reset by the user. This means that a read-only attribute is always a mandatory attribute.

One such attribute is `HostName`, which is read only for any type of MySQL NDB Cluster process. Any attempt to change or reset a read-only attribute fails, as shown here:

```
mcm> reset HostName:ndb_mgmd mycluster;
ERROR 6008 (00MGR): Config attribute HostName is readonly and cannot be changed
mcm> reset HostName:ndbd mycluster;
ERROR 6008 (00MGR): Config attribute HostName is readonly and cannot be changed
mcm> reset HostName:mysqld mycluster;
ERROR 6008 (00MGR): Config attribute HostName is readonly and cannot be changed

mcm> set HostName:ndb_mgmd mycluster;
ERROR 6008 (00MGR): Config attribute HostName is readonly and cannot be changed
mcm> set HostName:ndbd mycluster;
ERROR 6008 (00MGR): Config attribute HostName is readonly and cannot be changed
mcm> set HostName:mysqld mycluster;
ERROR 6008 (00MGR): Config attribute HostName is readonly and cannot be changed
```

An attribute that is mandatory or read only is set when a cluster is created. Neither a mandatory attribute nor a read-only attribute can be reset. (Neither type of attribute has a default value other than what is set for it when the cluster is created.) A mandatory attribute can be changed at any time by the user; a read-only attribute cannot be changed once the cluster has been created. You can obtain a listing of mandatory and read-only attributes using the `get` command.

A listing of attribute properties also can be found in the output of `ndb_config --configinfo --xml` (see ndb_config — Extract NDB Cluster Configuration Information); for more complete information, see Configuration of NDB Cluster.

MySQL Cluster Manager determines internally which attributes are considered read-only for reasons of cluster stability and performance. You can use the `get` command to see which attributes are read only.

Command-line-only attributes.
Command-line-only attributes are attributes that, when outside of MySQL Cluster Manager, must be specified as command-line options instead of parameters in a configuration file (for example, `config.ini` or `my.cnf`). These include all the command-line options of the `ndb_mgmd`, `ndbd`, and `ndbmtd` nodes, as well as `mysqld` options listed in Server Option, System Variable, and Status Variable Reference as not valid in option files. A small number of these command-line-only attributes can, however, be configured with MySQL Cluster Manager using the `set` and `reset` commands, and their values can be checked with the `get` command; they include:

- For `ndb_mgmd`: `--log-name`, `--verbose`
- For `ndbd` and `ndbmtd`: `--core-file`, `--verbose`
- The `mysqld --core-file` option.

These command-line-only attributes supported by the `get`, `set`, and `reset` commands are marked with `Command Line` in the `Comment` column of the `get` command's output:

```
mcm> set log-name:ndb_mgmd=Mgm145  mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (27.37 sec)

mcm> get -d log-name:ndb_mgmd mycluster;
+----------+--------+----------+---------+----------+---------+---------+--------------+
| Name     | Value  | Process1 | NodeId1 | Process2 | NodeId2 | Level   | Comment      |
+----------+--------+----------+---------+----------+---------+---------+--------------+
```

```
| log-name | Mgm145 | ndb_mgmd | 145     |          |         | Process | Command Line |
+----------+--------+----------+---------+----------+---------+---------+--------------+
1 row in set (0.10 sec)

mcm> reset log-name:ndb_mgmd mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (5.85 sec)

mcm> get -d log-name:ndb_mgmd mycluster;
+----------+----------+----------+---------+----------+---------+---------+--------------+
| Name     | Value    | Process1 | NodeId1 | Process2 | NodeId2 | Level   | Comment      |
+----------+----------+----------+---------+----------+---------+---------+--------------+
| log-name | MgmtSrvr | ndb_mgmd | 145     |          |         | Default | Command Line |
+----------+----------+----------+---------+----------+---------+---------+--------------+
1 row in set (0.10 sec)
```

5.5.1 The get Command

```
get [--include-defaults|-d] [--all] [filter_specification_list] cluster_name

filter_specification_list:
    filter_specification[,filter_specification][,...]

filter_specification:
    [attribute_name][:process_specification][+process_specification]]

process_specification:
    [process_name][:process_id]

process_name:
    {ndb_mgmd|ndbd|ndbmtd|mysqld|ndbapi}
```

This command is used in the MySQL Cluster Manager client to obtain configuration attribute values from a MySQL NDB Cluster. (See Section 5.5, “MySQL Cluster Manager Configuration Commands”, for a definition of the term “attribute” as it applies in the MySQL Cluster Manager.) The output includes the following columns:

- `Name`: This column contains the name of the configuration attribute.
- `Value`: This column shows the attribute's current value.
- `Process1`: This column holds the process type to which the attribute applies. This is one of `ndb_mgmd`, `ndbd`, `ndbmtd`, or `mysqld`.
- `Id1`: This is the process ID of the process to which the attribute applies.
- `Process2`: For attributes that require specifying two nodes, such as those relating to TCP/IP connections, this column shows the process type of the second node.
- `Id2`: For attributes that require specifying two nodes, this column shows the process ID for the second node.
- `Level`: This is the attribute process level. This value in this column can be `Default`, `Process`, or empty; if this column is empty, it means that the attribute applies on the instance level.
- `Comment`: This column is used to show whether the attribute is `Mandatory`, `Read only`, `Default` attribute, or user defined (in which case the `Comment` column is empty).

By default, `get` returns only those attributes that have been set explicitly, either by the MySQL Cluster Manager itself, or by the user. In other words, it shows only attributes that are mandatory (including read-only attributes), or that have been set by the user after the cluster was created. Hereafter in this discussion, we refer to these as “non-default attributes”.

Thus, prior to setting any configuration attributes, you can obtain a list of all mandatory and read-only attributes by running the simplest possible form of this command, as shown here:

```
mcm> get mycluster\G
*************************** 1. row ***************************
    Name: Name
   Value: mycluster
Process1:
 NodeId1:
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 2. row ***************************
    Name: DataDir
   Value: /opt/mcm_data/clusters/mycluster/49/data
Process1: ndb_mgmd
 NodeId1: 49
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 3. row ***************************
    Name: HostName
   Value: torsk
Process1: ndb_mgmd
 NodeId1: 49
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 4. row ***************************
    Name: NodeId
   Value: 49
Process1: ndb_mgmd
 NodeId1: 49
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 5. row ***************************
    Name: PortNumber
   Value: 1186
Process1: ndb_mgmd
 NodeId1: 49
Process2:
 NodeId2:
   Level: Process
 Comment:
*************************** 6. row ***************************
    Name: DataDir
   Value: /opt/mcm_data/clusters/mycluster/1/data
Process1: ndbmtd
 NodeId1: 1
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 7. row ***************************
    Name: HostName
   Value: torsk
Process1: ndbmtd
 NodeId1: 1
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 8. row ***************************
    Name: NodeId
   Value: 1
Process1: ndbmtd
 NodeId1: 1
```

```
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 9. row ***************************
    Name: DataDir
   Value: /opt/mcm_data/clusters/mycluster/2/data
Process1: ndbmtd
 NodeId1: 2
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 10. row ***************************
    Name: HostName
   Value: torsk
Process1: ndbmtd
 NodeId1: 2
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 11. row ***************************
    Name: NodeId
   Value: 2
Process1: ndbmtd
 NodeId1: 2
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 12. row ***************************
    Name: datadir
   Value: /opt/mcm_data/clusters/mycluster/50/data
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 13. row ***************************
    Name: default_storage_engine
   Value: ndbcluster
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level: Process
 Comment:
*************************** 14. row ***************************
    Name: HostName
   Value: torsk
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 15. row ***************************
    Name: ndb_nodeid
   Value: 50
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 16. row ***************************
    Name: ndbcluster
   Value: on
Process1: mysqld
 NodeId1: 50
```

```
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 17. row ***************************
    Name: NodeId
   Value: 50
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 18. row ***************************
    Name: port
   Value: 3306
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 19. row ***************************
    Name: socket
   Value: /tmp/mysql.mycluster.50.sock
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 20. row ***************************
    Name: tmpdir
   Value: /opt/mcm_data/clusters/mycluster/50/tmp
Process1: mysqld
 NodeId1: 50
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 21. row ***************************
    Name: datadir
   Value: /opt/mcm_data/clusters/mycluster/51/data
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 22. row ***************************
    Name: default_storage_engine
   Value: ndbcluster
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level: Process
 Comment:
*************************** 23. row ***************************
    Name: HostName
   Value: torsk
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 24. row ***************************
    Name: ndb_nodeid
   Value: 51
Process1: mysqld
 NodeId1: 51
```

```
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 25. row ***************************
    Name: ndbcluster
   Value: on
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 26. row ***************************
    Name: NodeId
   Value: 51
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 27. row ***************************
    Name: port
   Value: 3307
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 28. row ***************************
    Name: socket
   Value: /tmp/mysql.mycluster.51.sock
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 29. row ***************************
    Name: tmpdir
   Value: /opt/mcm_data/clusters/mycluster/51/tmp
Process1: mysqld
 NodeId1: 51
Process2:
 NodeId2:
   Level:
 Comment:
*************************** 30. row ***************************
    Name: NodeId
   Value: 52
Process1: ndbapi
 NodeId1: 52
Process2:
 NodeId2:
   Level:
 Comment: Read only
30 rows in set (0.07 sec)
```

On Windows, no substitutions for backslashes or other characters used in values of paths reported by the `get` command is performed. However, it is possible to see forward slashes used in such paths if the values were set using the `set` command. See Setting Attributes Containing Paths on Windows [129], for more information.

Although a `socket` attribute is shown for `mysqld` nodes in the `get` output from the previous example and is not marked `Read only`, MySQL Cluster Manager does not support socket files on Windows. For this reason; you should not attempt to set `socket` attributes for Windows `mysqld` processes using MySQL Cluster Manager.

To include default values for attributes that have not (or not yet) been set explicitly, you can invoke this command with the `--include-defaults` option (short form: `-d`), as shown here (in part):

```
mcm> get --include-defaults mycluster\G
*************************** 1. row ***************************
    Name: Name
   Value: mycluster
Process1:
 NodeId1:
Process2:
 NodeId2:
   Level:
 Comment: Read only
*************************** 2. row ***************************
    Name: Checksum
   Value: false
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 3. row ***************************
    Name: Group
   Value: 55
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 4. row ***************************
    Name: HostName1
   Value: NULL
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 5. row ***************************
    Name: HostName2
   Value: NULL
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 6. row ***************************
    Name: NodeId1
   Value: NULL
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment: Mandatory
*************************** 7. row ***************************
    Name: NodeId2
   Value: NULL
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment: Mandatory
*************************** 8. row ***************************
    Name: NodeIdServer
   Value: NULL
Process1: ndb_mgmd
 NodeId1: 49
```

```
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment: Mandatory
*************************** 9. row ***************************
    Name: OverloadLimit
   Value: 0
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 10. row ***************************
    Name: Proxy
   Value: NULL
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 11. row ***************************
    Name: ReceiveBufferMemory
   Value: 2097152
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 12. row ***************************
    Name: SendBufferMemory
   Value: 2097152
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 13. row ***************************
    Name: SendSignalId
   Value: true
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:
*************************** 14. row ***************************
    Name: TCP_MAXSEG_SIZE
   Value: 0
Process1: ndb_mgmd
 NodeId1: 49
Process2: ndbmtd
 NodeId2: 1
   Level: Default
 Comment:

...

*************************** 1901. row ***************************
    Name: StartConnectBackoffMaxTime
   Value: 0
Process1: ndbapi
 NodeId1: 52
Process2:
 NodeId2:
   Level: Default
 Comment:
*************************** 1902. row ***************************
    Name: TotalSendBufferMemory
```

```
   Value: 0
Process1: ndbapi
 NodeId1: 52
Process2:
 NodeId2:
   Level: Default
 Comment:
*************************** 1903. row ***************************
    Name: wan
   Value: false
Process1: ndbapi
 NodeId1: 52
Process2:
 NodeId2:
   Level: Default
 Comment:
1903 rows in set (0.11 sec)
```

As you can see, the output from this `get` command is quite long (and the number of rows generated increases with the number of nodes in the cluster.) However, it is possible to filter the output so that you can view only the attribute or attributes in which you are interested. This can be done by using a comma-separated list of one or more filter specifications. A filter specification is defined as shown here (condensed from that given at the beginning of this section, but effectively the same):

```
[attribute_name][:[process_name][:process_id]]
```

Filtering can be applied per attribute, per process type, and per process instance. We now provide some examples illustrating the use of such filters.

To obtain the value of a given attribute for all processes to which it applies in the cluster, you need only use the name of the attribute as a filter. For example, to obtain the `HostName` of all processes in the cluster named `mycluster`, you can execute the command shown here:

```
mcm> get HostName mycluster;
+----------+----------+----------+---------+----------+---------+-------+-----------+
| Name     | Value    | Process1 | NodeId1 | Process2 | NodeId2 | Level | Comment   |
+----------+----------+----------+---------+----------+---------+-------+-----------+
| HostName | flundra  | ndbd     | 1       |          |         |       | Read only |
| HostName | tonfisk  | ndbd     | 2       |          |         |       | Read only |
| HostName | grindval | ndb_mgmd | 49      |          |         |       | Read only |
| HostName | haj      | mysqld   | 50      |          |         |       | Read only |
| HostName | torsk    | mysqld   | 51      |          |         |       | Read only |
+----------+----------+----------+---------+----------+---------+-------+-----------+
5 rows in set (0.04 sec)
```

The wildcard `*` (asterisk character) can be used to match a single or multiple attribute names; for example:

```
mcm> get Host* mycluster;
+----------+----------+----------+---------+----------+---------+-------+-----------+
| Name     | Value    | Process1 | NodeId1 | Process2 | NodeId2 | Level | Comment   |
+----------+----------+----------+---------+----------+---------+-------+-----------+
| HostName | flundra  | ndbd     | 1       |          |         |       | Read only |
| HostName | tonfisk  | ndbd     | 2       |          |         |       | Read only |
| HostName | grindval | ndb_mgmd | 49      |          |         |       | Read only |
| HostName | haj      | mysqld   | 50      |          |         |       | Read only |
| HostName | torsk    | mysqld   | 51      |          |         |       | Read only |
+----------+----------+----------+---------+----------+---------+-------+-----------+
5 rows in set (0.04 sec)
```

```
mcm> get H* yourcluster;
+------------------------+---------+----------+---------+----------+---------+---------+-----------+
| Name                   | Value   | Process1 | NodeId1 | Process2 | NodeId2 | Level   | Comment   |
+------------------------+---------+----------+---------+----------+---------+---------+-----------+
| HostName               | tonfisk | ndb_mgmd | 49      |          |         |         | Read only |
| HostName               | flundra | ndb_mgmd | 53      |          |         |         | Read only |
| HeartbeatIntervalDbApi | 1500    | ndbmtd   | 1       |          |         | Process |           |
| HeartbeatIntervalDbDb  | 1500    | ndbmtd   | 1       |          |         | Process |           |
| HostName               | tonfisk | ndbmtd   | 1       |          |         |         | Read only |
```

```
| HeartbeatIntervalDbApi | 1500    | ndbmtd   | 2       |          |         | Process |           |
| HeartbeatIntervalDbDb  | 1500    | ndbmtd   | 2       |          |         | Process |           |
| HostName               | flundra | ndbmtd   | 2       |          |         |         | Read only |
| HostName               | tonfisk | mysqld   | 50      |          |         |         | Read only |
| HostName               | flundra | mysqld   | 51      |          |         |         | Read only |
+------------------------+---------+----------+---------+----------+---------+---------+-----------+
10 rows in set (0.09 sec)
```

To obtain the value of a given attribute for all processes of a given type, you can specify a filter of the form *attribute_name*:*process_name*. The following command retrieves the HostName of all ndbd processes (only) in the cluster mycluster:

```
mcm> get HostName:ndbd mycluster;
+----------+---------+----------+------+----------+------+-------+----------+
| Name     | Value   | Process1 | Id1  | Process2 | Id2  | Level | Comment  |
+----------+---------+----------+------+----------+------+-------+----------+
| HostName | flundra | ndbd     | 1    |          |      |       | Readonly |
| HostName | tonfisk | ndbd     | 2    |          |      |       | Readonly |
+----------+---------+----------+------+----------+------+-------+----------+
2 rows in set (0.12 sec)
```

To retrieve the value of a given attribute for a particular instance of a process, you can use a filter that takes the form *attribute_name*:*process_name*:*process_id*. For example, you can use the following command to obtain the hostname for the process having 2 as its process ID:

```
mcm> get HostName:ndbd:2 mycluster;
+----------+---------+----------+------+----------+------+-------+----------+
| Name     | Value   | Process1 | Id1  | Process2 | Id2  | Level | Comment  |
+----------+---------+----------+------+----------+------+-------+----------+
| HostName | tonfisk | ndbd     | 2    |          |      |       | Readonly |
+----------+---------+----------+------+----------+------+-------+----------+
1 row in set (1.67 sec)
```

The command works the same if the process type is omitted:

```
mcm> get HostName::2 mycluster;
+----------+---------+----------+------+----------+------+-------+----------+
| Name     | Value   | Process1 | Id1  | Process2 | Id2  | Level | Comment  |
+----------+---------+----------+------+----------+------+-------+----------+
| HostName | tonfisk | ndbd     | 2    |          |      |       | Readonly |
+----------+---------+----------+------+----------+------+-------+----------+
1 row in set (1.67 sec)
```

You can obtain information about multiple attributes within a single get command by specifying a list of filters, separated by commas. *Each filter in the list must be a complete, valid filter*. The command shown here retrieves the HostName and DataDir for all processes in mycluster:

```
mcm> get HostName,DataDir mycluster;
+----------+-------------+----------+---------+----------+---------+-------+-----------+
| Name     | Value       | Process1 | NodeId1 | Process2 | NodeId2 | Level | Comment   |
+----------+-------------+----------+---------+----------+---------+-------+-----------+
| DataDir  | /opt/c1data | ndbd     | 1       |          |         |       |           |
| HostName | flundra     | ndbd     | 1       |          |         |       | Read only |
| DataDir  | /opt/c2data | ndbd     | 2       |          |         |       |           |
| HostName | tonfisk     | ndbd     | 2       |          |         |       | Read only |
| DataDir  | /opt/c49data | ndb_mgmd | 49      |          |         |       |           |
| HostName | grindval    | ndb_mgmd | 49      |          |         |       | Read only |
| datadir  | /opt/c50data | mysqld   | 50      |          |         |       |           |
| HostName | haj         | mysqld   | 50      |          |         |       | Read only |
| datadir  | /opt/c51data | mysqld   | 51      |          |         |       |           |
| HostName | torsk       | mysqld   | 51      |          |         |       | Read only |
+----------+-------------+----------+---------+----------+---------+-------+-----------+
10 rows in set (0.05 sec)
```

To retrieve the values of HostName and DataDir for only the data nodes in mycluster, you can use the get command shown here:

```
mcm> get HostName:ndbd,DataDir:ndbd mycluster;
+----------+------------+----------+-----+----------+-----+-------+-----------+
```

```
| Name     | Value       | Process1 | Id1 | Process2 | Id2 | Level | Comment   |
+----------+-------------+----------+-----+----------+-----+-------+-----------+
| DataDir  | /opt/c2data | ndbd     | 1   |          |     |       |           |
| HostName | tonfisk     | ndbd     | 1   |          |     |       | Read only |
| DataDir  | /opt/c3data | ndbd     | 2   |          |     |       |           |
| HostName | flundra     | ndbd     | 2   |          |     |       | Read only |
+----------+-------------+----------+-----+----------+-----+-------+-----------+
4 rows in set (1.36 sec)
```

In the example just shown, each filter includes a process type specifier. If you omit this specifier from one of the filters, you obtain a result that you might not expect:

```
mcm> get HostName,DataDir:ndbd mycluster;
+----------+-------------+----------+-----+----------+-----+-------+-----------+
| Name     | Value       | Process1 | Id1 | Process2 | Id2 | Level | Comment   |
+----------+-------------+----------+-----+----------+-----+-------+-----------+
| HostName | grindval    | ndb_mgmd | 49  |          |     |       | Read only |
| DataDir  | /opt/c2data | ndbd     | 1   |          |     |       |           |
| HostName | tonfisk     | ndbd     | 1   |          |     |       | Read only |
| DataDir  | /opt/c3data | ndbd     | 2   |          |     |       |           |
| HostName | flundra     | ndbd     | 2   |          |     |       | Read only |
| HostName | haj         | mysqld   | 50  |          |     |       | Read only |
| HostName | torsk       | mysqld   | 51  |          |     |       | Read only |
+----------+-------------+----------+-----+----------+-----+-------+-----------+
6 rows in set (0.58 sec)
```

The filter list `HostName,DataDir:ndbd` is perfectly valid. However, it actually consists of the filters `HostName` and `DataDir:ndbd`—in other words, it means “the `HostName` for all processes, and the `DataDir` for `ndbd` processes”.

Suppose you wish to obtain the values for `HostName` for just the `ndb_mgmd` and `mysqld` processes in `mycluster`. You might be tempted to try using something like `HostName:ndb_mgmd,mysqld` for the filter list, but this does not work, as you can see here:

```
mcm> get HostName:ndb_mgmd,mysqld mycluster;
ERROR 6003 (00MGR): No such config variable mysqld for process
```

This is due to the fact that each filter in the filter list must be a valid filter, and must include an attribute name. (In the filter list just shown, MySQL Cluster Manager tries to interpret the first string following the comma as an attribute name.) The correct filter list to use in a `get` command for retrieving the `HostName` for the `ndb_mgmd` and `mysqld` processes in `mycluster` is shown in this example:

```
mcm> get HostName:ndb_mgmd,HostName:mysqld mycluster;
+----------+----------+----------+------+----------+------+-------+-----------+
| Name     | Value    | Process1 | Id1  | Process2 | Id2  | Level | Comment   |
+----------+----------+----------+------+----------+------+-------+-----------+
| HostName | grindval | ndb_mgmd | 49   |          |      |       | Read only |
| HostName | haj      | mysqld   | 50   |          |      |       | Read only |
| HostName | torsk    | mysqld   | 51   |          |      |       | Read only |
+----------+----------+----------+------+----------+------+-------+-----------+
2 rows in set (0.21 sec)
```

It is also possible to obtain a list of attributes and their values for a given process type or instance of a process. For a given process type, use a filter having the form `:process_name`. For example, to retrieve all non-default attributes applying to `ndbd` processes in a cluster named `mycluster`, you can use the filter `:ndbd`, as shown here:

```
mcm> get :ndbd mycluster;
+--------------+-------------+----------+-----+----------+-----+-------+-----------+
| Name         | Value       | Process1 | Id1 | Process2 | Id2 | Level | Comment   |
+--------------+-------------+----------+-----+----------+-----+-------+-----------+
| DataDir      | /opt/c2data | ndbd     | 1   |          |     |       |           |
| HostName     | tonfisk     | ndbd     | 1   |          |     |       | Read only |
| NodeId       | 1           | ndbd     | 1   |          |     |       | Read only |
| DataDir      | /opt/c3data | ndbd     | 2   |          |     |       |           |
| HostName     | flundra     | ndbd     | 2   |          |     |       | Read only |
| NodeId       | 2           | ndbd     | 2   |          |     |       | Read only |
+--------------+-------------+----------+-----+----------+-----+-------+-----------+
6 rows in set (0.77 sec)
```

(The example just shown assumes that no attributes are set to non-default values.)

To get a list of all non-default attributes for a single instance of a process, use a filter having the form *:process_name:process_id*, as shown in this example, which retrieves all non-default attributes for the ndbd process having 2 as its process ID:

```
mcm> get :ndbd:2 mycluster;
+-------------+-------------+----------+-----+----------+-----+-------+-----------+
| Name        | Value       | Process1 | Id1 | Process2 | Id2 | Level | Comment   |
+-------------+-------------+----------+-----+----------+-----+-------+-----------+
| DataDir     | /opt/c2data | ndbd     | 2   |          |     |       |           |
| HostName    | flundra     | ndbd     | 2   |          |     |       | Read only |
| NodeId      | 2           | ndbd     | 2   |          |     |       | Read only |
+-------------+-------------+----------+-----+----------+-----+-------+-----------+
4 rows in set (0.32 sec)
```

If you try to obtain values for an attribute that you know is supported by your MySQL NDB Cluster version, but the result is empty, this almost certainly means that it is a default attribute that either has not been changed since the cluster was created or has been reset. In order to view default attributes using get, you must execute the command using the --include-defaults option (short form: -d).

Suppose you want to see how much DataMemory is configured for the ndbd processes in the cluster named mycluster, and you execute what appears to be the correct get command, but an empty result is returned, as shown here:

```
mcm> get DataMemory:ndbd mycluster;
Empty set (1.19 sec)
```

This means that the DataMemory attribute has its default value for all data nodes in the cluster. If you do not recall what this value is, you can determine it easily by repeating the same command with the addition of the --include-defaults (-d) option:

```
mcm> get --include-defaults DataMemory:ndbd mycluster;
+------------+----------+----------+-----+----------+-----+---------+---------+
| Name       | Value    | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------+----------+----------+-----+----------+-----+---------+---------+
| DataMemory | 83886080 | ndbd     | 1   |          |     | Default |         |
| DataMemory | 83886080 | ndbd     | 2   |          |     | Default |         |
+------------+----------+----------+-----+----------+-----+---------+---------+
2 rows in set (0.62 sec)
```

Now suppose that you increase the DataMemory to 500 megabytes per data node, then repeat the get command to verify the new value:

```
mcm> set DataMemory:ndbd=500M mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.77 sec)

mcm> get --include-defaults DataMemory:ndbd mycluster;
+------------+-------+----------+-----+----------+-----+---------+---------+
| Name       | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------+-------+----------+-----+----------+-----+---------+---------+
| DataMemory | 500M  | ndbd     | 1   |          |     | Process |         |
| DataMemory | 500M  | ndbd     | 2   |          |     | Process |         |
+------------+-------+----------+-----+----------+-----+---------+---------+
2 rows in set (1.46 sec)
```

You can see that, not only has the Value column in the get command output been updated to the new value, but the Level column has also been updated from Default to Process. This means that you no longer need the --include-defaults option to view this attribute, as shown here:

```
mcm> get DataMemory:ndbd mycluster;
+------------+-------+----------+-----+----------+-----+---------+---------+
| Name       | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
```

```
+------------+-------+----------+-----+----------+-----+---------+---------+
| DataMemory | 500M  | ndbd     | 1   |          |     | Process |         |
| DataMemory | 500M  | ndbd     | 2   |          |     | Process |         |
+------------+-------+----------+-----+----------+-----+---------+---------+
2 rows in set (0.63 sec)
```

However, if you reset `DataMemory` (also on the process level), this is no longer the case. Then, `DataMemory` once again assumes its default value, after which you must use the `--include-defaults` option to retrieve it, as shown in this example:

```
mcm> reset DataMemory:ndbd mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.65 sec)

mcm> get DataMemory:ndbd mycluster;
Empty set (1.76 sec)

mcm> get --include-defaults DataMemory:ndbd mycluster;
+------------+----------+----------+-----+----------+-----+---------+---------+
| Name       | Value    | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------+----------+----------+-----+----------+-----+---------+---------+
| DataMemory | 83886080 | ndbd     | 1   |          |     | Default |         |
| DataMemory | 83886080 | ndbd     | 2   |          |     | Default |         |
+------------+----------+----------+-----+----------+-----+---------+---------+
2 rows in set (1.01 sec)
```

For more information about these commands, see Section 5.5.3, “The `set` Command”, and Section 5.5.2, “The `reset` Command”.

The `get` command also tags multi-entry replication attributes as so in the `Comment` column; for example:

```
mcm> get replicate_ignore_table:mysqld mycluster;
+------------------------+--------------+----------+---------+----------+---------+---------+-------------
| Name                   | Value        | Process1 | NodeId1 | Process2 |NodeId2  | Level   | Comment     |
+------------------------+--------------+----------+---------+----------+---------+---------+-------------
| replicate_ignore_table | mydb.t1      | mysqld   | 50      |          |         |         | Multi-entry |
| replicate_ignore_table | mydb.t50     | mysqld   | 50      |          |         |         | Multi-entry |
| replicate_ignore_table | mydb.mytable | mysqld   | 50      |          |         | Process | Multi-entry |
| replicate_ignore_table | mydb.t51     | mysqld   | 51      |          |         |         | Multi-entry |
| replicate_ignore_table | mydb.mytable | mysqld   | 51      |          |         | Process | Multi-entry |
+------------------------+--------------+----------+---------+----------+---------+---------+-------------
5 rows in set (0.05 sec)
```

On how to reset multi-entry attributes, see Section 5.5.2, “The `reset` Command”.

The `get` command does not normally display configuration attributes applying to TCP or SHM connections. However, such attributes can be set in the MySQL Cluster Manager client (using the `set` command); and once they have been set, they are displayed by applicable `get` commands. See Setting TCP Connection Attributes, which provides an example of this.

The `--all` option of the `get` command creates two extra columns, `Type` and `Restart`, in the output:

```
mcm> get -d --all TimeBetween*:ndbmtd mycluster;

+---------------------------------------+--------+----------+---------+----------+---------+---------+--
| Name                                  | Value  | Process1 | NodeId1 | Process2 | NodeId2 | Level   | T
+---------------------------------------+--------+----------+---------+----------+---------+---------+--
| TimeBetweenEpochs                     | 100    | ndbmtd   | 1       |          |         | Default | u
| TimeBetweenEpochsTimeout              | 0      | ndbmtd   | 1       |          |         | Default | u
| TimeBetweenGlobalCheckpoints          | 2000   | ndbmtd   | 1       |          |         | Default | u
| TimeBetweenGlobalCheckpointsTimeout   | 120000 | ndbmtd   | 1       |          |         | Default | u
| TimeBetweenInactiveTransactionAbortCheck | 1000   | ndbmtd   | 1       |          |         | Default | u
| TimeBetweenLocalCheckpoints           | 20     | ndbmtd   | 1       |          |         | Default | u
| TimeBetweenWatchDogCheck              | 6000   | ndbmtd   | 1       |          |         | Default | u
```

```
| TimeBetweenWatchDogCheckInitial          | 6000   | ndbmtd   | 1       |          |          | Default
| TimeBetweenEpochs                        | 100    | ndbmtd   | 2       |          |          | Default
| TimeBetweenEpochsTimeout                 | 0      | ndbmtd   | 2       |          |          | Default
| TimeBetweenGlobalCheckpoints             | 2000   | ndbmtd   | 2       |          |          | Default
| TimeBetweenGlobalCheckpointsTimeout      | 120000 | ndbmtd   | 2       |          |          | Default
| TimeBetweenInactiveTransactionAbortCheck | 1000   | ndbmtd   | 2       |          |          | Default
| TimeBetweenLocalCheckpoints              | 20     | ndbmtd   | 2       |          |          | Default
| TimeBetweenWatchDogCheck                 | 6000   | ndbmtd   | 2       |          |          | Default
| TimeBetweenWatchDogCheckInitial          | 6000   | ndbmtd   | 2       |          |          | Default
+------------------------------------------+--------+----------+---------+----------+----------+--------
16 rows in set (0.11 sec)
```

The Type column shows the expected type of the configuration attribute.

The Restart column shows the expected process restarts needed when changing the configuration attribute. There are three kinds of values in the column

- all means all nodes need restarts.
- A specific node type (ndbd, ndbmtd, ndb_mgmd, or mysqld) means all nodes of the type need restarts.
- instance means only the process instances affected by the change need restarts.

5.5.2 The reset Command

```
reset [--sequential-restart] filter_specification_list cluster_name

filter_specification_list:
    filter_specification[,filter_specification][,...]

filter_specification:
    attribute_name[:process_specification][+process_specification]]

process_specification:
    [process_name][:process_id]

process_name:
    {ndb_mgmd|ndbd|ndbmtd|mysqld|ndbapi}
```

This command resets an attribute to its default value. Attributes can be set on either the process level or instance level. To reset an attribute on the process level, use a filter specification having the form *attribute_name*:*process_name*, where *attribute_name* is the name of the attribute to be reset, and *process_name* is the name of a MySQL NDB Cluster process. To reset a configuration attribute on the instance level, use a filter specification of the form *attribute_name*:*process_name*:*process_id*, where *process_id* is the process ID.

You cannot issue a reset command that resets all values for a given configuration attribute regardless of process type; each reset command must specify a process type or instance of a process. Otherwise, the command fails, as shown here:

```
mcm> reset DataMemory mycluster;
ERROR 3 (00MGR): Illegal syntax
```

You also cannot revert all configuration attributes for a given process type or instance of a process using a single filter specification; you must always include the name of the attribute to be reset. Otherwise, the reset command fails, as shown here:

```
mcm> reset :ndbd mycluster;
ERROR 3 (00MGR): Illegal syntax

mcm> reset :ndbd:3 mycluster;
ERROR 3 (00MGR): Illegal syntax
```

Suppose that the data memory for all ndbd processes in the cluster named mycluster has been set to 500 MB, as shown in the output of this get command:

```
mcm> get DataMemory mycluster;
+------------+-------+----------+-----+----------+-----+---------+---------+
| Name       | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------+-------+----------+-----+----------+-----+---------+---------+
| DataMemory | 500M  | ndbd     | 2   |          |     | Process |         |
| DataMemory | 500M  | ndbd     | 3   |          |     | Process |         |
+------------+-------+----------+-----+----------+-----+---------+---------+
2 rows in set (1.91 sec)
```

We can see from the entries in the Level column that the DataMemory setting for both ndbd processes applies on the process level. A process-level setting cannot be reset on the instance level, as shown here:

```
mcm> reset DataMemory:ndbd:2 mycluster;
ERROR 6010 (00MGR): No matching user defined setting was
found for config attribute DataMemory
mcm> reset DataMemory:ndbd:3 mycluster;
ERROR 6010 (00MGR): No matching user defined setting was
found for config attribute DataMemory
```

The following reset command also does not work, although you might think that it would do so, since it attempts to reset the attribute's value for both ndbd processes:

```
mcm> reset DataMemory:ndbd:2,DataMemory:ndbd:3 mycluster;
ERROR 6010 (00MGR): No matching user defined setting was
found for config attribute DataMemory
```

The previous command fails because MySQL Cluster Manager regards this as an attempt to apply two instance-level configuration changes. Because the DataMemory setting is a process-level setting, you must instead reset DataMemory to its default value on the process level; you can do this by using the filter specification DataMemory:ndbd in the reset command, as shown here:

```
mcm> reset DataMemory:ndbd mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (6.16 sec)
```

If you execute the same get command as shown previously, the result is now empty:

```
mcm> get DataMemory mycluster;
Empty set (0.74 sec)
```

This is because the get command by default does not report default values. To retrieve the DataMemory values after resetting them, you must invoke get using the --include-defaults (short form: -d) option:

```
mcm> get --include-defaults DataMemory mycluster;
+------------+----------+----------+-----+----------+-----+---------+---------+
| Name       | Value    | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------+----------+----------+-----+----------+-----+---------+---------+
| DataMemory | 83886080 | ndbd     | 2   |          |     | Default |         |
| DataMemory | 83886080 | ndbd     | 3   |          |     | Default |         |
+------------+----------+----------+-----+----------+-----+---------+---------+
2 rows in set (1.21 sec)
```

The DataMemory values are now included in the output, and are marked with the word Default in the Comments column.

Now suppose that the mysqld configuration attribute wait_timeout for the mysqld process having the ID 4 in the cluster named mycluster has previously been set to the value 200 as shown here, and that no other changes have been to this attribute:

```
mcm> set wait_timeout:mysqld:4=200 mycluster;
+----------------------------------+
```

```
| Command result                    |
+-----------------------------------+
| Cluster reconfigured successfully |
+-----------------------------------+
1 row in set (7.78 sec)

mcm> get -d wait_timeout:mysqld:4 mycluster;
+--------------+-------+----------+-----+----------+-----+-------+---------+
| Name         | Value | Process1 | Id1 | Process2 | Id2 | Level | Comment |
+--------------+-------+----------+-----+----------+-----+-------+---------+
| wait_timeout | 200   | mysqld   | 4   |          |     |       |         |
+--------------+-------+----------+-----+----------+-----+-------+---------+
1 row in set (0.98 sec)
```

Because the `Level` column is empty, we know that this setting applies on the instance level. If you try to reset it on the process level, the attempt fails, as shown here:

```
mcm> reset wait_timeout:mysqld mycluster2;
ERROR 6010 (00MGR): No matching user defined setting was
found for config attribute wait_timeout
```

If you wish to reset this attribute to its default value, you must use the `reset` command with the instance-level filter specification `wait_timeout:mysqld:4`, as shown here:

```
mcm> reset wait_timeout:mysqld:4 mycluster;
+-----------------------------------+
| Command result                    |
+-----------------------------------+
| Cluster reconfigured successfully |
+-----------------------------------+
1 row in set (7.61 sec)
```

Once you have reset `wait_timeout`, it no longer appears in the output of the earlier `get` command:

```
mcm> get wait_timeout:mysqld mycluster;
Empty set (1.42 sec)
```

This is because the default behavior of the `get` command is to display only those values that have been set either by the MySQL Cluster Manager or by the user. Since `wait_timeout` has been allowed to revert to its default value, you must use the `--include-defaults` (short form: `-d`) option to retrieve it, as shown here:

```
mcm> get -d wait_timeout:mysqld mycluster;
+--------------+-------+----------+-----+----------+-----+---------+---------+
| Name         | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+--------------+-------+----------+-----+----------+-----+---------+---------+
| wait_timeout | 28800 | mysqld   | 4   |          |     | Default |         |
+--------------+-------+----------+-----+----------+-----+---------+---------+
1 row in set (1.66 sec)
```

Now consider a situation in which process-level and instance-level settings have been made to a configuration attribute; in this example, we use `IndexMemory`. First, verify that `IndexMemory` is set to its default value for all data node processes (in this case, there are two of them):

```
mcm> get -d IndexMemory mycluster;
+-------------+----------+----------+-----+----------+-----+---------+---------+
| Name        | Value    | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+-------------+----------+----------+-----+----------+-----+---------+---------+
| IndexMemory | 18874368 | ndbd     | 2   |          |     | Default |         |
| IndexMemory | 18874368 | ndbd     | 3   |          |     | Default |         |
+-------------+----------+----------+-----+----------+-----+---------+---------+
2 rows in set (1.24 sec)
```

Now apply both a process-level change and an instance-level change to this attribute. You can do this with a single `set` command, as shown here:

```
mcm> set IndexMemory:ndbd=500M,IndexMemory:ndbd:3=750M mycluster;
+-----------------------------------+
| Command result                    |
```

```
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.29 sec)
```

Because the process-level change was specified first, it is overridden for the ndbd process by the instance-level change specified second. The output from the following get command confirms that this is the case:

```
mcm> get IndexMemory mycluster;
+-------------+-------+----------+-----+----------+-----+---------+---------+
| Name        | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+-------------+-------+----------+-----+----------+-----+---------+---------+
| IndexMemory | 500M  | ndbd     | 2   |          |     | Process |         |
| IndexMemory | 750M  | ndbd     | 3   |          |     |         |         |
+-------------+-------+----------+-----+----------+-----+---------+---------+
2 rows in set (0.85 sec)
```

If the instance-level IndexMemory setting for the ndbd process with process ID 3 is reset, the process-level setting still applies, as shown here:

```
mcm> reset IndexMemory:ndbd:3 mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (6.41 sec)

mcm> get IndexMemory mycluster;
+-------------+-------+----------+-----+----------+-----+---------+---------+
| Name        | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+-------------+-------+----------+-----+----------+-----+---------+---------+
| IndexMemory | 500M  | ndbd     | 2   |          |     | Process |         |
| IndexMemory | 500M  | ndbd     | 3   |          |     | Process |         |
+-------------+-------+----------+-----+----------+-----+---------+---------+
2 rows in set (1.09 sec)
```

Now, re-apply the instance-level IndexMemory setting, and verify using get that it has taken effect:

```
mcm> set IndexMemory:ndbd:3=750M mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (6.79 sec)

mcm> get IndexMemory mycluster;
+-------------+-------+----------+-----+----------+-----+---------+---------+
| Name        | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+-------------+-------+----------+-----+----------+-----+---------+---------+
| IndexMemory | 500M  | ndbd     | 2   |          |     | Process |         |
| IndexMemory | 750M  | ndbd     | 3   |          |     |         |         |
+-------------+-------+----------+-----+----------+-----+---------+---------+
2 rows in set (1.76 sec)
```

If you reset the process-level setting, the instance-level setting remains, and only the ndbd process having process ID 2 has its IndexMemory reset to the default value; the instance-level setting remains in effect, as you can see from the following sequence of commands:

```
mcm> reset IndexMemory:ndbd mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.36 sec)

mcm> get -d IndexMemory mycluster;
```

```
+-------------+----------+----------+-----+----------+-----+---------+---------+
| Name        | Value    | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+-------------+----------+----------+-----+----------+-----+---------+---------+
| IndexMemory | 18874368 | ndbd     | 2   |          |     | Default |         |
| IndexMemory | 750M     | ndbd     | 3   |          |     |         |         |
+-------------+----------+----------+-----+----------+-----+---------+---------+
2 rows in set (0.10 sec)
```

Note

If the order of the specifiers in the original command that set `IndexMemory` had been reversed as `IndexMemory:ndbd:3=750M,IndexMemory:ndbd=500M`, the instance-level change would have been overridden by the process-level change, and the resulting `IndexMemory` setting for both `ndbd` processes would be `500M`. As discussed elsewhere, a process-level setting made after an instance-level setting that affects the same process completely removes the instance-level setting; the instance-level setting is not preserved, and resetting the attribute on the process level merely restores the default setting for all processes of that type. See Section 5.5, "MySQL Cluster Manager Configuration Commands", for more information.

The `get` and `reset` commands fully support multi-entry replication attributes; for example, if the `replicate_ignore_table` attribute has multiple entries:

```
mcm> get replicate_ignore_table:mysqld mycluster;
+-----------------------+--------------+----------+---------+----------+---------+---------+----------
| Name                  | Value        | Process1 | NodeId1 | Process2 |NodeId2  | Level   | Comment
+-----------------------+--------------+----------+---------+----------+---------+---------+----------
| replicate_ignore_table | mydb.t1      | mysqld   | 50      |          |         |         | Multi-ent
| replicate_ignore_table | mydb.t50     | mysqld   | 50      |          |         |         | Multi-ent
| replicate_ignore_table | mydb.mytable | mysqld   | 50      |          |         | Process | Multi-ent
| replicate_ignore_table | mydb.t51     | mysqld   | 51      |          |         |         | Multi-ent
| replicate_ignore_table | mydb.mytable | mysqld   | 51      |          |         | Process | Multi-ent
+-----------------------+--------------+----------+---------+----------+---------+---------+----------
5 rows in set (0.05 sec)
```

Without specifying a node ID, all the attribute's entries associated with the specified process type are reset with the following command:

```
mcm> reset replicate_ignore_table:mysqld mycluster;  # removes all process level entries
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (0.47 sec)

mcm> get replicate_ignore_table:mysqld mycluster;
+-----------------------+----------+----------+---------+----------+---------+-------+-------------+
| Name                  | Value    | Process1 | NodeId1 | Process2 | NodeId2 | Level | Comment     |
+-----------------------+----------+----------+---------+----------+---------+-------+-------------+
| replicate_ignore_table | mydb.t1  | mysqld   | 50      |          |         |       | Multi-entry |
| replicate_ignore_table | mydb.t50 | mysqld   | 50      |          |         |       | Multi-entry |
| replicate_ignore_table | mydb.t51 | mysqld   | 51      |          |         |       | Multi-entry |
+-----------------------+----------+----------+---------+----------+---------+-------+-------------+
3 rows in set (0.08 sec)
```

With a node ID specified, only the instance entries associated with the node ID are reset by the following command:

```
mcm> reset replicate_ignore_table:mysqld:51 mycluster;  # removes all instance level entries for nodeid
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (0.57 sec)
```

```
mcm> get replicate_ignore_table:mysqld mycluster;
+-----------------------+----------+----------+---------+----------+---------+-------+-------------+
| Name                  | Value    | Process1 | NodeId1 | Process2 | NodeId2 | Level | Comment     |
+-----------------------+----------+----------+---------+----------+---------+-------+-------------+
| replicate_ignore_table | mydb.t1  | mysqld   | 50      |          |         |       | Multi-entry |
| replicate_ignore_table | mydb.t50 | mysqld   | 50      |          |         |       | Multi-entry |
+-----------------------+----------+----------+---------+----------+---------+-------+-------------+
2 rows in set (0.09 sec)
```

reset commands are executed whether or not the cluster has been started. In a cluster that is not running, the MySQL Cluster Manager merely updates the configuration files. However, in a running cluster, the MySQL Cluster Manager in addition automatically performs any node restarts or rolling restarts (see Performing a Rolling Restart of an NDB Cluster) that are required to cause the attribute changes to take effect (use the --sequential-restart option to make the rolling restart a sequential one). However, since restart operations—particularly rolling restarts—can take a great deal of time, it is preferable to make configuration changes before starting the cluster and putting it into use.

Resetting TCP Connection Attributes. Certain configuration attributes, such as those relating to TCP connections, apply to connections between processes rather than to individual processes or individual process types. As shown elsewhere (see Setting Attributes for mysqld nodes), when you set such an attribute on the process level using MySQL Cluster Manager, this means that the attribute applies to all connections between the two types of processes specified when issuing the set command. It is also possible to set such an attribute on the instance level, in which case it applies only to a single connection between two process instances.

Similarly, it is possible to reset such an attribute on either the process or instance level, depending on the level or levels at which it was set. In either case, an extended form of the process specifier is required, just as it is when setting an attribute that applies to a connection between processes. Assume that the SendBufferMemory attribute has previously been set for all connections between the two ndbd processes and the two mysqld processes that are found in a MySQL NDB Cluster named mycluster2, as shown in the output of this get command:

```
mcm> get SendBufferMemory mycluster2;
+------------------+-------+----------+-----+----------+-----+---------+---------+
| Name             | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------------+-------+----------+-----+----------+-----+---------+---------+
| SendBufferMemory | 4M    | ndbd     | 2   | mysqld   | 4   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 2   | mysqld   | 5   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 3   | mysqld   | 4   | Process |         |
| SendBufferMemory | 8M    | ndbd     | 3   | mysqld   | 5   |         |         |
+------------------+-------+----------+-----+----------+-----+---------+---------+
4 rows in set (0.59 sec)
```

Suppose that you wish to reset SendBufferMemory only for the connection between the ndbd process having process ID 3 and the mysqld process having process ID 5. The SendBufferMemory setting that applies to this connection is specified on the instance level, as you can see because the Level column value corresponding to this connection is empty; this means that it is possible to reset this value on the instance level. You can do this using the reset command shown here:

```
mcm> reset SendBufferMemory:ndbd:3+mysqld:5 mycluster2;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.03 sec)
```

You can verify that the attribute was reset using the get command. However, as noted previously, once the instance-level setting has been removed, the process-level setting for this attribute again takes effect, so that the same setting applies to all connections between ndbd and mysqld processes, as shown here:

```
mcm> get SendBufferMemory mycluster2;
+------------------+-------+----------+-----+----------+-----+---------+---------+
```

```
| Name              | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+-------------------+-------+----------+-----+----------+-----+---------+---------+
| SendBufferMemory | 4M    | ndbd     | 2   | mysqld   | 4   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 2   | mysqld   | 5   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 3   | mysqld   | 4   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 3   | mysqld   | 5   | Process |         |
+-------------------+-------+----------+-----+----------+-----+---------+---------+
4 rows in set (0.87 sec)
```

To reset this attribute on the process level, you can use the following `reset` command:

```
mcm> reset SendBufferMemory:ndbd+mysqld mycluster2;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (8.01 sec)
```

You can verify that the attribute has been reset for all connection between `ndbd` processes and `mysqld` processes, by using the `get` command, as shown here:

```
mcm> get -d SendBufferMemory mycluster2;
Empty set (1.39 sec)
```

As noted elsewhere in this manual (see Section 5.5.1, “The `get` Command”), the empty result set is to be expected in this case, even when `get` is invoked using the `--include-defaults` (or `-d`) option, because the MySQL Cluster Manager client does not display attributes that appear in the `[tcp]` or `[shm]` sections of the `config.ini` configuration file if they have not been explicitly set by the user.

5.5.3 The `set` Command

- Performing a reset with a set command
- Setting Attributes for `mysqld` nodes
- Setting TCP Connection Attributes
- Setting up `mysqld` connection pooling
- Setting Up Encryption
- Overriding the Default Restart Type

```
set [--sequential-restart] [--retry] [--restart=restart_level] attribute_assignment_list cluster_name

attribute_assignment_list:
    attribute_assignment[,attribute_assignment][,...]

attribute_assignment:
    [~]attribute_name:process_specification[+process_specification][=value]

process_specification:
    [process_name][:process_id]

process_name:
    {ndb_mgmd|ndbd|ndbmtd|mysqld|ndbapi}

restart_level:
    {N|NI}
```

This command is used to set values for one or more configuration attributes. Attributes can be set on either the process level or instance level.

`set` commands are executed whether or not the cluster has been started. In a cluster that is not running, the MySQL Cluster Manager merely updates the configuration files. However, in a running cluster, the MySQL Cluster Manager in addition automatically performs any node restarts or rolling

restarts (see Performing a Rolling Restart of an NDB Cluster) that are required to cause the attribute changes to take effect. However, since restart operations—particularly rolling restarts—can take a great deal of time, it is preferable to make configuration changes before starting the cluster and putting it into use.

For any configuration options that normally require the nodes of the cluster to be restarted for the a running cluster to be reconfigured, if a `set` command is attempted with the same value as was already in use, the command returns an error, telling the user that the command results in no changes to the cluster. If it is really necessary to run the `set` command in the situation and force a restart of the relevant processes, use the `--retry` option.

Use the `--sequential-restart` option to make the rolling restart performed by the `set` command a sequential one.

Sets the path to a password file when NDB Cluster TDE is in use. This is actually implemented as a configuration attribute. Example:

```
mcm> set filesystem-password-file:ndbmtd:=/home/myndb/myc.pwd mycluster;
```

See Setting Up Encryption, for more information.

To set an attribute on the process level, use a `set` statement that contains an attribute assignment having the form *`attribute_name`*:*`process_name`*=*`value`*.

For example, to set `DataMemory` to 500 MB on the `ndbd` process level, so that the new value applies to all `ndbd` processes in the cluster, you can issue a `set` command containing the attribute assignment `DataMemory:ndbd=500M`, as shown here:

```
mcm> set DataMemory:ndbd=500M mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (5.68 sec)
```

To verify that the new setting is being used, you can issue the following `get` command:

```
mcm> get DataMemory mycluster;
+------------+-------+----------+------+----------+------+---------+---------+
| Name       | Value | Process1 | Id1  | Process2 | Id2  | Level   | Comment |
+------------+-------+----------+------+----------+------+---------+---------+
| DataMemory | 500M  | ndbd     | 1    |          |      | Process |         |
| DataMemory | 500M  | ndbd     | 2    |          |      | Process |         |
+------------+-------+----------+------+----------+------+---------+---------+
2 rows in set (0.79 sec)
```

Note

For more information about this command, see Section 5.5.1, “The `get` Command”.

To set an attribute for a specific process instance, include the process ID in the attribute assignment; the form of such an attribute assignment is *`attribute_name`*:*`process_name`*:*`process_id`*=*`value`*. For example, to set the wait_timeout attribute for the `mysqld` process that has process ID `50` to 200, you would issue a `set` command that contains the attribute assignment `wait_timeout:mysqld:51=200`, like this:

```
mcm> set wait_timeout:mysqld:50=200 mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (6.18 sec)
```

You can verify that the setting has taken effect using an applicable get command:

```
mcm> get wait_timeout mycluster;
+--------------+-------+----------+------+----------+------+-------+---------+
| Name         | Value | Process1 | Id1  | Process2 | Id2  | Level | Comment |
+--------------+-------+----------+------+----------+------+-------+---------+
| wait_timeout | 200   | mysqld   | 50   |          |      |       |         |
+--------------+-------+----------+------+----------+------+-------+---------+
1 row in set (0.50 sec)
```

Attributes that are marked Read only cannot be set. Attempting to do so fails with an error, as shown here:

```
mcm> get :ndbd mycluster;
+--------------+-------------+----------+-----+----------+-----+-------+-----------+
| Name         | Value       | Process1 | Id1 | Process2 | Id2 | Level | Comment   |
+--------------+-------------+----------+-----+----------+-----+-------+-----------+
| DataDir      | /opt/c2data | ndbd     | 1   |          |     |       |           |
| HostName     | tonfisk     | ndbd     | 1   |          |     |       | Read only |
| NodeId       | 2           | ndbd     | 1   |          |     |       | Read only |
| DataDir      | /opt/c3data | ndbd     | 2   |          |     |       |           |
| HostName     | grindval    | ndbd     | 2   |          |     |       | Read only |
| NodeId       | 3           | ndbd     | 2   |          |     |       | Read only |
+--------------+-------------+----------+-----+----------+-----+-------+-----------+
6 rows in set (1.42 sec)
mcm> set HostName:ndbd:1=lax mycluster;
ERROR 6008 (00MGR): Config attribute HostName is read only and cannot be changed
```

However, you can set mandatory attributes, such as in the example shown previously in this section where the DataDir configuration attribute was set to a user-defined value.

Warning

The mandatory NoOfReplicas attribute must be set on the process level only. Attempting to set it on the instance level may leave the cluster, the MySQL Cluster Manager, or both in an unusable configuration.

Unlike the case with the get command, you cannot issue a set acting on a “global” scope—that is, you cannot, in a single attribute assignment, set a single value for an attribute such that the new attribute value applies to all processes regardless of process type, even if the attribute having that name can be applied to all process types. Nor can you specify multiple process types in a single attribute assignment. Attempting to do either of these things causes an error, as shown here:

```
mcm> set DataDir=/var/cluster-data mycluster;
ERROR 3 (00MGR): Illegal syntax

mcm> set DataDir:ndb_mgmd,ndbd,mysqld=/var/cluster-data mycluster;
ERROR 3 (00MGR): Illegal syntax
```

Instead, you must use a process-level attribute assignment for each process type. However, you are not necessarily required to issue a separate set command for each process type. Instead, you can also make multiple attribute assignments in a single set command, supplying the assignments as a comma-separated list. This set command assigns /var/cdata as the data directory (DataDir) for all MySQL NDB Cluster processes in the cluster named mycluster:

```
mcm> set DataDir:ndb_mgmd=/var/cdata, \
           DataDir:ndbd=/var/cdata, \
           DataDir:mysqld=/var/cdata mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.66 sec)
```

```
mcm> get DataDir mycluster;
+---------+-----------+----------+---------+----------+---------+-------+---------+
| Name    | Value     | Process1 | NodeId1 | Process2 | NodeId2 | Level | Comment |
+---------+-----------+----------+---------+----------+---------+-------+---------+
| DataDir | /var/cdata | ndbmtd   | 1       |          |         |       |         |
| DataDir | /var/cdata | ndbmtd   | 2       |          |         |       |         |
| DataDir | /var/cdata | ndb_mgmd | 49      |          |         |       |         |
| datadir | /var/cdata | mysqld   | 50      |          |         |       |         |
| datadir | /var/cdata | mysqld   | 51      |          |         |       |         |
+---------+-----------+----------+---------+----------+---------+-------+---------+
5 rows in set (0.08 sec)
```

As you can see from the `get` command just shown, the attribute assignments were successful, and took effect on the process level.

Note

In MySQL Cluster Manager, configuration attribute names are not case-sensitive. See Case Sensitivity in String Searches for more information about case-sensitivity issues in MySQL Cluster Manager.

Similarly, you cannot reference multiple process IDs in a single attribute assignment, even if they are processes of the same type; the following command does *not* work:

```
mcm> set DataMemory:ndbd:1,2=750M mycluster;
ERROR 3 (00MGR): Illegal syntax
```

Instead, you would need to use the following command:

```
mcm> set DataMemory:ndbd:1=750M,DataMemory:ndbd:2=750M mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.70 sec)
```

(Of course, if these are the only two data nodes in `mycluster`, then the command `set DataMemory:ndbd=750M mycluster` also accomplishes the same task.)

Note

A few configuration attributes apply to connections between processes and so require you to refer to both processes in the course of setting them. In such cases, you must use a special process specification syntax; see Setting TCP Connection Attributes, for information about how this is done.

You also cannot set values for multiple attributes in a single attribute assignment; this means that the following commands do *not* work:

```
mcm> set UndoDataBuffer=32M,UndoIndexBuffer=8M:ndbd mycluster;
ERROR 3 (00MGR): Illegal syntax

mcm> set DataMemory,IndexMemory:ndbd=1G mycluster;
ERROR 3 (00MGR): Illegal syntax
```

However, if you write a complete and valid attribute assignment for each attribute whose value you wish to update, you can rewrite these two commands so that they execute successfully, as shown here:

```
mcm> set UndoDataBuffer:ndbd=32M,UndoIndexBuffer:ndbd=8M mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
```

```
+----------------------------------+
1 row in set (6.62 sec)

mcm> set DataMemory:ndbd=1G,IndexMemory:ndbd=1G mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.04 sec)
```

In fact, there is no reason that you cannot perform all four assignments in a single `set` command, using a list of four attribute assignments, like this:

```
mcm> set UndoDataBuffer:ndbd=32M,UndoIndexBuffer:ndbd=8M, \
          DataMemory:ndbd=1G, IndexMemory:ndbd=1G mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (6.24 sec)
```

However, it a good idea not to perform too many attribute assignments in any single `set` command, since this makes it more difficult to spot errors.

On Windows, when setting attributes whose values contain paths (such as `DataDir`), you must replace any backslash characters in the path with forward slashes. Suppose that you want to use `C:\temp\node50` for the `tmpdir` attribute of the `mysqld` process having node ID 50 in a MySQL NDB Cluster named `mycluster` that is running on Windows. The original value for this attribute can be seen using the appropriate `get` command:

```
mcm> get tmpdir mycluster;
+--------+----------------+----------+-----+----------+-----+-------+---------+
| Name   | Value          | Process1 | Id1 | Process2 | Id2 | Level | Comment |
+--------+----------------+----------+-----+----------+-----+-------+---------+
| tmpdir | c:\c50data\tmp | mysqld   | 50  |          |     |       |         |
+--------+----------------+----------+-----+----------+-----+-------+---------+
1 row in set (0.22 sec)
```

The correct `set` command to make the desired configuration change is shown here:

```
mcm> set tmpdir:mysqld:50=c:/temp/node50 mycluster;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (2.62 sec)
```

When you check the value using `get`—even though it was originally shown using backslashes—the forward slashes are used when displaying the new value:

```
mcm> get tmpdir mycluster;
+--------+----------------+----------+-----+----------+-----+-------+---------+
| Name   | Value          | Process1 | Id1 | Process2 | Id2 | Level | Comment |
+--------+----------------+----------+-----+----------+-----+-------+---------+
| tmpdir | c:/temp/node50 | mysqld   | 50  |          |     |       |         |
+--------+----------------+----------+-----+----------+-----+-------+---------+
1 row in set (0.22 sec)
```

However, if you try to use backslashes in the path when issuing the `set` command, the command fails:

```
mcm> set tmpdir:mysqld:4=c:\temp\4 mycluster;
Outfile disabled.
ERROR:
Unknown command '\4'.
ERROR 6014 (00MGR): Path name for parameter tmpdir must be absolute.
The value 'c:mp4' is illegal.
```

Performing a reset with a set command

You can reset a configuration attribute's value using a set command by putting a tilde (~) before the attribute name. For example, this command is equivalent to reset ndb_batch_size:mysqld:146 mycluster:

```
mcm> set ~ndb_batch_size:mysqld:146 mycluster
```

The notation allows you to run a set and a reset command together, which can potentially save the cluster from going through an extra rolling restart. For example:

```
mcm> set ndb_recv_thread_activation_threshold:mysqld:146=8,~ndb_batch_size:mysqld mycluster
```

Setting Attributes for mysqld nodes

When a dynamic variable is set, mcmd sends a SET GLOBAL statement to the mysqld to apply the value and saves the value to the mysqld configuration file, so that the value can be applied again the next time this mysqld process is restarted. Setting a variable which is not dynamic triggers an immediate restart.

When no data nodes are available, a set command that restarts a mysqld node without also restarting the data nodes is rejected. This is to make sure that any issues with the data nodes are handled first, so that the mysqld restart actually succeeds.

Setting TCP Connection Attributes

For a few attributes that apply only when using TCP connections (such as the SendBufferMemory and ReceiveBufferMemory attributes), it is necessary to use a modified syntax for attribute value assignments. In this case, the attribute assignment contains two process specifications, one for each process type or instance to which the setting applies, joined with a plus sign (+). For the following example, consider the cluster named mycluster2, consisting of the processes shown here:

```
mcm> list processes mycluster2;
+------+----------+-------------+
| Id   | Name     | Host        |
+------+----------+-------------+
| 49   | ndb_mgmd | grindval    |
| 1    | ndbd     | tonfisk     |
| 2    | ndbd     | flundra     |
| 50   | mysqld   | haj         |
| 51   | mysqld   | torsk       |
+------+----------+-------------+
5 rows in set (0.16 sec)
```

(See Section 5.6.3, “The list processes Command”, for more information about this command.)

TCP connection attributes are not shown in the output from the get command unless they have been set. This means that, prior to setting SendBufferMemory for the first time, you obtain an empty result if you try to retrieve its value, as shown here:

```
mcm> get SendBufferMemory mycluster2;
Empty set (0.18 sec)

mcm> get --include-defaults SendBufferMemory mycluster2;
Empty set (0.93 sec)
```

To set the SendBufferMemory to 4 MB for all TCP connections between data nodes and SQL nodes, you can use the command shown here:

```
mcm> set SendBufferMemory:ndbd+mysqld=4M mycluster2;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (6.44 sec)
```

If you check the attribute's value afterwards using get, you can see that the value is applied to all possible connections between each of the two ndbd processes and each of the two mysqld processes in mycluster2, thus there are four rows in the output:

```
mcm> get SendBufferMemory mycluster2;
+------------------+-------+----------+-----+----------+-----+---------+---------+
| Name             | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------------+-------+----------+-----+----------+-----+---------+---------+
| SendBufferMemory | 4M    | ndbd     | 2   | mysqld   | 4   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 2   | mysqld   | 5   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 3   | mysqld   | 4   | Process |         |
| SendBufferMemory | 4M    | ndbd     | 3   | mysqld   | 5   | Process |         |
+------------------+-------+----------+-----+----------+-----+---------+---------+
4 rows in set (1.63 sec)
```

To override this setting for only the connection between the data node with process ID 2 and the mysqld process (process ID 4), you can include the process ID in each of the two parts of the process specification, as shown here:

```
mcm> set SendBufferMemory:ndbd:2+mysqld:4=8M mycluster2;
+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
1 row in set (7.95 sec)
```

When you check the result using a get command, you can see that the new setting applies on the instance level, and only to the connection between processes having IDs 2 and 4; the process-level setting made previously still applies to the remaining 3 connections:

```
mcm> get SendBufferMemory mycluster2;
+------------------+-------+----------+-----+----------+-----+---------+---------+
| Name             | Value | Process1 | Id1 | Process2 | Id2 | Level   | Comment |
+------------------+-------+----------+-----+----------+-----+---------+---------+
| SendBufferMemory | 8M    | ndbd     | 2   | mysqld   | 50  |         |         |
| SendBufferMemory | 4M    | ndbd     | 2   | mysqld   | 51  | Process |         |
| SendBufferMemory | 4M    | ndbd     | 3   | mysqld   | 50  | Process |         |
| SendBufferMemory | 4M    | ndbd     | 3   | mysqld   | 51  | Process |         |
+------------------+-------+----------+-----+----------+-----+---------+---------+
4 rows in set (0.24 sec)
```

You cannot set a connection attribute on the process level in one part of the process specification (that is, for one end of the connection) and on the instance level in the other. Attempting to do so fails with an error, as shown here:

```
mcm> set SendBufferMemory:ndbd+mysqld:4=2M mycluster2;
ERROR 3 (00MGR): Illegal syntax
mcm> set SendBufferMemory:ndbd:2+mysqld=2M mycluster2;
ERROR 3 (00MGR): Illegal syntax
```

Setting up mysqld connection pooling

Enabling connection pooling for mysqld can be done by setting the ndb-cluster-connection-pool attribute to the desired number of connections, but also requires an extra step in creating the cluster.

Because the mysqld process attempts to make multiple connections to the cluster when connection pooling is enabled, the cluster must be configured with “spare” or “empty” connections. You can do this by adding (otherwise) unused ndbapi entries in the process_host list used in the create cluster command, as shown here:

```
mcm> create cluster -P mypackage
   > -R ndb_mgmd@10.100.10.97,ndbd@10.100.10.98,ndbd@10.100.10.99, \
        mysqld@10.100.10.100,ndbapi@10.100.10.100, \
        ndbapi@10.100.10.100,ndbapi@10.100.10.100
   > mycluster;
```

```
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster created successfully |
+-----------------------------+
1 row in set (6.58 sec)
```

After this, you can use a set command like this one to set the size of the connection pool according to the number of excess connections available in the config.ini file:

```
mcm> set ndb_cluster_connection_pool:mysqld=4;
```

Note

Trying to set the user attribute for a mysqld process is not supported, and results in a warning being written to the MySQL Cluster Manager log.

Setting Up Encryption

NDB Cluster 8.0.31 and later supports transparent data encryption (TDE) for user data stored in NDB tables (see File System Encryption for NDB Cluster); this is also supported by MySQL Cluster Manager 8.0.31 and later. File system encryption is enabled on the data nodes by setting the EncryptedFileSystem configuration parameter equal to 1 on all data nodes. (Disable encryption by setting the parameter to 0.)

Encrypting and decrypting data require that the data nodes have the encryption password, which must be stored in a file readable by the data node processes. You can supply this to the data nodes with a set command that uses the filesystem-password-file option (introduced in MySQL Cluster Manager 8.0.31). This must be done before setting EncryptedFileSystem = 1, as shown later in this section.

The following example makes use of the cluster mycluster running as shown in the output of this show status command in the mcm client:

```
mcm> show status -r mycluster;

+--------+----------+--------+---------+-----------+-----------+
| NodeId | Process  | Host   | Status  | Nodegroup | Package   |
+--------+----------+--------+---------+-----------+-----------+
| 145    | ndb_mgmd | myndb3 | running |           | mypackage |
| 1      | ndbmtd   | myndb1 | running | 0         | mypackage |
| 2      | ndbmtd   | myndb2 | running | 0         | mypackage |
| 146    | mysqld   | myndb3 | running |           | mypackage |
| 147    | mysqld   | myndb4 | running |           | mypackage |
| 148    | ndbapi   | *      | added   |           |           |
+--------+----------+--------+---------+-----------+-----------+
```

Setting the password directly from the command line in the mcm client is not supported. Using a file on disk instead helps protect against unprivileged user access, provided that file system access rights are sufficiently strict. (On Linux and similar platforms, this file must have its permissions set to 0600.) This file should contain only the encryption password, which follows the same rules as passwords for encrypted NDB backups; see Using The NDB Cluster Management Client to Create a Backup, for more information.

Assuming that the password file exists and has the proper permissions, you can supply the password to the data nodes using the following set command:

```
mcm> set filesystem-password-file:ndbmtd=/opt/mcm_data/my.pwd mycluster;

+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
```

Once the path to the password file has been set, you can enable encryption, like this:

```
mcm> set EncryptedFileSystem:ndbmtd=1 mycluster;

+----------------------------------+
| Command result                   |
+----------------------------------+
| Cluster reconfigured successfully |
+----------------------------------+
```

If the encryption password file has not been set, the set command just shown is rejected with an error.

You can verify that encryption is enabled using a get command similar to this one:

```
mcm> get -d filesystem-pass*,encrypt* mycluster;

+--------------------------+---------------------------+----------+---------+----------+---------+-----
| Name                     | Value                     | Process1 | NodeId1 | Process2 | NodeId2 | Leve
+--------------------------+---------------------------+----------+---------+----------+---------+-----
| EncryptedFileSystem      | 1                         | ndbmtd   | 1       |          |         | Proc
| filesystem-password-file | /opt/mcm_data/my.pwd      | ndbmtd   | 1       |          |         | Proc
| EncryptedFileSystem      | 1                         | ndbmtd   | 2       |          |         | Proc
| filesystem-password-file | /opt/mcm_data/my.pwd      | ndbmtd   | 2       |          |         | Proc
+--------------------------+---------------------------+----------+---------+----------+---------+-----
```

You can also verify, outside of MySQL Cluster Manager or even a running NDB Cluster, that cluster data files have been encrypted using the ndbxfrm utility supplied with NDB Cluster, similarly to what is shown here:

```
$> ndbxfrm -i /home/mcm/clusters/mycluster/1/data/ndb_1_fs/LCP/0/T10F0.Data

File=/home/mcm/clusters/mycluster/1/data/ndb_1_fs/LCP/0/T10F0.Data, compression=no, encryption=yes
```

You can rotate file system passwords by changing the existing file (or setting a new file), then issuing set --retry EncryptedFilesystem:ndbmtd=1 to trigger an initial rolling restart. Alternatively, you can use stop process followed by start process --initial to replace the password used by each data node process, one at a time.

Overriding the Default Restart Type

Warning

The overriding of default restart type using the --restart option may cause unintended consequences. It should only be performed under guidance by the support personnel from Oracle.

When setting the MaxNoOfExecutionThreads or ThreadConfig parameter for data nodes, their default restart type (which is SI, System Initial) could be overridden with the --restart option to become NI (Node Initial) or N (Node). This can be used to change the configuration parameter without actually reconfiguring the number of LDM threads. For example:

```
mcm> set --restart=N ThreadConfig:ndbmtd='main={count=1},tc={count=0},ldm={count=4},io={count=1},
  rep={count=1},recv={count=1},send={count=0}', MaxNoOfExecutionThreads:ndbmtd=10 mycluster;

Cluster reconfigured successfully
```

Use of the option requires the following:

- At least one ndb_mgmd node is running.
- All data nodes and ndb_mgmd nodes are running for --restart=NI.
- The cluster remains alive while restarting the data nodes (i.e., there are at least two data nodes running in each nodegroup) for --restart=N.
- The set statement does not contain any additional parameters that only affect a ndb_mgmd or mysqld node.

5.5.4 The show variables Command

```
show variables
```

The command displays the relevant settings for the current agent, including the cipher list and TLS version.

```
mcm> show variables;
+--------------------------+--------------------------+
| Variable_name            | Value                    |
+--------------------------+--------------------------+
| auto_increment_increment | 1                        |
| character_set_client     | latin1                   |
| ssl_cipher_list          | LIST-OF-SUPPORTED-CIPHERS |
| tls_version_list         | TLSv1.2,TLSv1.3          |
+--------------------------+--------------------------+
4 rows in set (0.00 sec)
```

5.6 MySQL Cluster Manager Process Commands

This section contains information about MySQL Cluster Manager client commands used to start and stop MySQL NDB Cluster processes, and to determine which processes are currently running.

MySQL Cluster Manager, ndb_mgm, and starting or stopping processes. For a MySQL NDB Cluster under MySQL Cluster Manager control, it is recommended *not* to use the ndb_mgm command-line client that comes with the MySQL NDB Cluster distribution to perform operations that involve starting or stopping nodes. These include but are not limited to the following ndb_mgm client commands (Commands in the NDB Cluster Management Client):

- START
- STOP
- RESTART
- SHUTDOWN

5.6.1 The add process Command

```
add process {--processhosts=|-R }process_host_list
    [--set=attribute_assignment_list] [--verbose | -v] [--sequential-restart] cluster_name

process_host_list:
    process_name[:node_id]@host[,process_name@host[,...]]

process_name:
    {ndb_mgmd|ndbd|ndbmtd|mysqld|ndbapi}

attribute_assignment_list:
    attribute_assignment[,attribute_assignment][,...]

attribute_assignment:
    attribute_name:process_name[=value]
```

This command adds to an existing cluster one or more processes, which are specified using a *process_host_list* with the --processhosts option, the format of which is the same as that used with the create cluster command. Any hosts referenced in the list must be members of the site to which the cluster belongs. In addition, all hosts must be resolvable.

For example, the following add process command adds two mysqld processes on hosts tonfisk and flundra to the cluster named mycluster:

```
mcm> add process --processhosts=mysqld@tonfisk,mysqld@flundra mycluster;
+-----------------------------+
```

```
| Command result               |
+------------------------------+
| Processes added successfully |
+------------------------------+
1 row in set (2 min 10.39 sec)
```

With the `--verbose` option, the command shows an updated process list, after the new processes have been added:

```
mcm> add process --processhosts=ndbmtd@tonfisk,ndbmtd@flundra --verbose mycluster;
+--------+----------+---------+
| NodeId | Name     | Host    |
+--------+----------+---------+
| 49     | ndb_mgmd | tonfisk |
| 53     | ndb_mgmd | flundra |
| 1      | ndbmtd   | tonfisk |
| 2      | ndbmtd   | flundra |
| 3      | ndbmtd   | tonfisk |
| 4      | ndbmtd   | flundra |
| 50     | mysqld   | tonfisk |
| 51     | mysqld   | flundra |
| 52     | ndbapi   | *       |
+--------+----------+---------+
9 rows in set (2 min 7.57 sec)
```

You can also manually assign a node ID to the new process you are adding to the cluster by adding "`:node_ID.`" after the *`process_name`*. You are still recommended to follow the best practice of reserving node ID 1 to 144 for data nodes. The following command adds two `ndbd` processes with node IDs 10 and 11 on hosts `tonfisk` and `flundra`, respectively, to `mycluster`:

```
mcm> add process --processhosts=ndbd:10@tonfisk,ndbd:11@flundra mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Processes added successfully |
+------------------------------+
1 row in set (2 min 13.40 sec)
```

If the cluster is not running when you run the `add process` command, it is recommended that you start all the new processes added by this command together using the `start process --added` command or start them together with the whole cluster using the `start cluster` command—besides starting the nodes, either of the two commands also initializes the added nodes and causes new cluster nodegroups to be formed by issuing a `CREATE NODEGROUP` command to the cluster. If the added nodes are started with `start process --initial` instead, you are then required to run `CREATE NODEGROUP` manually via the `ndb_mgm` client.

If the cluster is running when you run the `add process` command, a rolling restart for the cluster is performed at the end of the `add process` command. Use the `--sequential-restart` option to make the rolling restart a sequential one.

Adding Free Processes

Using the `add process` command, you can add unmanaged `mysqld` processes, or `ndbapi` slots for `ndbapi` applications such as `ndb_restore`. To add an unmanaged `mysqld` process, prefix the hostname with the wildcard * (asterisk character):

```
mcm> add process --processhosts=mysqld@*tonfisk,mysqld@*flundra mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Processes added successfully |
+------------------------------+
1 row in set (2 min 3.14 sec)
```

To allow the unmanaged `mysqld` nodes to connect from any host, use the wildcard `*` (asterisk character) in place of the hostname or IP address:

```
mcm> add process --processhosts=mysqld@*,mysqld@* mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Processes added successfully |
+-----------------------------+
1 row in set (2 min 3.14 sec)
```

The same applies to ndbapi slots for ndbapi applications such as ndb_restore: prefix the hostname with the wildcard character to limit connectivity to a specific host, or use only a wildcard, without hostname, to allow ndbapi applications from any host:

```
mcm> add process --processhosts=ndbapi@*tonfisk,ndbapi@* mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Processes added successfully |
+-----------------------------+
1 row in set (2 min 8.13 sec)
```

Because “free” processes are not managed by MySQL Cluster Manager, there is no need to run the start process --added command after they have been successfully added to the cluster.

Using add process to Simplify create cluster Commands

Processes added before the cluster is started for the first time are started with the cluster. This makes it possible to use this command to break down what would otherwise be very long create cluster commands. Consider the following set of commands that creates and then starts a cluster named mycluster:

```
create cluster --processhosts=ndb_mgmd@host1,ndbd@host1,ndbd@host2, \
  mysqld@host3,mysqld@host4 mycluster;
start cluster mycluster;
```

The long create cluster command can be divided into a shorter (and more manageable) version of itself, plus several add process commands. This set of commands performs the same task as the previous set, creating mycluster with exactly the same processes and hosts as before, and then starting it:

```
create cluster --processhosts=ndb_mgmd@host1 mycluster;
add process --processhosts=ndbd@host1,ndbd@host2 mycluster;
add process --processhosts=mysqld@host3,mysqld@host4 mycluster;
start cluster mycluster;
```

Notice that a process that is added to a cluster that was created using create cluster --import and before the import takes place is added with status import, which means it cannot be started or stopped using start process or stop process before an import has taken place.

Configuring a New Process when Adding it

A newly added process inherits its configuration attribute settings from those in effect for its process type on the parent cluster, or assume the default settings for that process type if none apply. Existing attribute settings in the cluster must have process-level scope to be inherited by new processes added later; instance-level settings set for existing process instances prior to adding any new ones do not apply to any of the added processes. (See Configuration attributes, for more information about the scope of attribute settings.)

Inherited attribute settings can be overridden when adding processes; to do this, use the add process command's --set option. This option takes as its argument an attribute assignment list similar in format to that used with the get and set commands. Suppose that the current ndbd process-level setting in the cluster named mycluster for the DataDir attribute is /home/users/ndb/cluster-data, but you wish to add two new ndbd processes that use /tmp/cluster/data instead. You can do this using the following command:

```
mcm> add process --set=ndbd:DataDir=/tmp/cluster/data
   > --processhosts=mysqld@tonfisk,mysqld@flundra
   > mycluster;
```

Note

Unlike the way you use the `set` command, an equal sign (`=`) immediately following the `--set` option is required.

When setting attributes this way, which involves specifying paths for processes running on Windows, you must replace any backslashes (`\`) used with forward slashes (`/`), just as with the `set` command. See Setting Attributes Containing Paths on Windows [129], for more information.

After a process has been added using `add process`, you can also use the `set` command to modify its configuration attribute settings (or specify additional ones) as you would with any other cluster process being managed with MySQL Cluster Manager.

Note

When IPv6-enabled Windows systems are used as MySQL NDB Cluster hosts under MySQL Cluster Manager, you must reference these hosts using IPv4 addresses. Otherwise, MySQL Cluster Manager is unable to connect to the agent processes on those hosts. See Section 6.1, “MySQL Cluster Manager Usage and Design Limitations”.

5.6.2 The `change process` Command

```
change process [--sequential-restart] old_proc_type[:proc-id]=new_proc_type cluster_name

old_proc_type | new_proc_type:
    {ndbd|ndbmtd}
```

This command is used to change the process type for a given MySQL NDB Cluster process or group of MySQL NDB Cluster processes from one process type (*`old-process-type`*) to another process type (*`new-process-type`*).

Currently, the only two process types available for use with this command are `ndbd` and `ndbmtd`. This means that `change process` can be used to change the data node process running on one or more data nodes from the single-threaded data node daemon (`ndbd`) to the multithreaded data node daemon (`ndbmtd`) or vice versa.

By default, `change process` affects all data nodes running the *`old-process-type`*. By specifying an optional *`process_id`*, its action can be restricted to the data node having that process ID.

Suppose you have a cluster that is named `mycluster` and has two data nodes using `ndbd` processes, as reflected in the output of the following `show status` command:

```
mcm> show status --process mycluster;
+--------+----------+----------+----------+-----------+
| NodeId | Process  | Host     | Status   | Nodegroup |
+--------+----------+----------+----------+-----------+
| 49     | ndb_mgmd | flundra  | running  |           |
| 1      | ndbd     | tonfisk  | running  | n/a       |
| 2      | ndbd     | grindval | running  | n/a       |
| 50     | mysqld   | haj      | running  |           |
| 51     | mysqld   | torsk    | running  |           |
| 52     | ndbapi   | *        | running  |           |
+--------+----------+----------+----------+-----------+
6 rows in set (0.06 sec)
```

To change both data nodes to so that they use multithreaded (`ndbmtd`) processes, issue the command shown here, without any *`process_id`* specifier:

```
mcm> change process ndbd=ndbmtd mycluster;
```

```
+-----------------------------+
| Command result              |
+-----------------------------+
| Process changed successfully |
+-----------------------------+
1 row in set (2 min 17.51 sec)
```

After the command has executed, you can verify that both data nodes are now using ndbmtd by checking the output of the appropriate show status command, as shown here:

```
mcm> show status --process mycluster;
+--------+----------+----------+----------+-----------+
| NodeId | Process  | Host     | Status   | Nodegroup |
+--------+----------+----------+----------+-----------+
| 49     | ndb_mgmd | flundra  | running  |           |
| 1      | ndbmtd   | tonfisk  | running  | n/a       |
| 2      | ndbmtd   | grindval | running  | n/a       |
| 50     | mysqld   | haj      | running  |           |
| 51     | mysqld   | torsk    | running  |           |
| 52     | ndbapi   | *        | running  |           |
+--------+----------+----------+----------+-----------+
6 rows in set (0.09 sec)
```

A rolling restart for the cluster is performed at the end of the change process command. Use the --sequential-restart option to make the rolling restart a sequential one.

Note

The change process command can be used whether or not the cluster or the data node or data nodes to be changed are running. However, the command executes much more quickly if the data node or data nodes to be changed are not running. The next set of examples illustrates this.

It is possible (and sometimes desirable) to use ndbd and ndbmtd data node processes concurrently; thus, it is also possible using the change process command to change a single data node process from single-threaded to multithreaded, or from multithreaded to single-threaded. To do this, you must specify the data node process using its process ID.

First, we stop the cluster and verify that all processes are no longer running, as shown here:

```
mcm> stop cluster mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster stopped successfully |
+-----------------------------+
1 row in set (22.93 sec)

mcm> show status --process mycluster;
+--------+----------+----------+----------+-----------+
| NodeId | Process  | Host     | Status   | Nodegroup |
+--------+----------+----------+----------+-----------+
| 49     | ndb_mgmd | flundra  | stopped  |           |
| 1      | ndbmtd   | tonfisk  | stopped  | n/a       |
| 2      | ndbmtd   | grindval | stopped  | n/a       |
| 50     | mysqld   | haj      | stopped  |           |
| 51     | mysqld   | torsk    | stopped  |           |
| 52     | ndbapi   | *        | stopped  |           |
+--------+----------+----------+----------+-----------+
6 rows in set (0.05 sec)
```

The following command changes only the node having the process ID 2 from using the multithreaded data node daemon to the single-threaded version:

```
mcm> change process ndbmtd:2=ndbd mycluster;
+-----------------------------+
| Command result              |
```

```
+-----------------------------+
| Process changed successfully |
+-----------------------------+
1 row in set (6.52 sec)
```

As you can see, `change process` operates much more quickly when the process to be changed is not running. As before, you can verify that the command succeeded using `show status`:

```
mcm> show status --process mycluster;
+--------+----------+----------+----------+-----------+
| NodeId | Process  | Host     | Status   | Nodegroup |
+--------+----------+----------+----------+-----------+
| 49     | ndb_mgmd | flundra  | stopped  |           |
| 1      | ndbmtd   | tonfisk  | stopped  | n/a       |
| 2      | ndbd     | grindval | stopped  | n/a       |
| 50     | mysqld   | haj      | stopped  |           |
| 51     | mysqld   | torsk    | stopped  |           |
| 52     | ndbapi   | *        | stopped  |           |
+--------+----------+----------+----------+-----------+
6 rows in set (0.07 sec)
```

To complete the example, we start the cluster again, using `start cluster`, then change node number 2 back from `ndbd` (single-threaded) to `ndbmtd` (multithreaded) using `change process`, then verify the change using `show status`:

```
mcm> start cluster mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Cluster started successfully |
+-----------------------------+
1 row in set (36.43 sec)

mcm> change process ndbd:2=ndbmtd mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Process changed successfully |
+-----------------------------+
1 row in set (2 min 10.41 sec)

mcm> show status --process mycluster;
+--------+----------+----------+----------+-----------+
| NodeId | Process  | Host     | Status   | Nodegroup |
+--------+----------+----------+----------+-----------+
| 49     | ndb_mgmd | flundra  | running  |           |
| 1      | ndbmtd   | tonfisk  | running  | n/a       |
| 2      | ndbmtd   | grindval | running  | n/a       |
| 50     | mysqld   | haj      | running  |           |
| 51     | mysqld   | torsk    | running  |           |
| 52     | ndbapi   | *        | running  |           |
+--------+----------+----------+----------+-----------+
6 rows in set (0.11 sec)
```

You can see that it can require much less time to stop the cluster, change a data node process, and then start the cluster again than it is to change the process while the cluster is running. However, if you do this, the cluster is not available while it is stopped.

As noted previously, `change process` works only with `ndbd` and `ndbmtd` processes; attempting to use any other process type causes the command to fail with an error, as shown here:

```
mcm> change process ndb_mgmd=mysqld mycluster;
ERROR 7009 (00MGR): Processes ndb_mgmd and mysqld are not interchangeable in this package
mcm> change process ndbd=mysqld mycluster;
ERROR 7009 (00MGR): Processes ndbd and mysqld are not interchangeable in this package
```

5.6.3 The `list processes` Command

```
list processes cluster_name
```

This command displays all processes making up a given cluster. The following example demonstrates how to list all processes that are part of the cluster named mycluster:

```
mcm> list processes mycluster;
+--------+----------+----------+
| NodeId | Name     | Host     |
+--------+----------+----------+
| 49     | ndb_mgmd | flundra  |
| 1      | ndbd     | tonfisk  |
| 2      | ndbd     | grindval |
| 50     | mysqld   | haj      |
| 51     | mysqld   | torsk    |
| 52     | ndbapi   | *        |
+--------+----------+----------+
6 rows in set (0.03 sec)
```

The *cluster_name* argument is required. If this argument is omitted, the command fails with an error, as shown here:

```
mcm> list processes;
ERROR 6 (00MGR): Illegal number of operands
```

5.6.4 The start process Command

```
start process {[--initial|-i] nodespec | --added} cluster_name

nodespec:
    {nodetype | process_id_list}
process_id_list:
    process_id[, process_id[, ...]]
```

This command starts the MySQL NDB Cluster processes specified by *nodespec* in the cluster named *cluster_name*. The status of the processes to be started, as shown by show status --process, must be added, stopped, or failed (only if the failed process has exited properly can it be restarted with the command).

This example demonstrates how to start the process having the process ID 1 belonging to the cluster mycluster:

```
mcm> start process 1 mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Process started successfully |
+------------------------------+
1 row in set (13.93 sec)
```

You can obtain process IDs for all processes in a given cluster using show status --process or list processes. These are the same as the node IDs for these processes as shown in the output of other mcm client commands such as get or in the output of ndb_mgm -e "show" (see ndb_mgm — The NDB Cluster Management Client).

Instead of a single node, you can also specify the type of nodes or a list of nodes to start :

```
mcm> start process mysqld mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Process started successfully |
+------------------------------+
1 row in set (15.72 sec)
```

```
mcm> start process 146,147 mycluster;
+------------------------------+
| Command result               |
+------------------------------+
| Process started successfully |
+------------------------------+
```

```
1 row in set (3.92 sec)
```

The following requirements must be fulfilled when you specify a list of nodes to start, or the command will fail:

- All nodes in the list must be of the same process type.
- The list should not include all managed nodes of the cluster.
- After the command finishes running, there should be at least 1 running data node per node group, and more than half of all data nodes in the cluster should be running.
- The general rules on process dependencies are satisfied (for example, a `mysqld` node depends on some data nodes running, a data node depends on some management nodes running, and so on).
- `StartPartitionedTimeout > 0` is needed to allow a single data node out of a total of two to be started alone.

When the `--initial` option (short form: `-i`) is used, the following happens:

- For a data node, MySQL Cluster Manager starts it with the `--initial` option, causing the data node to rebuild its file system.
- For an SQL node, MySQL Cluster Manager rebuilds the `mysqld` data directory with the `mysqld --initialize-insecure` command for MySQL 9.6, 8.4, and 8.0 . The node's data directory must be empty, or the reinitialization will not be attempted.

Invoking this command with the `--added` option rather than with a *`nodespec`* starts all nodes that were added previously to the cluster using `add process` but not yet started. For the added data and **mysqld** nodes, the use of the `--added` option also implies the use of the `--initial` option, meaning that `mcmd` will attempt to initialize the added nodes (see description for the `--initial` option above). Also, when the `--added` option is used, once all the added nodes are running, a `CREATE NODEGROUP` command is issued to the management node for the creation of new nodegroups.

You cannot use this command to start a `mysqld` process in a stopped or unavailable cluster—trying to do so will cause an error. This applies, for example, to the case in which a cluster has been created for a cluster import, but the import is not yet completed (see Section 5.4.1, “The `create cluster` Command”, and Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”).

5.6.5 The `stop process` Command

```
stop process nodespec cluster_name

nodespec:
    {nodetype | process_id_list}
process_id_list:
    process_id[, process_id[, ...]]
```

This command stops the MySQL NDB Cluster processes specified by *`nodespec`* in the cluster named *`cluster_name`*. The status of the processes to be stopped, as shown by `show status --process`, must be `running`.

Suppose that the process ID of a data node in the cluster named `mycluster` is `3`. Then this data node can be stopped as shown here:

```
mcm> stop process 3 mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Process stopped successfully |
+-----------------------------+
```

```
1 row in set (33.07 sec)
```

Instead of a single node, you can also specify the type of nodes or a list of nodes (must be of the same type) to be stopped. For example:

```
mcm> stop process mysqld mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Process stopped successfully |
+-----------------------------+
1 row in set (15.70 sec)
```

```
mcm> stop process 146,147 mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Process stopped successfully |
+-----------------------------+
1 row in set (3.82 sec)
```

You can use show status --process or list processes to obtain process IDs for all processes in a given cluster.

In the event of a disk failure where MySQL Cluster Manager loses its manager directory (including its repository), the agent is able to recover information from other agents, but it does not actually control processes any longer, although it can detect them. This is due to the fact that the MySQL Cluster Manager agent cannot access the PID files. In this case, stop process no longer works, and you must kill such processes manually. Keep in mind that, if StopOnError is 0, the MySQL Cluster Manager agent restarts the data node process automatically; if StopOnError is 1 (the default), then you must execute the start process command manually.

Note

For release 9.3.0 and later: For *bootstrapped* clusters, StopOnError is 1 by default.

This command does not work with processes in a cluster created for import where the import has not yet actually been completed. See Section 5.4.1, “The create cluster Command”, and Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”, for more information.

5.6.6 The update process Command

```
update process [--remove-angel] --pid=os_pid process_id cluster_name
```

This command updates the status of the MySQL NDB Cluster process having the process ID *process_id* in the cluster named *cluster_name* when the status of the process is no longer reflected correctly in the output of the show status --process command. This typically happens in the following cases:

- The process is a data node configured with StopOnError=true, so that it would not be automatically restarted by mcmd after it has stopped. Instead of using the start process command to restart the process, a user might have restarted the process manually, which would have restored the process but left mcmd without the knowledge of the restore. An update process is then needed to restore the control of the process by mcmd.
- The process is a node that has been stopped by mcmd but, for some reasons, its PID remains valid with the operating system. In some cases, the process might even be running again, without mcmd knowing or being able to control it.
- mcmd cannot connect to a mysqld node due to various reasons (for example, there are already too many connections to the node); process status for the node becomes failed, while the PID file continues to exist.

- When a start process command for a mysqld node times out, mcmd looses control of the node. After fixing the issue on the mysqld node, run update process to restore control of the node by mcmd.

The command works by importing the process into the control of mcmd again. Checks performed on a process by mcmd during a cluster import are performed for the update process command. Both the process's ID in the cluster (*process_id*) and its PID on the operating system (specified with the --pid option) are required. Suppose that the process ID of a data node in the cluster named mycluster is 3 and its PID on the operating system is 9846, the data node can be updated as shown here:

```
mcm> update process --pid=9846 3 mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Process updated successfully |
+-----------------------------+
1 row in set (33.07 sec)
```

For a data node or an SQL node, the command only works if there is at least 1 replica per nodegroup running.

update process supports a --remove-angel option, which should be used when updating data nodes: it kills any running angel process for a data node and updates its PID file prior to the actual update; those steps are necessary for the update process.

Note

Some options, when used to start an applicable NDB node, are not preserved after the update process:

- --initial-start
- --nowait-nodes
- --logbuffer-size

. The same applies now to the

5.6.7 The remove process Command

```
remove process [--removedirs] process_id_list cluster_name

process_id_list:
    process_id[, process_id[, ...]]
```

This command removes permanently the processes in the *process_id_list* from the cluster named *cluster_name*. It provides a means to scale down a cluster offline.

If the --removedirs option is used, all data for the specified processes will be deleted.

The following restrictions apply when using this command:

1. The cluster must be in the status of created or stopped.
2. The processes to be removed must be in the status of stopped, added, or import.
3. The command cannot remove all processes from a cluster in the created status; at least one process must be left.
4. The command cannot remove all process of the same type from a cluster in the stopped status; at least one process must be left in the cluster for each type of nodes (management, data, and API).
5. The command cannot remove a data node that is in the stopped status if it is already a member of a node group (i.e., if it has ever been started and was fully functional).

You can use the `show status --process` or `list processes` command to obtain the process IDs for all the processes in a given cluster:

```
mcm> show status --process mycluster;
+--------+----------+---------+--------+-----------+-----------+
| NodeId | Process  | Host    | Status | Nodegroup | Package   |
+--------+----------+---------+--------+-----------+-----------+
| 49     | ndb_mgmd | flundra | added  |           | mypackage |
| 1      | ndbmtd   | flundra | added  | n/a       | mypackage |
| 2      | ndbmtd   | flundra | added  | n/a       | mypackage |
| 50     | mysqld   | flundra | added  |           | mypackage |
| 51     | mysqld   | flundra | added  |           | mypackage |
| 52     | ndbapi   | *       | added  |           |           |
| 53     | ndbapi   | *       | added  |           |           |
+--------+----------+---------+--------+-----------+-----------+
7 rows in set (0.03 sec)
```

The process IDs are the same as the node IDs for the processes shown in the output of the above or some other `mcm` client commands, or in the output of the `ndb_mgm -e "show"` command (see ndb_mgm — The NDB Cluster Management Client). In the above example, the SQL node with the process ID `50` in `mycluster` can be removed by the following command:

```
mcm> remove process 50 mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Process removed successfully |
+-----------------------------+
1 row in set (0.48 sec)
```

And in this case, since the cluster was never started, we may also remove both data nodes:

```
mcm> remove process 1,2 mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Process removed successfully |
+-----------------------------+
1 row in set (0.40 sec)
```

5.7 MySQL Cluster Manager TLS Connection Commands

This section contains information about MySQL Cluster Manager client commands relating to Section 4.11, “Using TLS Connections for NDB Clusters”.

5.7.1 The `create certs` Command

```
create certs [--ca|--keys|--renew] [--added] cluster_name
```

The command creates all certificate authorities (CAs), keys, and certificate files needed for a cluster to use TLS connections on all hosts in the site.

The `--ca` option limits the command to only create the CA key and certificate

The `--keys` option limits the command to only create the keys and certificates for all nodes.

The `--added` option limits the command to only create CAs and API certificates for recently added hosts and nodes. A CA and key must be present on at least one host in the site for the `--added` option to work.

The `--renew` option renews the keys and certificates for all nodes.

The host with the client connection runs ndb_sign_keys to create the CA in the cluster's default certificate directory, `<mcm_data>/clusters/<cluster_name>/certs`, and the CA is distributed

over the site. To allow secure CA distribution across the MCM site, the mcmd agent connections must be encrypted (see Section 4.10, “Using Encrypted Connections for MySQL Cluster Manager Agents and Clients” for details).

Certificates are created for every host using the available CA. The CA and certificates are created using the default CA and certificate file names defined on the NDB Cluster. The certificates are created in two locations:

- An agent, and any NDB tools that it spawns, uses an API certificate from the cluster's default certificate directory, which must be present on all hosts in the site.
- A cluster process uses the certificate from the process as specified by the ndb_mgm --ndb-tls-search-path option.

The command creates both sets of certificates—the API certificate in the cluster's default certificate folder, and a single certificate/key pair for each process in the process' certificate directories. If multiple processes use the same certificate type and share the same certificate directory on the same file system, only a single instance of the certificate will be created. Creation of subsequent certificates of the same type at the same location is skipped.

The command fails if any of the following conditions is true:

- A CA already exists when creating the CA, unless the --renew option is used (in which case the command fails if a CA does not already exist).
- A certificate already exists when creating the certificate, unless the --renew option is used (in which case the command fails if a certificate does not already exist). .
- When the --renew is used together with the --added or the --ca option.
- All hosts are not present.

Limitations: The following limitations apply for the command:

- If multiple hosts share the same network-mounted certificate directory, the certificates embedded hostnames may be incorrect.
- No actions are taken on the certificate folders or certificate files created by the command on delete cluster --removedirs.
- For now, only keys can be renewed. Always use the --keys option with the --renew option.

5.7.2 The list certs Command

```
list certs [--active|-A] [--all|-a] [--retired|-r] cluster_name
```

The command lists the files created by the create certs command for the MySQL NDB Cluster named *cluster_name*. By default, it shows for each host the number of active, retired, and pending certificates, as well as the total number of certificates:

```
mcm> list certs mycluster;
+---------+--------+---------+---------+-------+
| Host    | Active | Retired | Pending | Total |
+---------+--------+---------+---------+-------+
| tonfisk | 8      | 1       | 0       | 9     |
| flundra | 6      | 1       | 0       | 7     |
+---------+--------+---------+---------+-------+
```

With the --active option, the command shows the active certificates found on each host, listing the last-modification timestamp, full path, and filename for each certificate:

```
mcm> list certs --active mycluster;
+---------+----------------------+-------------------------------------------------------------------------+
```

```
| Host    | Timestamp            | File                                                                 |
+---------+----------------------+----------------------------------------------------------------------+
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-cert              |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-private-key       |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-cert                  |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-private-key           |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-cert            |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-private-key     |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-mgm-server-cert           |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-mgm-server-private-key    |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-cert              |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-private-key       |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-private-key           |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-cert                  |
| flundra | 2024-09-25 11:59:13Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-private-key     |
| flundra | 2024-09-25 11:59:13Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-cert            |
+---------+----------------------+----------------------------------------------------------------------+
```

With the `--retired` option, the command shows the same information for the retired certificates found on each host:

```
mcm> list certs --retired mycluster;
+---------+----------------------+----------------------------------------------------------------------+
| Host    | Timestamp            | File                                                                 |
+---------+----------------------+----------------------------------------------------------------------+
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-retired-cert    |
| flundra | 2024-10-23 14:05:57Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-retired-cert    |
+---------+----------------------+----------------------------------------------------------------------+
```

With the `--all` option, the command shows the active, retired, and pending certificates found on each host:

```
mcm> list certs --all mycluster;
+---------+----------------------+----------------------------------------------------------------------+
| Host    | Timestamp            | File                                                                 |
+---------+----------------------+----------------------------------------------------------------------+
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-cert              |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-private-key       |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-cert                  |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-private-key           |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-cert            |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-private-key     |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-retired-cert    |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-mgm-server-cert           |
| tonfisk | 2024-11-05 13:05:06Z | /opt/mcm_data/clusters/mycluster/certs/ndb-mgm-server-private-key    |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-cert              |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/NDB-Cluster-private-key       |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-cert                  |
| flundra | 2024-09-25 11:59:11Z | /opt/mcm_data/clusters/mycluster/certs/ndb-api-private-key           |
| flundra | 2024-09-25 11:59:13Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-cert            |
| flundra | 2024-09-25 11:59:13Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-private-key     |
| flundra | 2024-10-23 14:05:57Z | /opt/mcm_data/clusters/mycluster/certs/ndb-data-node-retired-cert    |
+---------+----------------------+----------------------------------------------------------------------+
```

5.8 MySQL Cluster Manager Backup and Restore Commands

This section contains information about MySQL Cluster Manager client commands relating to backing up a MySQL NDB Cluster and restoring it from backup.

5.8.1 The `abort backup` Command

```
abort backup --backupid=backup_id cluster_name
```

This command aborts a backup of cluster *`cluster_name`* having the ID *`backup_id`* specified with the `--backupid` option. You can obtain a list of backups and their IDs known to this MySQL Cluster Manager instance using the `list backups`. If the backup is not actually in progress, the command has no effect.

5.8.2 The backup cluster Command

```
backup cluster [{--backupid=|-I }backup_id]
    [{--snapshotstart|-S} | {--snapshotend|-E}]
    [{--waitstarted|-w} | {--waitcompleted|-W}]
    [{--password-file=|-F }filepath]
    cluster_name
```

This command creates a backup of the MySQL NDB Cluster named *`cluster_name`*. `backup cluster` takes a backup of the cluster's `NDB` tables only; tables using other MySQL storage engines (such as `InnoDB` or `MyISAM`) are ignored.

By default, this command uses the backup ID assigned and returned by `ndb_mgmd` (see the discussions on *`backup_id`* in Using The NDB Cluster Management Client to Create a Backup for more information); you can override this behavior by specifying a backup ID using the `--backupid` option (short form is `-I`).

The `--snapshotstart` option (short form is `-S`) causes the backup to match the state of the cluster when the backup began.

The `--snapshotend` option (short form is `-E`) causes the backup to reflect the state of the cluster when the backup was finished. If neither option is specified, the MySQL Cluster Manager client acts as though `--snapshotend` had been used.

When the `--waitstarted` option (short form is `-w`) is used, the MySQL Cluster Manager client waits until the backup has started before returning control to the user, after which the user can check the backup process's status with the show status command and the `--backup` option.

When the `--waitcompleted` option (short form is `-W`) is used, the MySQL Cluster Manager client waits until the backup process is complete before returning control to the user. If neither `--waitstarted` nor `--waitcompleted` is specified, the client behaves as if `--waitcompleted` had been used.

```
mcm> backup cluster mycluster;
+-------------------------------+
| Command result                |
+-------------------------------+
| Backup completed successfully |
+-------------------------------+
1 row in set (33.50 sec)
```

You can verify that the backup was performed by checking the output of `list backups`, as shown here:

```
mcm> list backups mycluster;
+----------+--------+---------+----------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp            | Parts | Comment |
+----------+--------+---------+----------------------+-------+---------+
| 1        | 1      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
| 1        | 2      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
| 2        | 1      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
| 2        | 2      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
+----------+--------+---------+----------------------+-------+---------+
4 rows in set (0.02 sec)
```

Each row in the output represents a backup *image*—that is, a set of backup files specific to a given backup of a named cluster on a given data node. `Timestamp` values are in UTC. The backup image is saved in the folder *`BackupDataDir`*`/BACKUP/BACKUP-`*`Id`*, where `BackupDataDir` is a cluster parameter. If `BackupDataDir` is not specified, it takes up the value of `DataDir`, so that the image is stored in the directory *`Datadir`*`/BACKUP/BACKUP-`*`backup_id`*.

It is possible to remove an unwanted backup from a given node by deleting this image directory and its contents. To remove a given backup completely, you must remove the corresponding image from each data node's `BACKUP` directory. You can do this as long as a backup or restore operation is not in

progress. It is not necessary to stop the cluster or MySQL Cluster Manager agent prior to removing the images.

The BackupId is used with abort backup and restore cluster.

MySQL Cluster Manager supports the creation of NDB native encrypted cluster backups using AES-256-CBC. To create an encrypted backup, use the --password-file option (short form is -F) to provide a file that contains the password. See Using The NDB Cluster Management Client to Create a Backup for restrictions on the choice of passwords. The password file must satisfy the following requirements:

- The file must be present on the same host as the mcmd agent that the client issuing the backup cluster command is connected to.
- A relative file path is considered relative to the working directory of the mcmd agent mentioned in the last bullet.
- On Unix-like platforms, the file should only be readable and writable by the file owner; on Windows platforms, it should not be readable by the Everyone group.

Logical Backup for NDB Table Metadata

To allow more flexibility for cluster reconfiguration during a restore, the backup cluster command also creates a logical backup for the metadata of the NDB tables in the cluster. Use the --all option with the list backups command to list all backups, including the logical backups for the NDB tables' metadata, which are marked by the comment “Schema”:

```
mcm> list backups --all newcluster;
+----------+--------+---------+---------------------+------+---------+
| BackupId | NodeId | Host    | Timestamp           | Part | Comment |
+----------+--------+---------+---------------------+------+---------+
| 1        | 1      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 2      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 3      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 4      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 50     | tonfisk | 2016-08-12 16:55:55Z |      | Schema  |
+----------+--------+---------+---------------------+------+---------+
5 rows in set (0.02 sec)
```

The logical backup was created using the mysqldump utility. The backup is saved with the file name BACKUP-*BackupID*.*mysql_nodeid*.schema.sql extension, to be found in the folder *backupdatadir*/BACKUP/BACKUP-*id*, where backupdatadir (notice that the name is in lowercase) is a mysqld parameter used only for specifying the location of the logical backup created by MySQL Cluster Manager. If backupdatadir is not specified using the set command with the mcm client, the default value of */mcm_data_repository/clusters/clustername/mysqld_nodeid/* is used, so that the logical backup is saved in the folder */mcm_data_repository*/clusters/*clustername/mysqld_nodeid*/BACKUP/BACKUP-*Id*.

The following restrictions apply for the creation of the logical backups for NDB table metadata:

- At least one mysqld node must be running on the cluster for the logical backup to be performed
- No backup was created for any mysqld node that was not running.
- Metadata for non-NDB tables are not backed up.
- The logical backup is NOT a proper point-in-time backup—no DDL operations should be performed on the cluster when the backup process is running on the cluster, or the backed-up metadata will become inconsistent with the backed-up data.

The backup for the NDB table metadata is helpful for restoring data from a cluster to another one with a different configuration (for example, when the target cluster for restore has a different number of data

nodes); see Section 4.6.2.4, “Partial restore—data nodes added” and Section 4.6.2.5, “Restoring a Backup to a Cluster with Fewer Data Nodes” for some use cases.

5.8.3 The list backups Command

```
list backups [{--backupid=|-I }backup_id] [--all|-a] cluster_name

list backups [{--backupid=|-I }backup_id] [--agent|-A] site_name
```

Without the --agent option, the command lists all backups of the MySQL NDB Cluster named *cluster_name* that are known to this instance of MySQL Cluster Manager. The output includes the backup and node ID as well as a UTC timestamp for each backup, as shown here:

```
mcm> list backups mycluster;
+----------+--------+---------+---------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment |
+----------+--------+---------+---------------------+-------+---------+
| 1        | 1      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
| 1        | 2      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
| 2        | 1      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
| 2        | 2      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
+----------+--------+---------+---------------------+-------+---------+
4 rows in set (0.02 sec)
```

The Timestamp column shows the timestamp (in UTC) of the first file to appear in any backup instance folder. There are 3 files in each backup fileset: *.ctl, *.data, and *.log. If the backup instance folder is empty, the timestamp of the folder itself is shown.

With the --backupid option used, the commands only list backups with the specified ID:

```
mcm> list backups --backupid=2 mycluster;
+----------+--------+---------+---------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment |
+----------+--------+---------+---------------------+-------+---------+
| 2        | 1      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
| 2        | 2      | tonfisk | 2016-10-24 22:24:54Z | 1     |         |
+----------+--------+---------+---------------------+-------+---------+
2 rows in set (0.02 sec)
```

The backup cluster command also creates backups of the metadata for a cluster's NDB tables, which are listed by the list backups command when the --all option is used. The metadata backups are marked by the comment Schema in the backup listing:

```
mcm> list backups --all newcluster;
+----------+--------+---------+---------------------+------+---------+
| BackupId | NodeId | Host    | Timestamp           | Part | Comment |
+----------+--------+---------+---------------------+------+---------+
| 1        | 1      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 2      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 3      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 4      | tonfisk | 2016-08-12 16:55:52Z | 1    |         |
| 1        | 50     | tonfisk | 2016-08-12 16:55:55Z |      | Schema  |
+----------+--------+---------+---------------------+------+---------+
5 rows in set (0.02 sec)
```

See Logical Backup for NDB Table Metadata, for details about the metadata backup.

When the --agent option is used and a *site_name* is specified, the command lists agent backups created for a specific site:

```
mcm> list backups --agent mysite;
+------------+-------+---------+---------------------+-------+--------------+
| BackupId   | Agent | Host    | Timestamp           | Files | Comment      |
+------------+-------+---------+---------------------+-------+--------------+
| 1522914101 | 0     | tonfisk | 2018-04-05 07:41:41Z | 5     | Agent backup |
| 1522914105 | 0     | tonfisk | 2018-04-05 07:41:45Z | 5     | Agent backup |
| 1522914121 | 0     | tonfisk | 2018-04-05 07:42:01Z | 5     | Agent backup |
+------------+-------+---------+---------------------+-------+--------------+
```

```
3 rows in set (0.00 sec)
```

The backup IDs reflect the Unix Epoch times at which the backups were taken.

The output can be filtered with the `--backupid` option:

```
mcm> list backups --agent --backupid=1522914121 mysite;
+------------+-------+---------+---------------------+-------+--------------+
| BackupId   | Agent | Host    | Timestamp           | Files | Comment      |
+------------+-------+---------+---------------------+-------+--------------+
| 1522914121 | 0     | tonfisk | 2018-04-05 07:42:01Z | 5     | Agent backup |
+------------+-------+---------+---------------------+-------+--------------+
1 row in set (0.07 sec)
```

5.8.4 The `delete backup` Command

```
delete backup {--backupid=|-I }backup_id [--skip-nodeid=nodeid-list] cluster_name

nodeid-list:
    nodeid[, nodeid[, ...]]
```

The command removes a backup's directories and their contents on both data nodes and `mysqld` nodes of the cluster named *`cluster_name`*:

```
mcm> list backups mycluster;
+----------+--------+---------+---------------------+-------+---------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment |
+----------+--------+---------+---------------------+-------+---------+
| 1        | 1      | flundra | 2018-04-04 06:31:12Z | 1     |         |
| 1        | 2      | tonfish | 2018-04-04 06:31:12Z | 1     |         |
+----------+--------+---------+---------------------+-------+---------+
2 rows in set (0.04 sec)

mcm> delete backup --backupid=1 mycluster;
+-----------------------------+
| Command result              |
+-----------------------------+
| Backup deleted successfully |
+-----------------------------+
1 row in set (1.22 sec)

mcm> list backups mycluster;
+----------+--------+---------+---------------------+-------+------------------+
| BackupId | NodeId | Host    | Timestamp           | Parts | Comment          |
+----------+--------+---------+---------------------+-------+------------------+
| None     | 2      | tonfish |                     |       | No backups found |
| None     | 1      | flundra |                     |       | No backups found |
+----------+--------+---------+---------------------+-------+------------------+
2 rows in set (0.09 sec)
```

The ID of the backup to be deleted must be specified using the `--backupid` option. To delete all backups, run the command on every single backup.

If any data nodes or `mysqld` nodes have been added after the specified backup was created, list their node IDs with the `--skip-nodeid` option, or the operation will fail with the complaint that no backup directories exist on those nodes.

Note

The command fails if there are files other than the backup files in any of the backup directories to be deleted. Remove those extra files manually before running the command.

5.8.5 The `restore cluster` Command

```
restore cluster
    {--backupid=|-I }backup_id
    [--disable-indexes|-x]
```

```
    [--disable-metadata|-M]
    [--epoch|-e]
    [--exclude-databases=db_name]
    [--exclude-intermediate-sql-tables]
    [--exclude-missing-columns]
    [--exclude-missing-tables]
    [--exclude-tables=db_name.tbl_name[,db_name.tbl_name][,...]]
    [--include-databases=db_name]
    [--include-stored-grants]
    [--include-tables=db_name.tbl_name[,db_name.tbl_name][,...]]
    [--lossy-conversions]
    [--no-binlog|-l]
    [--no-restore-disk-objects]
    [{--parallelism=|-p }#]
    [--progress-frequency]
    [--promote-attributes]
    [--rewrite-database]
    [--skip-broken-objects]
    [{--skip-nodeid=|-s }id_list]
    [--skip-table-check]
    [--skip-unknown-objects]
    [--password-file=filepath]
    cluster_name
```

This command restores a cluster from a backup having the specified backup ID (`--backupid` option; short form: `-I`) to the MySQL NDB Cluster named *`cluster_name`*. In its simplest form, it can be used as shown here, to restore the cluster named `mycluster` to the state saved in the backup having backup ID 3:

```
mcm> restore cluster --backupid=3 mycluster;
+--------------------------------+
| Command result                 |
+--------------------------------+
| Restore completed successfully |
+--------------------------------+
1 row in set (18.60 sec)
```

If you are restoring an existing cluster to a known good state, you must wipe any existing data first. Stop the cluster using `stop cluster`, then restart it using `start cluster` with the `--initial` option, which causes the data node file systems to be cleared. Following this, you can restore the cluster from the desired backup using `restore cluster`.

Important

In order to restore a backup using `restore cluster`, the cluster must have an unused slot for an `ndbapi` process in its configuration. Otherwise, the command fails with the error `Unable to perform restore - no vacant ndbapi slots in config for cluster cluster_name`. See Adding Free Processes, for information on how to add a free `ndbapi` slot to your cluster.

Additional options that can be employed with this command include:

`--disable-indexes` and `--disable-metadata`. To cause indexes to be ignored when restoring the table data, use the `--disable-indexes` option. Doing this can decrease the time required to restore a large data set, particularly where many indexes were in use. Similarly, you can cause metadata to be ignored during the restoration process by using the `--disable-metadata` option (short form: `-M`).

`--epoch`. When the `--epoch` option (short form: `-e`) is used, epoch information is restored to the cluster replication status table (`mysql.ndb_apply_status`), which can be useful for replicas in MySQL NDB Cluster replication.

`--exclude-databases` and `--exclude-tables`. Prevent one or more databases or tables from being restored using the options `--exclude-databases` and `--exclude-tables`.

--exclude-databases takes a comma-delimited list of one or more databases that should not be restored. --exclude-tables takes a comma-delimited list of one or more tables (using the *database.table* format) that should not be restored. When --exclude-databases or --exclude-tables is used, only those databases or tables named by the option are excluded; all other databases and tables are restored.

--exclude-missing-columns. When this option is used, restore cluster ignores any columns missing from tables being restored as compared to the versions of those tables found in the backup.

--exclude-missing-tables. When this option is used, restore cluster ignores any tables from the backup that are not found in the target database.

--exclude-intermediate-sql-tables[=TRUE|FALSE]. When performing ALTER TABLE operations, mysqld creates intermediate tables (whose names are prefixed with #sql-). When TRUE, the --exclude-intermediate-sql-tables option keeps restore cluster from restoring such tables that may have been left over from such operations. This option is TRUE by default.

--include-databases and --include-tables. Use the --include-databases option or the --include-tables option for restoring only specific databases or tables, respectively. --include-databases takes a comma-delimited list of databases to be restored. --include-tables takes a comma-delimited list of tables (in the *database.table* format) to be restored. When --include-databases or --include-tables is used, only those databases or tables named by the option are restored; all other databases and tables are excluded by restore cluster, and are not restored.

--include-stored-grants. *When managing NDB Cluster 8.0.19 and later,* the restore cluster command does not restore shared users and grants to the mysql.ndb_sql_metadata table by default; use the --include-stored-grants option to override this behavior and enable the restore of shared user and grant data and metadata.

--lossy-conversions. Using --lossy-conversions allows lossy conversions of column values (type demotions or changes in sign) when restoring data from backup. With some exceptions, the rules governing demotion are the same as for MySQL replication; see Replication of Columns Having Different Data Types, for information about specific type conversions currently supported by attribute demotion. restore cluster reports any truncation of data that it performs during lossy conversions once per attribute and column.

--no-binlog. The --no-binlog option (short form: -l) stops any SQL nodes (mysqld processes) in the cluster from writing data from the restore into their binary logs.

--no-restore-disk-objects. This option stops restore cluster from restoring any MySQL NDB Cluster Disk Data objects, such as tablespaces and log file groups; see NDB Cluster Disk Data Tables, for more information about these objects.

--parallelism=#. The --parallelism option (short form: -p) sets the maximum number of parallel transactions that the restore cluster command attempts to use. The default value is 128; the maximum is 1024, and the minimum is 1.

--progress-frequency=*N*. Print a status report each *N* seconds to a temporary stdout dump file mcm creates at mcm_data/clusters/*cluster_name*/*nodeid*/tmp while the backup is in progress. 0 (the default) causes no status reports to be printed. The maximum is 65535.

--promote-attributes. Allow attributes to be promoted when MySQL Cluster Manager restores data from a backup. See the discussion on attribute promotion in the MySQL NDB Cluster manual for more details.

--rewrite-database=*old_dbname*,*new_dbname*. This option causes a database with the name *old_dbname* in the backup to be restored under the name *new_dbname*.

--skip-nodeid. The --skip-nodeid option (short form: -s) takes a comma-separated list of node IDs. The nodes whose IDs are listed may include of data nodes, SQL nodes, or both. Nodes having these IDs are skipped by the restoration process.

--skip-broken-objects. This option causes restore cluster to ignore corrupt tables while reading a backup, and to continue restoring any remaining tables (that are not also corrupted). Currently, the --skip-broken-objects option works only in the case of missing blob parts tables.

--skip-table-check. It is possible to restore data without restoring table metadata. The default behavior when doing this is for restore cluster to fail with an error if table data do not match the table schema; this can be overridden using the --skip-table-check option.

--skip-unknown-objects. This option causes restore cluster to ignore any schema objects it does not recognize while reading a backup. This can be used for restoring, for example, a backup made from a newer version of MySQL NDB Cluster to an older version.

Supports the restoring of NDB native encrypted cluster backups. To restore an encrypted backup, use the --password-file option to provide a file that contains the encryption password for the backup. The following must be true for the password file:

- The file must be present on each host where mcmd runs a data node that is being restored.
- If the file path is relative, it is relative to the working directory of the mcmd agent mentioned in the last bullet.

5.8.6 The backup agents Command

```
backup agents [--hosts=host_list] [site_name]

host_list:
    host[, host[, ...]]
```

This command backs up the configuration data for the mcmd agents on the hosts specified in *host_list* with the --hosts option (short form: -h) for the site named *site_name*. If no host names are specified, all agents of the site are backed up. If no *site_name* is given, only the agent that the mcm client is connected to is backed up.

The backup for each agent is created in a subfolder named rep_backup/*timestamp* under the agent repository (the mcm_data folder), with *timestamp* reflecting the time the backup began. If you want the backup to be at another place, create a soft link from mcm_data/rep_backup to your desired storage location.

An empty file named INCOMPLETE is created in the folder in which the backup is created when the backup begins, and is deleted after the backup is finished. The continuous existence of the file after the backup process is over indicates that the backup is incomplete.

Notice that the backup agents command works differently from the backup cluster command, which backs up cluster data; the backup agents command, on the other hand, backs up agent configuration data. Using together the backups created by both commands, you can restore not just the cluster, but the complete cluster-plus-manager setup. See Section 4.7, “Backing Up and Restoring MySQL Cluster Manager Agents” for more details about backing up and restoring mcmd agents.

5.9 MySQL Cluster Manager Cluster Importation Commands

This section contains descriptions of MySQL Cluster Manager commands used to perform operations connected with importing clusters into MySQL Cluster Manager. These operations include migration of cluster processes and copying of configuration data.

5.9.1 The import cluster Command

```
import cluster [--dryrun|-y] [--remove-angel] cluster_name
```

This command imports a MySQL NDB Cluster created independently of MySQL Cluster Manager into a cluster named *cluster_name* that has been created in MySQL Cluster Manager. You are strongly advised to create *cluster_name* using the create cluster command's --import option; see that command's description for more information about the --import option and its effects.

import cluster requires a single argument, the name of the cluster created using MySQL Cluster Manager (*cluster_name*) into which you wish to import a MySQL NDB Cluster created externally into MySQL Cluster Manager and bring it under MySQL Cluster Manager control. The cluster named in the command must already exist in MySQL Cluster Manager.

import cluster also supports a --dryrun option. When this option is used, only the checks required for importation are performed against the existing cluster. This makes it possible to test a given configuration without actually placing any cluster processes under MCM control. -y is supported as a short form of this option.

import cluster supports a --remove-angel option. When this option is used, any running angel processes for the data nodes of the cluster to be imported are stopped by mcmd prior to the actual import, which is a necessary step for a cluster import unless the angel processes have already been stopped manually. When this option is used together with the --dryrun option, no removals of angel processes will actually be performed, but the checks for angel processes (which occur when the --dryrun option is used alone) will be skipped. It is recommended that you use the two options separately: perform the checks with the --dryrun option only, and once the only errors observed are with the angel processes, run import cluster again with the --remove-angel option only to complete the import.

For more information about importing clusters into MySQL Cluster Manager, including examples, see Section 4.5, “Importing MySQL NDB Clusters into MySQL Cluster Manager”.

5.9.2 The import config Command

```
import config [--dryrun|-y] [--retry] cluster_name
```

This command imports the configuration of an autonomous or “wild” cluster into the cluster named *cluster_name*.

import config requires a single argument, *cluster_name*, which is the name of the cluster created using MySQL Cluster Manager into which you wish to import the configuration of a MySQL NDB Cluster created externally. The cluster named in the command must already exist in MySQL Cluster Manager; you are also strongly advised to use create cluster --import when creating *cluster_name*.

Note

When importing configuration attributes for a mysqld node, if a relative path is used for the socket value or for any directory value (for example, plugin_dir), the import will be rejected by the mcm client. Make sure an absolute path is used in those cases and, if necessary, make adjustments to the attributes in the .mcm file produced by the import config --dryrun command and then import the settings by executing the file with the mcm client.

import config also supports a --dryrun option (short form: -y). When this option is used, the checks required for importing the configuration data are performed, and the set commands for performing the actual import are written to the file **/path-to-mcm-data-repository/**clusters/**clustername**/tmp/import_config.**message_id**.mcm for your examination. This makes it possible to test the configuration import without actually copying any of the settings into the cluster controlled by MySQL Cluster Manager. You can then import all the settings using the import config command (without the --dryrun option), or adjust some of the settings in the **/path-to-**

`mcm-data-repository``/clusters/`**`clustername`**`/tmp/import_config.`**`message_id`**`.mcm` file and then import the settings by executing the file with the `mcm` agent. See Section 4.5.2.3, “Creating and Configuring the Target Cluster” for examples on using the `import config` command.

`import config` supports a `--retry` option, which reimports the cluster configuration from the `config.ini` file after the cluster is already running.

Warning

The `--retry` option might cause unexpected effects on the cluster. It should only be used at the instruction of Oracle Support.

Chapter 6 MySQL Cluster Manager Limitations and Known Issues

Table of Contents

In this chapter we discuss limitations of and known issues in MySQL Cluster Manager version 9.6.0.

6.1 MySQL Cluster Manager Usage and Design Limitations

The limitations discussed in this section occur by intention or design in MySQL Cluster Manager 9.6.0. Some of these items may become obsolete in future versions; we will update this section accordingly if and as those changes come about.

`change process` command. Currently, the `change process` command can be used only to exchange an `ndbd` process for an `ndbmtd` process, or the reverse. That is, in effect, it can be used only to switch a data node between a single-threaded process and a multithreaded process. It cannot be used for changing a cluster node's type (for example, you cannot change a data node to an SQL node, management node, or NDB API application node).

Concurrent client sessions. Currently there is no negotiation or arbitration between multiple `mcm` clients. While it is possible to use the client from multiple locations, you should be careful always to allow a reconfiguration command issued in one `mcm` client session to finish executing before issuing a new command in a different client session.

IPv6 and host names (Windows). When IPv6 support is enabled on Windows systems, host names other than `localhost` are resolved using IPv6. When an IPv6-enabled Windows system is used as a MySQL NDB Cluster host under MySQL Cluster Manager, you must reference it using its IPv4 address. Otherwise, `mcm` will be unable to connect to the agent process on that host.

This applies to host names used with the MySQL Cluster Manager client commands `create cluster`, `create site`, `add hosts`, `add package`, `delete package`, `stop agents`, and `add process`.

Use of antivirus software on Windows platforms. On-access scanning by antivirus software on Windows platforms might cause access to the cluster configuration file being denied for the `mcmd` agent, causing updates for the cluster configuration to fail sometimes.

6.2 MySQL Cluster Manager 9.6.0 Limitations Relating to the MySQL Server

The limitations described in this section relate to functionality in the MySQL Server that is unsupported or reduced, or otherwise differs when using it with MySQL Cluster Manager.

Replication. Replication is currently not directly supported by MySQL Cluster Manager. See Section 6.3, “MySQL Cluster Manager Limitations Relating to MySQL NDB Cluster”, for more information.

Limited `mysqld` option modifier support. MySQL Cluster Manager does not recognize the `--loose`, `--maximum`, `--enable`, and `--disable` prefixes for `mysqld` options used as MySQL Cluster Manager configuration attributes (for a description of these modifiers, see Program Option Modifiers).

For example, the command `set loose-skip-innodb:mysqld=true mycluster;` fails with the error `No such config variable loose-skip-innodb for process mysqld`.

The `--skip` option modifier is supported in some but not all cases, so that commands such as `set skip-innodb:mysqld=true mycluster;` and `set skip-grant-tables:mysqld=true mycluster;` can be used with MySQL Cluster Manager, while `set skip-column-names:mysqld=true mycluster;` cannot. (Bug #48559, Bug #47779)

Dashes and underscores in MySQL option and variable names. When using the `mysql` client or other MySQL client applications, many MySQL system options and variables can be named using either dashes or underscores in their names. For example, you can use either `ndb_batch_size` or `ndb-batch-size` with the MySQL Server, and the variable is set correctly. This is not the case in MySQL Cluster Manager, where only the forms using underscores are accepted as attribute names. For example, assuming that `mycluster` is a viable cluster, the command `set ndb_batch_size:mysqld=65536 mycluster;` works to set the size of ndb_batch_size on all `mysqld` processes in the cluster, but `set ndb-batch-size:mysqld=65536 mycluster;` fails.

Dependencies between MySQL Cluster Manager `mysqld` attributes and MySQL server variables. MySQL Cluster Manager does not track dependencies between `mysqld` attributes (MySQL server options and system variables). That means MySQL Cluster Manager might have `mysqld` started successfully and report so, even though the server has ended up in a non-functional state because dependent attributes were set inconsistently. It is therefore a good idea for users to check the `mysqld` attributes before starting the node and the `mysql` log for status of the node after it has been started.

MySQL Cluster Manager `mysqld` attributes and MySQL user variables. MySQL user variables are not accessible as MySQL Cluster Manager configuration attributes.

Unsupported MySQL 8.4 and 8.0 Features. These MySQL 8.4 and 8.0 features are **not** supported by MySQL Cluster Manager 9.6:

- The `--upgrade` option for `mysqld`.
- The `SET PERSIST` statement for persisting system variables.

6.3 MySQL Cluster Manager Limitations Relating to MySQL NDB Cluster

This section describes limitations relating to MySQL NDB Cluster functionality that is unsupported or curtailed by MySQL Cluster Manager 9.6.

MySQL Cluster Manager and replication. MySQL Cluster Manager currently does not provide any explicit support for MySQL NDB Cluster Replication. However, you should still be able to perform manual setup of replication of a MySQL NDB Cluster that is managed by MySQL Cluster Manager.

Backup and restore operations. MySQL Cluster Manager provides integrated backup and restore functionality. You can back up `NDB` databases and tables using the `mcm` client `backup cluster` command, and restore them using the `restore cluster` client command. MySQL Cluster Manager also supports restoration of distributed privileges.

You can also back up `NDB` databases and tables using the `ndb_mgm` client `START BACKUP` command, and restore them using the `ndb_restore` program; however MySQL Cluster Manager is not aware of backups that it was not employed to create. Both of the programs just mentioned are supplied with the MySQL NDB Cluster distribution.

Note

Backups of tables using storage engines other than `NDB`, as well as of all other database objects that are not tables, cannot be made using MySQL Cluster Manager, and must be made using some other method, such as `mysqldump`.

Rolling restarts. Currently, all cluster nodes must be running in order to perform a rolling restart using MySQL Cluster Manager. However, MySQL NDB Cluster itself requires only that at least one management server and all data nodes are running (in other words, any `mysqld` processes and any additional `ndb_mgmd` processes can be stopped). In such cases, you can perform the rolling restart manually, after stopping the MySQL Cluster Manager agent.

When making changes in configuration attributes only those nodes requiring a restart to make the change take effect are actually restarted. `ndbapi` nodes are never restarted by MySQL Cluster Manager.

Cluster Imports.
MySQL Cluster Manager will reject an import if it cannot access the process information of the cluster being imported. Therefore, the MySQL Cluster Manager agents must be run by a sufficiently privileged user—normally the same user that runs the cluster.

Cluster Reconfiguration. Cluster configuration updates (using the `set` or `reset` command) that would trigger a rolling restart of the nodes are not executed by MySQL Cluster Manager unless there are more than one data node defined for each node group; to perform such updates when the requirement is not met, a user should, using MySQL Cluster Manager, stop the cluster, use the `set` or `reset` command to change the cluster configurations, and then start the cluster again. If, however, your configuration changes require an initial restart of your cluster, you will need to backup the data, recreate your cluster form scratch with the new settings, and then restore your old data onto it.

6.4 Syntax and Related Issues in MySQL Cluster Manager

This section covers MySQL Cluster Manager issues relating to limitations in SQL and other syntax.

Appendix A Attribute Summary Tables

Table of Contents

This appendix provides tables of configuration attributes, grouped according to their process type or by the section of the MySQL NDB Cluster configuration file in which they appear. This information is current for MySQL NDB Cluster 9.6. For information regarding MySQL NDB Cluster 9.4, see MySQL NDB Cluster 9.6. For information regarding MySQL NDB Cluster 8.4, see MySQL NDB Cluster 8.4. For information regarding MySQL NDB Cluster 8.0, see MySQL NDB Cluster 8.0.

Each table provides the following information:

- *Name*: The name of the attribute. The name of the attribute is linked to the attribute's full description in the online MySQL NDB Cluster documentation.
- *Type/Units*: The data type or unit by which the attribute is measured.
- *Range*: The default value of the attribute, if not set by the user, and the minimum and maximum values that can be set for the attribute.
- *Restart Type*: The type of restart required for a change in value in this attribute to be applied in a running MySQL NDB Cluster. The restart type is indicated in this column by an **N** for a node restart, or an **S** for a system restart. *Data node attributes*: The presence of an **I** in this column indicates that a data node must be restarted using the `--initial` option for a change to take effect.

  Attributes having restart type **N** can be changed using a rolling restart of the cluster, and thus can be changed at any time, even if the cluster is running. Changing an attribute whose restart type is **S** requires a complete shutdown of all cluster nodes, followed by a restart of the nodes once all of them have been stopped. Currently, such attributes can be set only before starting a cluster for the first time.

A.1 Management Node Configuration Parameters

- `ArbitrationDelay`: When asked to arbitrate, arbitrator waits this long before voting (milliseconds).
- `ArbitrationRank`: If 0, then management node is not arbitrator. Kernel selects arbitrators in order 1, 2.
- `DataDir`: Data directory for this node.
- `ExecuteOnComputer`: String referencing earlier defined COMPUTER.
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `HeartbeatIntervalMgmdMgmd`: Time between management-node-to-management-node heartbeats; connection between management nodes is considered lost after 3 missed heartbeats.
- `HeartbeatThreadPriority`: Set heartbeat thread policy and priority for management nodes; see manual for allowed values.
- `HostName`: Host name or IP address for this management node.

- `Id`: Number identifying management node. Now deprecated; use NodeId instead.
- `LocationDomainId`: Assign this management node to specific availability domain or zone. 0 (default) leaves this unset.
- `LogDestination`: Where to send log messages: console, system log, or specified log file.
- `NodeId`: Number uniquely identifying management node among all nodes in cluster.
- `PortNumber`: Port number to send commands to and fetch configuration from management server.
- `PortNumberStats`: Port number used to get statistical information from management server.
- `RequireTls`: Client connection must authenticate with TLS before being used otherwise.
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers.
- `wan`: Use WAN TCP setting as default.

A.2 Data Node Configuration Parameters

- `ApiFailureHandlingTimeout`: Maximum time for API node failure handling before escalating. 0 means no time limit; minimum usable value is 10.
- `Arbitration`: How arbitration should be performed to avoid split-brain issues in event of node failure.
- `ArbitrationTimeout`: Maximum time (milliseconds) database partition waits for arbitration signal.
- `BackupDataBufferSize`: Default size of databuffer for backup (in bytes).
- `BackupDataDir`: Path to where to store backups. Note that string '/BACKUP' is always appended to this setting, so that *effective* default is FileSystemPath/BACKUP.
- `BackupDiskWriteSpeedPct`: Sets percentage of data node's allocated maximum write speed (MaxDiskWriteSpeed) to reserve for LCPs when starting backup.
- `BackupLogBufferSize`: Default size of log buffer for backup (in bytes).
- `BackupMaxWriteSize`: Maximum size of file system writes made by backup (in bytes).
- `BackupMemory`: Total memory allocated for backups per node (in bytes).
- `BackupReportFrequency`: Frequency of backup status reports during backup in seconds.
- `BackupWriteSize`: Default size of file system writes made by backup (in bytes).
- `BatchSizePerLocalScan`: Used to calculate number of lock records for scan with hold lock.
- `BuildIndexThreads`: Number of threads to use for building ordered indexes during system or node restart. Also applies when running ndb_restore --rebuild-indexes. Setting this parameter to 0 disables multithreaded building of ordered indexes.
- `CompressedBackup`: Use zlib to compress backups as they are written.
- `CompressedLCP`: Write compressed LCPs using zlib.
- `ConnectCheckIntervalDelay`: Time between data node connectivity check stages. Data node is considered suspect after 1 interval and dead after 2 intervals with no response.
- `CrashOnCorruptedTuple`: When enabled, forces node to shut down whenever it detects corrupted tuple.
- `DataDir`: Data directory for this node.

- `DataMemory`: Number of bytes on each data node allocated for storing data; subject to available system RAM and size of IndexMemory.
- `DefaultHashMapSize`: Set size (in buckets) to use for table hash maps. Three values are supported: 0, 240, and 3840.
- `DictTrace`: Enable DBDICT debugging; for NDB development.
- `DiskDataUsingSameDisk`: Set to false if Disk Data tablespaces are located on separate physical disks.
- `DiskIOThreadPool`: Number of unbound threads for file access, applies to disk data only.
- `Diskless`: Run without using disk.
- `DiskPageBufferEntries`: Memory to allocate in DiskPageBufferMemory; very large disk transactions may require increasing this value.
- `DiskPageBufferMemory`: Number of bytes on each data node allocated for disk page buffer cache.
- `DiskSyncSize`: Amount of data written to file before synch is forced.
- `EnablePartialLcp`: Enable partial LCP (true); if this is disabled (false), all LCPs write full checkpoints.
- `EnableRedoControl`: Enable adaptive checkpointing speed for controlling redo log usage.
- `EncryptedFileSystem`: Encrypt local checkpoint and tablespace files. EXPERIMENTAL; NOT SUPPORTED IN PRODUCTION.
- `EventLogBufferSize`: Size of circular buffer for NDB log events within data nodes.
- `ExecuteOnComputer`: String referencing earlier defined COMPUTER.
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `FileSystemPath`: Path to directory where data node stores its data (directory must exist).
- `FileSystemPathDataFiles`: Path to directory where data node stores its Disk Data files. Default value is FilesystemPathDD, if set; otherwise, FilesystemPath is used if it is set; otherwise, value of DataDir is used.
- `FileSystemPathDD`: Path to directory where data node stores its Disk Data and undo files. Default value is FileSystemPath, if set; otherwise, value of DataDir is used.
- `FileSystemPathUndoFiles`: Path to directory where data node stores its undo files for Disk Data. Default value is FilesystemPathDD, if set; otherwise, FilesystemPath is used if it is set; otherwise, value of DataDir is used.
- `FragmentLogFileSize`: Size of each redo log file.
- `HeartbeatIntervalDbApi`: Time between API node-data node heartbeats. (API connection closed after 3 missed heartbeats).
- `HeartbeatIntervalDbDb`: Time between data node-to-data node heartbeats; data node considered dead after 3 missed heartbeats.
- `HeartbeatOrder`: Sets order in which data nodes check each others' heartbeats for determining whether given node is still active and connected to cluster. Must be zero for all data nodes or distinct nonzero values for all data nodes; see documentation for further guidance.

- `HostName`: Host name or IP address for this data node.
- `IndexMemory`: Number of bytes on each data node allocated for storing indexes; subject to available system RAM and size of DataMemory.
- `IndexStatAutoCreate`: Enable/disable automatic statistics collection when indexes are created.
- `IndexStatAutoUpdate`: Monitor indexes for changes and trigger automatic statistics updates.
- `IndexStatSaveScale`: Scaling factor used in determining size of stored index statistics.
- `IndexStatSaveSize`: Maximum size in bytes for saved statistics per index.
- `IndexStatTriggerPct`: Threshold percent change in DML operations for index statistics updates. Value is scaled down by IndexStatTriggerScale.
- `IndexStatTriggerScale`: Scale down IndexStatTriggerPct by this amount, multiplied by base 2 logarithm of index size, for large index. Set to 0 to disable scaling.
- `IndexStatUpdateDelay`: Minimum delay between automatic index statistics updates for given index. 0 means no delay.
- `InitFragmentLogFiles`: Initialize fragment log files, using sparse or full format.
- `InitialLogFileGroup`: Describes log file group that is created during initial start. See documentation for format.
- `InitialNoOfOpenFiles`: Initial number of files open per data node. (One thread is created per file).
- `InitialTablespace`: Describes tablespace that is created during initial start. See documentation for format.
- `InsertRecoveryWork`: Percentage of RecoveryWork used for inserted rows; has no effect unless partial local checkpoints are in use.
- `KeepAliveSendInterval`: Time between keep-alive signals on links between data nodes, in milliseconds. Set to 0 to disable.
- `LateAlloc`: Allocate memory after connection to management server has been established.
- `LcpScanProgressTimeout`: Maximum time that local checkpoint fragment scan can be stalled before node is shut down to ensure systemwide LCP progress. Use 0 to disable.
- `LocationDomainId`: Assign this data node to specific availability domain or zone. 0 (default) leaves this unset.
- `LockExecuteThreadToCPU`: Comma-delimited list of CPU IDs.
- `LockMaintThreadsToCPU`: CPU ID indicating which CPU runs maintenance threads.
- `LockPagesInMainMemory`: 0=disable locking, 1=lock after memory allocation, 2=lock before memory allocation.
- `LogLevelCheckpoint`: Log level of local and global checkpoint information printed to stdout.
- `LogLevelCongestion`: Level of congestion information printed to stdout.
- `LogLevelConnection`: Level of node connect/disconnect information printed to stdout.
- `LogLevelError`: Transporter, heartbeat errors printed to stdout.
- `LogLevelInfo`: Heartbeat and log information printed to stdout.

- `LogLevelNodeRestart`: Level of node restart and node failure information printed to stdout.
- `LogLevelShutdown`: Level of node shutdown information printed to stdout.
- `LogLevelStartup`: Level of node startup information printed to stdout.
- `LogLevelStatistic`: Level of transaction, operation, and transporter information printed to stdout.
- `LongMessageBuffer`: Number of bytes allocated on each data node for internal long messages.
- `MaxAllocate`: No longer used; has no effect.
- `MaxBufferedEpochs`: Allowed numbered of epochs that subscribing node can lag behind (unprocessed epochs). Exceeding causes lagging subscribers to be disconnected.
- `MaxBufferedEpochBytes`: Total number of bytes allocated for buffering epochs.
- `MaxDiskDataLatency`: Maximum allowed mean latency of disk access (ms) before starting to abort transactions.
- `MaxDiskWriteSpeed`: Maximum number of bytes per second that can be written by LCP and backup when no restarts are ongoing.
- `MaxDiskWriteSpeedOtherNodeRestart`: Maximum number of bytes per second that can be written by LCP and backup when another node is restarting.
- `MaxDiskWriteSpeedOwnRestart`: Maximum number of bytes per second that can be written by LCP and backup when this node is restarting.
- `MaxFKBuildBatchSize`: Maximum scan batch size to use for building foreign keys. Increasing this value may speed up builds of foreign keys but impacts ongoing traffic as well.
- `MaxDMLOperationsPerTransaction`: Limit size of transaction; aborts transaction if it requires more than this many DML operations.
- `MaxLCPStartDelay`: Time in seconds that LCP polls for checkpoint mutex (to allow other data nodes to complete metadata synchronization), before putting itself in lock queue for parallel recovery of table data.
- `MaxNoOfAttributes`: Suggests total number of attributes stored in database (sum over all tables).
- `MaxNoOfConcurrentIndexOperations`: Total number of index operations that can execute simultaneously on one data node.
- `MaxNoOfConcurrentOperations`: Maximum number of operation records in transaction coordinator.
- `MaxNoOfConcurrentScans`: Maximum number of scans executing concurrently on data node.
- `MaxNoOfConcurrentSubOperations`: Maximum number of concurrent subscriber operations.
- `MaxNoOfConcurrentTransactions`: Maximum number of transactions executing concurrently on this data node, total number of transactions that can be executed concurrently is this value times number of data nodes in cluster.
- `MaxNoOfFiredTriggers`: Total number of triggers that can fire simultaneously on one data node.
- `MaxNoOfLocalOperations`: Maximum number of operation records defined on this data node.
- `MaxNoOfLocalScans`: Maximum number of fragment scans in parallel on this data node.
- `MaxNoOfOpenFiles`: Maximum number of files open per data node.(One thread is created per file).
- `MaxNoOfOrderedIndexes`: Total number of ordered indexes that can be defined in system.

- `MaxNoOfSavedMessages`: Maximum number of error messages to write in error log and maximum number of trace files to retain.
- `MaxNoOfSubscribers`: Maximum number of subscribers.
- `MaxNoOfSubscriptions`: Maximum number of subscriptions (default 0 = MaxNoOfTables).
- `MaxNoOfTables`: Suggests total number of NDB tables stored in database.
- `MaxNoOfTriggers`: Total number of triggers that can be defined in system.
- `MaxNoOfUniqueHashIndexes`: Total number of unique hash indexes that can be defined in system.
- `MaxParallelCopyInstances`: Number of parallel copies during node restarts. Default is 0, which uses number of LDMs on both nodes, to maximum of 16.
- `MaxParallelScansPerFragment`: Maximum number of parallel scans per fragment. Once this limit is reached, scans are serialized.
- `MaxReorgBuildBatchSize`: Maximum scan batch size to use for reorganization of table partitions. Increasing this value may speed up table partition reorganization but impacts ongoing traffic as well.
- `MaxStartFailRetries`: Maximum retries when data node fails on startup, requires StopOnError = 0. Setting to 0 causes start attempts to continue indefinitely.
- `MaxUIBuildBatchSize`: Maximum scan batch size to use for building unique keys. Increasing this value may speed up builds of unique keys but impacts ongoing traffic as well.
- `MemReportFrequency`: Frequency of memory reports in seconds; 0 = report only when exceeding percentage limits.
- `MinDiskWriteSpeed`: Minimum number of bytes per second that can be written by LCP and backup.
- `MinFreePct`: Percentage of memory resources to keep in reserve for restarts.
- `NodeGroup`: Node group to which data node belongs; used only during initial start of cluster.
- `NodeGroupTransporters`: Number of transporters to use between nodes in same node group.
- `NodeId`: Number uniquely identifying data node among all nodes in cluster.
- `NoOfFragmentLogFiles`: Number of 16 MB redo log files in each of 4 file sets belonging to data node.
- `NoOfReplicas`: Number of copies of all data in database.
- `Numa`: (Linux only; requires libnuma) Controls NUMA support. Setting to 0 permits system to determine use of interleaving by data node process; 1 means that it is determined by data node.
- `ODirect`: Use O_DIRECT file reads and writes when possible.
- `ODirectSyncFlag`: O_DIRECT writes are treated as synchronized writes; ignored when ODirect is not enabled, InitFragmentLogFiles is set to SPARSE, or both.
- `RealtimeScheduler`: When true, data node threads are scheduled as real-time threads. Default is false.
- `RecoveryWork`: Percentage of storage overhead for LCP files: greater value means less work in normal operations, more work during recovery.
- `RedoBuffer`: Number of bytes on each data node allocated for writing redo logs.

- `RedoOverCommitCounter`: When RedoOverCommitLimit has been exceeded this many times, transactions are aborted, and operations are handled as specified by DefaultOperationRedoProblemAction.
- `RedoOverCommitLimit`: Each time that flushing current redo buffer takes longer than this many seconds, number of times that this has happened is compared to RedoOverCommitCounter.
- `RequireEncryptedBackup`: Whether backups must be encrypted (1 = encryption required, otherwise 0).
- `RequireCertificate`: Node is required to find key and certificate in TLS search path.
- `RequireTls`: Require TLS-authenticated secure connections.
- `ReservedConcurrentIndexOperations`: Number of simultaneous index operations having dedicated resources on one data node.
- `ReservedConcurrentOperations`: Number of simultaneous operations having dedicated resources in transaction coordinators on one data node.
- `ReservedConcurrentScans`: Number of simultaneous scans having dedicated resources on one data node.
- `ReservedConcurrentTransactions`: Number of simultaneous transactions having dedicated resources on one data node.
- `ReservedFiredTriggers`: Number of triggers having dedicated resources on one data node.
- `ReservedLocalScans`: Number of simultaneous fragment scans having dedicated resources on one data node.
- `ReservedTransactionBufferMemory`: Dynamic buffer space (in bytes) for key and attribute data allocated to each data node.
- `RestartOnErrorInsert`: Control type of restart caused by inserting error (when StopOnError is enabled).
- `RestartSubscriberConnectTimeout`: Amount of time for data node to wait for subscribing API nodes to connect. Set to 0 to disable timeout, which is always resolved to nearest full second.
- `SchedulerExecutionTimer`: Number of microseconds to execute in scheduler before sending.
- `SchedulerResponsiveness`: Set NDB scheduler response optimization 0-10; higher values provide better response time but lower throughput.
- `SchedulerSpinTimer`: Number of microseconds to execute in scheduler before sleeping.
- `ServerPort`: Port used to set up transporter for incoming connections from API nodes.
- `SharedGlobalMemory`: Total number of bytes on each data node allocated for any use.
- `SpinMethod`: Determines spin method used by data node; see documentation for details.
- `StartFailRetryDelay`: Delay in seconds after start failure prior to retry; requires StopOnError = 0.
- `StartFailureTimeout`: Milliseconds to wait before terminating. (0=Wait forever).
- `StartNoNodeGroupTimeout`: Time to wait for nodes without nodegroup before trying to start (0=forever).
- `StartPartialTimeout`: Milliseconds to wait before trying to start without all nodes. (0=Wait forever).

- `StartPartitionedTimeout`: Milliseconds to wait before trying to start partitioned. (0=Wait forever).
- `StartupStatusReportFrequency`: Frequency of status reports during startup.
- `StopOnError`: When set to 0, data node automatically restarts and recovers following node failures.
- `StringMemory`: Default size of string memory (0 to 100 = % of maximum, 101+ = actual bytes).
- `TcpBind_INADDR_ANY`: Bind IP_ADDR_ANY so that connections can be made from anywhere (for autogenerated connections).
- `TimeBetweenEpochs`: Time between epochs (synchronization used for replication).
- `TimeBetweenEpochsTimeout`: Timeout for time between epochs. Exceeding causes node shutdown.
- `TimeBetweenGlobalCheckpoints`: Time between group commits of transactions to disk.
- `TimeBetweenGlobalCheckpointsTimeout`: Minimum timeout for group commit of transactions to disk.
- `TimeBetweenInactiveTransactionAbortCheck`: Time between checks for inactive transactions.
- `TimeBetweenLocalCheckpoints`: Time between taking snapshots of database (expressed in base-2 logarithm of bytes).
- `TimeBetweenWatchDogCheck`: Time between execution checks inside data node.
- `TimeBetweenWatchDogCheckInitial`: Time between execution checks inside data node (early start phases when memory is allocated).
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers..
- `TransactionBufferMemory`: Dynamic buffer space (in bytes) for key and attribute data allocated for each data node.
- `TransactionDeadlockDetectionTimeout`: Time transaction can spend executing within data node. This is time that transaction coordinator waits for each data node participating in transaction to execute request. If data node takes more than this amount of time, transaction is aborted.
- `TransactionInactiveTimeout`: Milliseconds that application waits before executing another part of transaction. This is time transaction coordinator waits for application to execute or send another part (query, statement) of transaction. If application takes too much time, then transaction is aborted. Timeout = 0 means that application never times out.
- `TransactionMemory`: Memory allocated for transactions on each data node.
- `TwoPassInitialNodeRestartCopy`: Copy data in 2 passes during initial node restart, which enables multithreaded building of ordered indexes for such restarts.
- `UndoDataBuffer`: Unused; has no effect.
- `UndoIndexBuffer`: Unused; has no effect.
- `UseShm`: Use shared memory connections between this data node and API node also running on this host.
- `WatchDogImmediateKill`: When true, threads are immediately killed whenever watchdog issues occur; used for testing and debugging.
- `AutomaticThreadConfig`: Use automatic thread configuration; overrides any settings for ThreadConfig and MaxNoOfExecutionThreads, and disables ClassicFragmentation.

- `ClassicFragmentation`: When true, use traditional table fragmentation; set false to enable flexible distribution of fragments among LDMs. Disabled by AutomaticThreadConfig.
- `EnableMultithreadedBackup`: Enable multi-threaded backup.
- `MaxNoOfExecutionThreads`: For ndbmtd only, specify maximum number of execution threads.
- `MaxSendDelay`: Maximum number of microseconds to delay sending by ndbmtd.
- `NoOfFragmentLogParts`: Number of redo log file groups belonging to this data node.
- `NumCPUs`: Specify number of CPUs to use with AutomaticThreadConfig.
- `PartitionsPerNode`: Determines the number of table partitions created on each data node; not used if ClassicFragmentation is enabled.
- `ThreadConfig`: Used for configuration of multithreaded data nodes (ndbmtd). Default is empty string; see documentation for syntax and other information.

A.3 API Node Configuration Parameters

- `ApiVerbose`: Enable NDB API debugging; for NDB development.
- `ArbitrationDelay`: When asked to arbitrate, arbitrator waits this many milliseconds before voting.
- `ArbitrationRank`: If 0, then API node is not arbitrator. Kernel selects arbitrators in order 1, 2.
- `AutoReconnect`: Specifies whether an API node should reconnect fully when disconnected from cluster.
- `BatchByteSize`: Default batch size in bytes.
- `BatchSize`: Default batch size in number of records.
- `ConnectBackoffMaxTime`: Specifies longest time in milliseconds (~100ms resolution) to allow between connection attempts to any given data node by this API node. Excludes time elapsed while connection attempts are ongoing, which in worst case can take several seconds. Disable by setting to 0. If no data nodes are currently connected to this API node, StartConnectBackoffMaxTime is used instead.
- `ConnectionMap`: Specifies which data nodes to connect.
- `DefaultHashMapSize`: Set size (in buckets) to use for table hash maps. Three values are supported: 0, 240, and 3840.
- `DefaultOperationRedoProblemAction`: How operations are handled in event that RedoOverCommitCounter is exceeded.
- `ExecuteOnComputer`: String referencing earlier defined COMPUTER.
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `HeartbeatThreadPriority`: Set heartbeat thread policy and priority for API nodes; see manual for allowed values.
- `HostName`: Host name or IP address for this SQL or API node.
- `Id`: Number identifying MySQL server or API node (Id). Now deprecated; use NodeId instead.
- `LocationDomainId`: Assign this API node to specific availability domain or zone. 0 (default) leaves this unset.

- MaxScanBatchSize: Maximum collective batch size for one scan.
- NodeId: Number uniquely identifying SQL node or API node among all nodes in cluster.
- StartConnectBackoffMaxTime: Same as ConnectBackoffMaxTime except that this parameter is used in its place if no data nodes are connected to this API node.
- TotalSendBufferMemory: Total memory to use for all transporter send buffers.
- wan: Use WAN TCP setting as default.

A.4 Other Node Configuration Parameters

- HostName: Host name or IP address of this computer.
- Id: Unique identifier for this computer.
- AllowUnresolvedHostNames: When false (default), failure by management node to resolve host name results in fatal error; when true, unresolved host names are reported as warnings only.
- Checksum: If checksum is enabled, all signals between nodes are checked for errors.
- Group: Used for group proximity; smaller value is interpreted as being closer.
- HostName1: Name or IP address of first of two computers joined by TCP connection.
- HostName2: Name or IP address of second of two computers joined by TCP connection.
- NodeId1: ID of node (data node, API node, or management node) on one side of connection.
- NodeId2: ID of node (data node, API node, or management node) on one side of connection.
- NodeIdServer: Set server side of TCP connection.
- OverloadLimit: When more than this many unsent bytes are in send buffer, connection is considered overloaded.
- PreferIPVersion: Indicate DNS resolver preference for IP version 4 or 6.
- PreSendChecksum: If this parameter and Checksum are both enabled, perform pre-send checksum checks, and check all TCP signals between nodes for errors.
- Proxy: ....
- ReceiveBufferMemory: Bytes of buffer for signals received by this node.
- RequireLinkTls: Read-only; is set to true if either endpoint of this connection requires TLS.
- SendBufferMemory: Bytes of TCP buffer for signals sent from this node.
- SendSignalId: Sends ID in each signal. Used in trace files. Defaults to true in debug builds.
- TcpSpinTime: Time to spin before going to sleep when receiving.
- TCP_MAXSEG_SIZE: Value used for TCP_MAXSEG.
- TCP_RCV_BUF_SIZE: Value used for SO_RCVBUF.
- TCP_SND_BUF_SIZE: Value used for SO_SNDBUF.
- TcpBind_INADDR_ANY: Bind InAddrAny instead of host name for server part of connection.
- Checksum: If checksum is enabled, all signals between nodes are checked for errors.

- `Group`: Used for group proximity; smaller value is interpreted as being closer.
- `HostName1`: Name or IP address of first of two computers joined by SHM connection.
- `HostName2`: Name or IP address of second of two computers joined by SHM connection.
- `NodeId1`: ID of node (data node, API node, or management node) on one side of connection.
- `NodeId2`: ID of node (data node, API node, or management node) on one side of connection.
- `NodeIdServer`: Set server side of SHM connection.
- `OverloadLimit`: When more than this many unsent bytes are in send buffer, connection is considered overloaded.
- `PreSendChecksum`: If this parameter and Checksum are both enabled, perform pre-send checksum checks, and check all SHM signals between nodes for errors.
- `SendBufferMemory`: Bytes in shared memory buffer for signals sent from this node.
- `SendSignalId`: Sends ID in each signal. Used in trace files.
- `ShmKey`: Shared memory key; when set to 1, this is calculated by NDB.
- `ShmSpinTime`: When receiving, number of microseconds to spin before sleeping.
- `ShmSize`: Size of shared memory segment.
- `Signum`: Signal number to be used for signalling.

A.5 MySQL Server Option and Variable Reference for MySQL Cluster

- `Com_show_ndb_status`: Count of SHOW NDB STATUS statements.
- `Handler_discover`: Number of times that tables have been discovered.
- `ndb-applier-allow-skip-epoch`: Lets replication applier skip epochs.
- `ndb-batch-size`: Size (in bytes) to use for NDB transaction batches.
- `ndb-blob-read-batch-bytes`: Specifies size in bytes that large BLOB reads should be batched into. 0 = no limit.
- `ndb-blob-write-batch-bytes`: Specifies size in bytes that large BLOB writes should be batched into. 0 = no limit.
- `ndb-cluster-connection-pool`: Number of connections to cluster used by MySQL.
- `ndb-cluster-connection-pool-nodeids`: Comma-separated list of node IDs for connections to cluster used by MySQL; number of nodes in list must match value set for --ndb-cluster-connection-pool.
- `ndb-connectstring`: Address of NDB management server distributing configuration information for this cluster.
- `ndb-default-column-format`: Use this value (FIXED or DYNAMIC) by default for COLUMN_FORMAT and ROW_FORMAT options when creating or adding table columns.
- `ndb-deferred-constraints`: Specifies that constraint checks on unique indexes (where these are supported) should be deferred until commit time. Not normally needed or used; for testing purposes only.

- `ndb-distribution`: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH).
- `ndb-log-apply-status`: Cause MySQL server acting as replica to log mysql.ndb_apply_status updates received from its immediate source in its own binary log, using its own server ID. Effective only if server is started with --ndbcluster option.
- `ndb-log-empty-epochs`: When enabled, causes epochs in which there were no changes to be written to ndb_apply_status and ndb_binlog_index tables, even when --log-slave-updates is enabled.
- `ndb-log-empty-update`: When enabled, causes updates that produced no changes to be written to ndb_apply_status and ndb_binlog_index tables, even when --log-slave-updates is enabled.
- `ndb-log-exclusive-reads`: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts.
- `ndb-log-fail-terminate`: Terminate mysqld process if complete logging of all found row events is not possible.
- `ndb-log-orig`: Log originating server id and epoch in mysql.ndb_binlog_index table.
- `ndb-log-row-slice-count`: Number of slices to be calculated by this server when subscribing to NDB table change event streams used for writing binary logs.
- `ndb-log-row-slice-id`: ID of virtual slice (of NDB table change event streams) subscribed to by this server.
- `ndb-log-transaction-dependency`: Make binary log thread calculate transaction dependencies for every transaction it writes to binary log.
- `ndb-log-transaction-id`: Write NDB transaction IDs in binary log. Requires --log-bin-v1-events=OFF.
- `ndb-log-update-minimal`: Log updates in minimal format.
- `ndb-log-updated-only`: Log updates only (ON) or complete rows (OFF).
- `ndb-log-update-as-write`: Toggles logging of updates on source between updates (OFF) and writes (ON).
- `ndb-mgm-tls`: Whether TLS connection requirements are strict or relaxed.
- `ndb-mgmd-host`: Set host (and port, if desired) for connecting to management server.
- `ndb-nodeid`: NDB Cluster node ID for this MySQL server.
- `ndb-optimized-node-selection`: Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable.
- `ndb-tls-search-path`: Directories to search for NDB TLS CAs and private keys.
- `ndb-transid-mysql-connection-map`: Enable or disable ndb_transid_mysql_connection_map plugin; that is, enable or disable INFORMATION_SCHEMA table having that name.
- `ndb-wait-connected`: Time (in seconds) for MySQL server to wait for connection to cluster management and data nodes before accepting MySQL client connections.
- `ndb-wait-setup`: Time (in seconds) for MySQL server to wait for NDB engine setup to complete.
- `ndb-allow-copying-alter-table`: Set to OFF to keep ALTER TABLE from using copying operations on NDB tables.
- `Ndb_api_adaptive_send_deferred_count`: Number of adaptive send calls not actually sent by this MySQL Server (SQL node).

- `Ndb_api_adaptive_send_deferred_count_session`: Number of adaptive send calls not actually sent in this client session.
- `Ndb_api_adaptive_send_deferred_count_replica`: Number of adaptive send calls not actually sent by this replica.
- `Ndb_api_adaptive_send_deferred_count_slave`: Number of adaptive send calls not actually sent by this replica.
- `Ndb_api_adaptive_send_forced_count`: Number of adaptive sends with forced-send set sent by this MySQL Server (SQL node).
- `Ndb_api_adaptive_send_forced_count_session`: Number of adaptive sends with forced-send set in this client session.
- `Ndb_api_adaptive_send_forced_count_replica`: Number of adaptive sends with forced-send set sent by this replica.
- `Ndb_api_adaptive_send_forced_count_slave`: Number of adaptive sends with forced-send set sent by this replica.
- `Ndb_api_adaptive_send_unforced_count`: Number of adaptive sends without forced-send sent by this MySQL Server (SQL node).
- `Ndb_api_adaptive_send_unforced_count_session`: Number of adaptive sends without forced-send in this client session.
- `Ndb_api_adaptive_send_unforced_count_replica`: Number of adaptive sends without forced-send sent by this replica.
- `Ndb_api_adaptive_send_unforced_count_slave`: Number of adaptive sends without forced-send sent by this replica.
- `Ndb_api_bytes_received_count`: Quantity of data (in bytes) received from data nodes by this MySQL Server (SQL node).
- `Ndb_api_bytes_received_count_session`: Quantity of data (in bytes) received from data nodes in this client session.
- `Ndb_api_bytes_received_count_replica`: Quantity of data (in bytes) received from data nodes by this replica.
- `Ndb_api_bytes_received_count_slave`: Quantity of data (in bytes) received from data nodes by this replica.
- `Ndb_api_bytes_sent_count`: Quantity of data (in bytes) sent to data nodes by this MySQL Server (SQL node).
- `Ndb_api_bytes_sent_count_session`: Quantity of data (in bytes) sent to data nodes in this client session.
- `Ndb_api_bytes_sent_count_replica`: Qunatity of data (in bytes) sent to data nodes by this replica.
- `Ndb_api_bytes_sent_count_slave`: Qunatity of data (in bytes) sent to data nodes by this replica.
- `Ndb_api_event_bytes_count`: Number of bytes of events received by this MySQL Server (SQL node).
- `Ndb_api_event_bytes_count_injector`: Number of bytes of event data received by NDB binary log injector thread.

- Ndb_api_event_data_count: Number of row change events received by this MySQL Server (SQL node).
- Ndb_api_event_data_count_injector: Number of row change events received by NDB binary log injector thread.
- Ndb_api_event_nondata_count: Number of events received, other than row change events, by this MySQL Server (SQL node).
- Ndb_api_event_nondata_count_injector: Number of events received, other than row change events, by NDB binary log injector thread.
- Ndb_api_pk_op_count: Number of operations based on or using primary keys by this MySQL Server (SQL node).
- Ndb_api_pk_op_count_session: Number of operations based on or using primary keys in this client session.
- Ndb_api_pk_op_count_replica: Number of operations based on or using primary keys by this replica.
- Ndb_api_pk_op_count_slave: Number of operations based on or using primary keys by this replica.
- Ndb_api_pruned_scan_count: Number of scans that have been pruned to one partition by this MySQL Server (SQL node).
- Ndb_api_pruned_scan_count_session: Number of scans that have been pruned to one partition in this client session.
- Ndb_api_pruned_scan_count_replica: Number of scans that have been pruned to one partition by this replica.
- Ndb_api_pruned_scan_count_slave: Number of scans that have been pruned to one partition by this replica.
- Ndb_api_range_scan_count: Number of range scans that have been started by this MySQL Server (SQL node).
- Ndb_api_range_scan_count_session: Number of range scans that have been started in this client session.
- Ndb_api_range_scan_count_replica: Number of range scans that have been started by this replica.
- Ndb_api_range_scan_count_slave: Number of range scans that have been started by this replica.
- Ndb_api_read_row_count: Total number of rows that have been read by this MySQL Server (SQL node).
- Ndb_api_read_row_count_session: Total number of rows that have been read in this client session.
- Ndb_api_read_row_count_replica: Total number of rows that have been read by this replica.
- Ndb_api_read_row_count_slave: Total number of rows that have been read by this replica.
- Ndb_api_scan_batch_count: Number of batches of rows received by this MySQL Server (SQL node).
- Ndb_api_scan_batch_count_session: Number of batches of rows received in this client session.

- Ndb_api_scan_batch_count_replica: Number of batches of rows received by this replica.
- Ndb_api_scan_batch_count_slave: Number of batches of rows received by this replica.
- Ndb_api_table_scan_count: Number of table scans that have been started, including scans of internal tables, by this MySQL Server (SQL node).
- Ndb_api_table_scan_count_session: Number of table scans that have been started, including scans of internal tables, in this client session.
- Ndb_api_table_scan_count_replica: Number of table scans that have been started, including scans of internal tables, by this replica.
- Ndb_api_table_scan_count_slave: Number of table scans that have been started, including scans of internal tables, by this replica.
- Ndb_api_trans_abort_count: Number of transactions aborted by this MySQL Server (SQL node).
- Ndb_api_trans_abort_count_session: Number of transactions aborted in this client session.
- Ndb_api_trans_abort_count_replica: Number of transactions aborted by this replica.
- Ndb_api_trans_abort_count_slave: Number of transactions aborted by this replica.
- Ndb_api_trans_close_count: Number of transactions closed by this MySQL Server (SQL node); may be greater than sum of TransCommitCount and TransAbortCount.
- Ndb_api_trans_close_count_session: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) in this client session.
- Ndb_api_trans_close_count_replica: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) by this replica.
- Ndb_api_trans_close_count_slave: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) by this replica.
- Ndb_api_trans_commit_count: Number of transactions committed by this MySQL Server (SQL node).
- Ndb_api_trans_commit_count_session: Number of transactions committed in this client session.
- Ndb_api_trans_commit_count_replica: Number of transactions committed by this replica.
- Ndb_api_trans_commit_count_slave: Number of transactions committed by this replica.
- Ndb_api_trans_local_read_row_count: Total number of rows that have been read by this MySQL Server (SQL node).
- Ndb_api_trans_local_read_row_count_session: Total number of rows that have been read in this client session.
- Ndb_api_trans_local_read_row_count_replica: Total number of rows that have been read by this replica.
- Ndb_api_trans_local_read_row_count_slave: Total number of rows that have been read by this replica.
- Ndb_api_trans_start_count: Number of transactions started by this MySQL Server (SQL node).
- Ndb_api_trans_start_count_session: Number of transactions started in this client session.

- Ndb_api_trans_start_count_replica: Number of transactions started by this replica.
- Ndb_api_trans_start_count_slave: Number of transactions started by this replica.
- Ndb_api_uk_op_count: Number of operations based on or using unique keys by this MySQL Server (SQL node).
- Ndb_api_uk_op_count_session: Number of operations based on or using unique keys in this client session.
- Ndb_api_uk_op_count_replica: Number of operations based on or using unique keys by this replica.
- Ndb_api_uk_op_count_slave: Number of operations based on or using unique keys by this replica.
- Ndb_api_wait_exec_complete_count: Number of times thread has been blocked while waiting for operation execution to complete by this MySQL Server (SQL node).
- Ndb_api_wait_exec_complete_count_session: Number of times thread has been blocked while waiting for operation execution to complete in this client session.
- Ndb_api_wait_exec_complete_count_replica: Number of times thread has been blocked while waiting for operation execution to complete by this replica.
- Ndb_api_wait_exec_complete_count_slave: Number of times thread has been blocked while waiting for operation execution to complete by this replica.
- Ndb_api_wait_meta_request_count: Number of times thread has been blocked waiting for metadata-based signal by this MySQL Server (SQL node).
- Ndb_api_wait_meta_request_count_session: Number of times thread has been blocked waiting for metadata-based signal in this client session.
- Ndb_api_wait_meta_request_count_replica: Number of times thread has been blocked waiting for metadata-based signal by this replica.
- Ndb_api_wait_meta_request_count_slave: Number of times thread has been blocked waiting for metadata-based signal by this replica.
- Ndb_api_wait_nanos_count: Total time (in nanoseconds) spent waiting for some type of signal from data nodes by this MySQL Server (SQL node).
- Ndb_api_wait_nanos_count_session: Total time (in nanoseconds) spent waiting for some type of signal from data nodes in this client session.
- Ndb_api_wait_nanos_count_replica: Total time (in nanoseconds) spent waiting for some type of signal from data nodes by this replica.
- Ndb_api_wait_nanos_count_slave: Total time (in nanoseconds) spent waiting for some type of signal from data nodes by this replica.
- Ndb_api_wait_scan_result_count: Number of times thread has been blocked while waiting for scan-based signal by this MySQL Server (SQL node).
- Ndb_api_wait_scan_result_count_session: Number of times thread has been blocked while waiting for scan-based signal in this client session.
- Ndb_api_wait_scan_result_count_replica: Number of times thread has been blocked while waiting for scan-based signal by this replica.
- Ndb_api_wait_scan_result_count_slave: Number of times thread has been blocked while waiting for scan-based signal by this replica.

- `ndb_autoincrement_prefetch_sz`: NDB auto-increment prefetch size.
- `ndb_clear_apply_status`: Causes RESET SLAVE/RESET REPLICA to clear all rows from ndb_apply_status table; ON by default.
- `Ndb_cluster_node_id`: Node ID of this server when acting as NDB Cluster SQL node.
- `Ndb_config_from_host`: NDB Cluster management server host name or IP address.
- `Ndb_config_from_port`: Port for connecting to NDB Cluster management server.
- `Ndb_config_generation`: Generation number of the current configuration of the cluster.
- `Ndb_conflict_fn_epoch`: Number of rows that have been found in conflict by NDB$EPOCH() NDB replication conflict detection function.
- `Ndb_conflict_fn_epoch2`: Number of rows that have been found in conflict by NDB replication NDB$EPOCH2() conflict detection function.
- `Ndb_conflict_fn_epoch2_trans`: Number of rows that have been found in conflict by NDB replication NDB$EPOCH2_TRANS() conflict detection function.
- `Ndb_conflict_fn_epoch_trans`: Number of rows that have been found in conflict by NDB $EPOCH_TRANS() conflict detection function.
- `Ndb_conflict_fn_max`: Number of times that NDB replication conflict resolution based on "greater timestamp wins" has been applied to update and delete operations.
- `Ndb_conflict_fn_max_del_win`: Number of times that NDB replication conflict resolution based on outcome of NDB$MAX_DELETE_WIN() has been applied to update and delete operations.
- `Ndb_conflict_fn_max_ins`: Number of times that NDB replication conflict resolution based on "greater timestamp wins" has been applied to insert operations.
- `Ndb_conflict_fn_max_del_win_ins`: Number of times that NDB replication conflict resolution based on outcome of NDB$MAX_DEL_WIN_INS() has been applied to insert operations.
- `Ndb_conflict_fn_old`: Number of times that NDB replication "same timestamp wins" conflict resolution has been applied.
- `Ndb_conflict_last_conflict_epoch`: Most recent NDB epoch on this replica in which some conflict was detected.
- `Ndb_conflict_last_stable_epoch`: Most recent epoch containing no conflicts.
- `Ndb_conflict_reflected_op_discard_count`: Number of reflected operations that were not applied due error during execution.
- `Ndb_conflict_reflected_op_prepare_count`: Number of reflected operations received that have been prepared for execution.
- `Ndb_conflict_refresh_op_count`: Number of refresh operations that have been prepared.
- `ndb_conflict_role`: Role for replica to play in conflict detection and resolution. Value is one of PRIMARY, SECONDARY, PASS, or NONE (default). Can be changed only when replication SQL thread is stopped. See documentation for further information.
- `Ndb_conflict_trans_conflict_commit_count`: Number of epoch transactions committed after requiring transactional conflict handling.
- `Ndb_conflict_trans_detect_iter_count`: Number of internal iterations required to commit epoch transaction. Should be (slightly) greater than or equal to Ndb_conflict_trans_conflict_commit_count.

- Ndb_conflict_trans_reject_count: Number of transactions rejected after being found in conflict by transactional conflict function.
- Ndb_conflict_trans_row_conflict_count: Number of rows found in conflict by transactional conflict function. Includes any rows included in or dependent on conflicting transactions.
- Ndb_conflict_trans_row_reject_count: Total number of rows realigned after being found in conflict by transactional conflict function. Includes Ndb_conflict_trans_row_conflict_count and any rows included in or dependent on conflicting transactions.
- ndb_data_node_neighbour: Specifies cluster data node "closest" to this MySQL Server, for transaction hinting and fully replicated tables.
- ndb_default_column_format: Sets default row format and column format (FIXED or DYNAMIC) used for new NDB tables.
- ndb_deferred_constraints: Specifies that constraint checks should be deferred (where these are supported). Not normally needed or used; for testing purposes only.
- ndb_dbg_check_shares: Check for any lingering shares (debug builds only).
- ndb-schema-dist-timeout: How long to wait before detecting timeout during schema distribution.
- ndb_distribution: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH).
- Ndb_epoch_delete_delete_count: Number of delete-delete conflicts detected (delete operation is applied, but row does not exist).
- ndb_eventbuffer_free_percent: Percentage of free memory that should be available in event buffer before resumption of buffering, after reaching limit set by ndb_eventbuffer_max_alloc.
- ndb_eventbuffer_max_alloc: Maximum memory that can be allocated for buffering events by NDB API. Defaults to 0 (no limit).
- ndb_extra_logging: Controls logging of NDB Cluster schema, connection, and data distribution events in MySQL error log.
- Ndb_fetch_table_stats: Number of times table statistics were fetched from tables rather than cache.
- ndb_force_send: Forces sending of buffers to NDB immediately, without waiting for other threads.
- ndb_fully_replicated: Whether new NDB tables are fully replicated.
- ndb_index_stat_enable: Use NDB index statistics in query optimization.
- ndb_index_stat_option: Comma-separated list of tunable options for NDB index statistics; list should contain no spaces.
- ndb_join_pushdown: Enables pushing down of joins to data nodes.
- Ndb_last_commit_epoch_server: Epoch most recently committed by NDB.
- Ndb_last_commit_epoch_session: Epoch most recently committed by this NDB client.
- ndb_log_apply_status: Whether or not MySQL server acting as replica logs mysql.ndb_apply_status updates received from its immediate source in its own binary log, using its own server ID.
- ndb_log_bin: Write updates to NDB tables in binary log. Effective only if binary logging is enabled with --log-bin.

- ndb_log_binlog_index: Insert mapping between epochs and binary log positions into ndb_binlog_index table. Defaults to ON. Effective only if binary logging is enabled.
- ndb_log_cache_size: Set size of transaction cache used for recording NDB binary log.
- ndb_log_empty_epochs: When enabled, epochs in which there were no changes are written to ndb_apply_status and ndb_binlog_index tables, even when log_replica_updates or log_slave_updates is enabled.
- ndb_log_empty_update: When enabled, updates which produce no changes are written to ndb_apply_status and ndb_binlog_index tables, even when log_replica_updates or log_slave_updates is enabled.
- ndb_log_exclusive_reads: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts.
- ndb_log_orig: Whether id and epoch of originating server are recorded in mysql.ndb_binlog_index table. Set using --ndb-log-orig option when starting mysqld.
- ndb_log_transaction_id: Whether NDB transaction IDs are written into binary log (Read-only).
- ndb_log_transaction_compression: Whether to compress NDB binary log; can also be enabled on startup by enabling --binlog-transaction-compression option.
- ndb_log_transaction_compression_level_zstd: The ZSTD compression level to use when writing compressed transactions to the NDB binary log.
- ndb_metadata_check: Enable auto-detection of NDB metadata changes with respect to MySQL data dictionary; enabled by default.
- ndb_metadata_check_interval: Interval in seconds to perform check for NDB metadata changes with respect to MySQL data dictionary.
- Ndb_metadata_detected_count: Number of times NDB metadata change monitor thread has detected changes.
- Ndb_metadata_excluded_count: Number of NDB metadata objects that NDB binlog thread has failed to synchronize.
- ndb_metadata_sync: Triggers immediate synchronization of all changes between NDB dictionary and MySQL data dictionary; causes ndb_metadata_check and ndb_metadata_check_interval values to be ignored. Resets to false when synchronization is complete.
- Ndb_metadata_synced_count: Number of NDB metadata objects which have been synchronized.
- Ndb_number_of_data_nodes: Number of data nodes in this NDB cluster; set only if server participates in cluster.
- ndb-optimization-delay: Number of milliseconds to wait between processing sets of rows by OPTIMIZE TABLE on NDB tables.
- ndb_optimized_node_selection: Determines how SQL node chooses cluster data node to use as transaction coordinator.
- Ndb_pruned_scan_count: Number of scans executed by NDB since cluster was last started where partition pruning could be used.
- Ndb_pushed_queries_defined: Number of joins that API nodes have attempted to push down to data nodes.
- Ndb_pushed_queries_dropped: Number of joins that API nodes have tried to push down, but failed.

- Ndb_pushed_queries_executed: Number of joins successfully pushed down and executed on data nodes.
- Ndb_pushed_reads: Number of reads executed on data nodes by pushed-down joins.
- ndb_read_backup: Enable read from any replica for all NDB tables; use NDB_TABLE=READ_BACKUP={0|1} with CREATE TABLE or ALTER TABLE to enable or disable for individual NDB tables.
- ndb_recv_thread_activation_threshold: Activation threshold when receive thread takes over polling of cluster connection (measured in concurrently active threads).
- ndb_recv_thread_cpu_mask: CPU mask for locking receiver threads to specific CPUs; specified as hexadecimal. See documentation for details.
- Ndb_replica_max_replicated_epoch: Most recently committed NDB epoch on this replica. When this value is greater than or equal to Ndb_conflict_last_conflict_epoch, no conflicts have yet been detected.
- ndb_replica_batch_size: Batch size in bytes for replica applier.
- ndb_report_thresh_binlog_epoch_slip: NDB 7.5 and later: Threshold for number of epochs completely buffered, but not yet consumed by binlog injector thread which when exceeded generates BUFFERED_EPOCHS_OVER_THRESHOLD event buffer status message; prior to NDB 7.5: Threshold for number of epochs to lag behind before reporting binary log status.
- ndb_report_thresh_binlog_mem_usage: Threshold for percentage of free memory remaining before reporting binary log status.
- ndb_row_checksum: When enabled, set row checksums; enabled by default.
- Ndb_scan_count: Total number of scans executed by NDB since cluster was last started.
- ndb_schema_dist_lock_wait_timeout: Time during schema distribution to wait for lock before returning error.
- ndb_schema_dist_timeout: Time to wait before detecting timeout during schema distribution.
- ndb_schema_dist_upgrade_allowed: Allow schema distribution table upgrade when connecting to NDB.
- Ndb_schema_participant_count: Number of MySQL servers participating in NDB schema change distribution.
- ndb_show_foreign_key_mock_tables: Show mock tables used to support foreign_key_checks=0.
- ndb_slave_conflict_role: Role for replica to play in conflict detection and resolution. Value is one of PRIMARY, SECONDARY, PASS, or NONE (default). Can be changed only when replication SQL thread is stopped. See documentation for further information.
- Ndb_slave_max_replicated_epoch: Most recently committed NDB epoch on this replica. When this value is greater than or equal to Ndb_conflict_last_conflict_epoch, no conflicts have yet been detected.
- Ndb_system_name: Configured cluster system name; empty if server not connected to NDB.
- ndb_table_no_logging: NDB tables created when this setting is enabled are not checkpointed to disk (although table schema files are created). Setting in effect when table is created with or altered to use NDBCLUSTER persists for table's lifetime.
- ndb_table_temporary: NDB tables are not persistent on disk: no schema files are created and tables are not logged.

- `Ndb_trans_hint_count_session`: Number of transactions using hints that have been started in this session.
- `ndb_use_copying_alter_table`: Use copying ALTER TABLE operations in NDB Cluster.
- `ndb_use_exact_count`: Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query.
- `ndb_use_transactions`: Set to OFF, to disable transaction support by NDB. Not recommended except in certain special cases; see documentation for details.
- `ndb_version`: Shows build and NDB engine version as an integer.
- `ndb_version_string`: Shows build information including NDB engine version in ndb-x.y.z format.
- `ndbcluster`: Enable NDB Cluster (if this version of MySQL supports it). Disabled by `--skip-ndbcluster`.
- `ndbinfo`: Enable ndbinfo plugin, if supported.
- `ndbinfo_database`: Name used for NDB information database; read only.
- `ndbinfo_max_bytes`: Used for debugging only.
- `ndbinfo_max_rows`: Used for debugging only.
- `ndbinfo_offline`: Put ndbinfo database into offline mode, in which no rows are returned from tables or views.
- `ndbinfo_show_hidden`: Whether to show ndbinfo internal base tables in mysql client; default is OFF.
- `ndbinfo_table_prefix`: Prefix to use for naming ndbinfo internal base tables; read only.
- `ndbinfo_version`: ndbinfo engine version; read only.
- `replica_allow_batching`: Turns update batching on and off for replica.
- `server_id_bits`: Number of least significant bits in server_id actually used for identifying server, permitting NDB API applications to store application data in most significant bits. server_id must be less than 2 to power of this value.
- `skip-ndbcluster`: Disable NDB Cluster storage engine.
- `slave_allow_batching`: Turns update batching on and off for replica.
- `transaction_allow_batching`: Allows batching of statements within one transaction. Disable AUTOCOMMIT to use.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X