MySQL and Linux/Unix

Abstract

This is the MySQL Linux extract from the MySQL 8.0 Reference Manual.

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: 2024-02-17 (revision: 77859)

Table of Contents

Preface and Legal Notices

This is the MySQL Linux extract from the MySQL 8.0 Reference Manual.

Licensing information—MySQL 8.0. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL 8.0, see the MySQL 8.0 Commercial Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL 8.0, see the MySQL 8.0 Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Legal Notices

License Restrictions

Warranty Disclaimer

Restricted Rights Notice

Hazardous Applications Notice

Trademark Notice

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 Installing MySQL on Unix/Linux Using Generic Binaries

Oracle provides a set of binary distributions of MySQL. These include generic binary distributions in the form of compressed `tar` files (files with a `.tar.xz` extension) for a number of platforms, and binaries in platform-specific package formats for selected platforms.

This section covers the installation of MySQL from a compressed `tar` file binary distribution on Unix/Linux platforms. For Linux-generic binary distribution installation instructions with a focus on MySQL security features, refer to the Secure Deployment Guide. For other platform-specific binary package formats, see the other platform-specific sections in this manual. For example, for Windows distributions, see Installing MySQL on Microsoft Windows. See How to Get MySQL on how to obtain MySQL in different distribution formats.

MySQL compressed `tar` file binary distributions have names of the form `mysql-VERSION-OS.tar.xz`, where *`VERSION`* is a number (for example, `8.0.36`), and *`OS`* indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

There is also a “minimal install” version of the MySQL compressed `tar` file for the Linux generic binary distribution, which has a name of the form `mysql-VERSION-OS-GLIBCVER-ARCH-minimal.tar.xz`. The minimal install distribution excludes debug binaries and is stripped of debug symbols, making it significantly smaller than the regular binary distribution. If you choose to install the minimal install distribution, remember to adjust for the difference in file name format in the instructions that follow.

Warnings

- If you have previously installed MySQL using your operating system native package management system, such as Yum or APT, you may experience problems installing using a native binary. Make sure your previous MySQL installation has been removed entirely (using your package management system), and that any additional files, such as old versions of your data files, have also been removed. You should also check for configuration files such as `/etc/my.cnf` or the `/etc/mysql` directory and delete them.

  For information about replacing third-party packages with official MySQL packages, see the related APT guide or Yum guide.

- MySQL has a dependency on the `libaio` library. Data directory initialization and subsequent server startup steps fail if this library is not installed locally. If necessary, install it using the appropriate package manager. For example, on Yum-based systems:

  ```
  $> yum search libaio  # search for info
  $> yum install libaio # install library
  ```

  Or, on APT-based systems:

  ```
  $> apt-cache search libaio # search for info
  $> apt-get install libaio1 # install library
  ```

- **Oracle Linux 8 / Red Hat 8** (EL8): These platforms by default do not install the file `/lib64/libtinfo.so.5`, which is required by the MySQL client `bin/mysql` for packages `mysql-VERSION-el7-x86_64.tar.gz` and `mysql-VERSION-linux-glibc2.12-x86_64.tar.xz`. To work around this issue, install the `ncurses-compat-libs` package:

  ```
  $> yum install ncurses-compat-libs
  ```

To install a compressed `tar` file binary distribution, unpack it at the installation location you choose (typically `/usr/local/mysql`). This creates the directories shown in the following table.

Table 1.1 MySQL Installation Layout for Generic Unix/Linux Binary Package

Directory	Contents of Directory
`bin`	`mysqld` server, client and utility programs
`docs`	MySQL manual in Info format
`man`	Unix manual pages
`include`	Include (header) files
`lib`	Libraries
`share`	Error messages, dictionary, and SQL for database installation
`support-files`	Miscellaneous support files

Debug versions of the `mysqld` binary are available as `mysqld-debug`. To compile your own debug version of MySQL from a source distribution, use the appropriate configuration options to enable debugging support. See Installing MySQL from Source.

To install and use a MySQL binary distribution, the command sequence looks like this:

```
$> groupadd mysql
$> useradd -r -g mysql -s /bin/false mysql
$> cd /usr/local
$> tar xvf /path/to/mysql-VERSION-OS.tar.xz
$> ln -s full-path-to-mysql-VERSION-OS mysql
$> cd mysql
$> mkdir mysql-files
$> chown mysql:mysql mysql-files
$> chmod 750 mysql-files
$> bin/mysqld --initialize --user=mysql
$> bin/mysql_ssl_rsa_setup
$> bin/mysqld_safe --user=mysql &
# Next command is optional
$> cp support-files/mysql.server /etc/init.d/mysql.server
```

> **Note**
>
> This procedure assumes that you have `root` (administrator) access to your system. Alternatively, you can prefix each command using the `sudo` (Linux) or `pfexec` (Solaris) command.

The `mysql-files` directory provides a convenient location to use as the value for the `secure_file_priv` system variable, which limits import and export operations to a specific directory. See Server System Variables.

A more detailed version of the preceding description for installing a binary distribution follows.

Create a mysql User and Group

If your system does not already have a user and group to use for running `mysqld`, you may need to create them. The following commands add the `mysql` group and the `mysql` user. You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following instructions. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix/Linux, or they may have different names such as `adduser` and `addgroup`.

```
$> groupadd mysql
$> useradd -r -g mysql -s /bin/false mysql
```

> **Note**
>
> Because the user is required only for ownership purposes, not login purposes, the `useradd` command uses the `-r` and `-s /bin/false` options to create a user that does not have login permissions to your server host. Omit these options if your `useradd` does not support them.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it. The example here unpacks the distribution under `/usr/local`. The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as `root`.

```
$> cd /usr/local
```

Obtain a distribution file using the instructions in How to Get MySQL. For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

Unpack the distribution, which creates the installation directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

```
$> tar xvf /path/to/mysql-VERSION-OS.tar.xz
```

The `tar` command creates a directory named `mysql-VERSION-OS`.

To install MySQL from a compressed `tar` file binary distribution, your system must have GNU `XZ Utils` to uncompress the distribution and a reasonable `tar` to unpack it.

> **Note**
>
> The compression algorithm changed from Gzip to XZ in MySQL Server 8.0.12; and the generic binary's file extension changed from .tar.gz to .tar.xz.

GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.

If your `tar` does not support the `xz` format then use the `xz` command to unpack the distribution and `tar` to unpack it. Replace the preceding `tar` command with the following alternative command to uncompress and extract the distribution:

```
$> xz -dc /path/to/mysql-VERSION-OS.tar.xz | tar x
```

Next, create a symbolic link to the installation directory created by `tar`:

```
$> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `ln` command makes a symbolic link to the installation directory. This enables you to refer more easily to it as `/usr/local/mysql`. To avoid having to type the path name of client programs always when you are working with MySQL, you can add the `/usr/local/mysql/bin` directory to your `PATH` variable:

```
$> export PATH=$PATH:/usr/local/mysql/bin
```

Perform Postinstallation Setup

The remainder of the installation process involves setting distribution ownership and access permissions, initializing the data directory, starting the MySQL server, and setting up the configuration file. For instructions, see Postinstallation Setup and Testing.

Chapter 2 Installing MySQL on Linux

Table of Contents

Linux supports a number of different solutions for installing MySQL. We recommend that you use one of the distributions from Oracle, for which several methods for installation are available:

Table 2.1 Linux Installation Methods and Information

Type	Setup Method	Additional Information
Apt	Enable the MySQL Apt repository	Documentation
Yum	Enable the MySQL Yum repository	Documentation
Zypper	Enable the MySQL SLES repository	Documentation
RPM	Download a specific package	Documentation
DEB	Download a specific package	Documentation
Generic	Download a generic package	Documentation
Source	Compile from source	Documentation
Docker	Use the Oracle Container Registry. You can also use My Oracle Support for the MySQL Enterprise Edition.	Documentation
Oracle Unbreakable Linux Network	Use ULN channels	Documentation

As an alternative, you can use the package manager on your system to automatically download and install MySQL with packages from the native software repositories of your Linux distribution. These native packages are often several versions behind the currently available release. You are also normally unable to install innovation releases, since these are not usually made available in the native repositories. For more information on using the native package installers, see Section 2.7, “Installing MySQL on Linux from the Native Software Repositories”.

> **Note**
>
> For many Linux installations, you want to set up MySQL to be started automatically when your machine starts. Many of the native package installations perform this operation for you, but for source, binary and RPM solutions you may need to set this up separately. The required script, `mysql.server`, can be found in the `support-files` directory under the

MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See mysql.server — MySQL Server Startup Script.

2.1 Installing MySQL on Linux Using the MySQL Yum Repository

The MySQL Yum repository for Oracle Linux, Red Hat Enterprise Linux, CentOS, and Fedora provides RPM packages for installing the MySQL server, client, MySQL Workbench, MySQL Utilities, MySQL Router, MySQL Shell, Connector/ODBC, Connector/Python and so on (not all packages are available for all the distributions; see Installing Additional MySQL Products and Components with Yum for details).

Before You Start

As a popular, open-source software, MySQL, in its original or re-packaged form, is widely installed on many systems from various sources, including different software download sites, software repositories, and so on. The following instructions assume that MySQL is not already installed on your system using a third-party-distributed RPM package; if that is not the case, follow the instructions given in Upgrading MySQL with the MySQL Yum Repository or Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository.

Note

Repository setup RPM file names begin with `mysql80-community` to highlight the default active MySQL subrepository, which is MySQL 8.0 today. They also install subrepositories for the MySQL innovation track, which allows installing and upgrading the MySQL 8.3 innovation release.

Steps for a Fresh Installation of MySQL

Follow the steps below to install the latest GA version of MySQL with the MySQL Yum repository:

Adding[1] the MySQL Yum Repository

First, add the MySQL Yum repository to your system's repository list. This is a one-time operation, which can be performed by installing an RPM provided by MySQL. Follow these steps:

a. Go to the Download MySQL Yum Repository page (https://dev.mysql.com/downloads/repo/yum/) in the MySQL Developer Zone.

b. Select and download the release package for your platform.

c. Install the downloaded release package with the following command, replacing *`platform-and-version-specific-package-name`* with the name of the downloaded RPM package:

```
$> sudo yum install platform-and-version-specific-package-name.rpm
```

For an EL6-based system, the command is in the form of:

```
$> sudo yum install mysql80-community-release-el6-{version-number}.noarch.rpm
```

For an EL7-based system:

```
$> sudo yum install mysql80-community-release-el7-{version-number}.noarch.rpm
```

For an EL8-based system:

```
$> sudo yum install mysql80-community-release-el8-{version-number}.noarch.rpm
```

For an EL9-based system:

```
$> sudo yum install mysql80-community-release-el9-{version-number}.noarch.rpm
```

For Fedora 37:

```
$> sudo dnf install mysql80-community-release-fc37-{version-number}.noarch.rpm
```

For Fedora 38:

```
$> sudo dnf install mysql80-community-release-fc38-{version-number}.noarch.rpm
```

For Fedora 39:

```
$> sudo dnf install mysql80-community-release-fc39-{version-number}.noarch.rpm
```

The installation command adds the MySQL Yum repository to your system's repository list and downloads the GnuPG key to check the integrity of the software packages. See Signature Checking Using GnuPG for details on GnuPG key checking.

You can check that the MySQL Yum repository has been successfully added by the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> yum repolist enabled | grep "mysql.*-community.*"
```

> **Note**
>
> Once the MySQL Yum repository is enabled on your system, any system-wide update by the `yum update` command (or `dnf upgrade` for dnf-enabled systems) upgrades MySQL packages on your system and replaces any native third-party packages, if Yum finds replacements for them in the MySQL Yum repository; see Upgrading MySQL with the MySQL Yum Repository, for a discussion on some possible effects of that on your system, see Upgrading the Shared Client Libraries.

2 Selecting a Release Series

When using the MySQL Yum repository, the latest GA series (currently MySQL 8.0) is selected for installation by default. If this is what you want, you can skip to the next step, Installing MySQL.

Within the MySQL Yum repository, different release series of the MySQL Community Server are hosted in different subrepositories. The subrepository for the latest GA series (currently MySQL 8.0) is enabled by default, and the subrepositories for all other series (for example, the MySQL 8.0 series) are disabled by default. Use this command to see all the subrepositories in the MySQL Yum repository, and see which of them are enabled or disabled (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> yum repolist all | grep mysql
```

To install the latest release from the latest GA series, no configuration is needed. To install the latest release from a specific series other than the latest GA series, disable the subrepository for the latest GA series and enable the subrepository for the specific series before running the installation command. If your platform supports `yum-config-manager`, you can do that by issuing these commands, which disable the subrepository for the 5.7 series and enable the one for the 8.0 series:

```
$> sudo yum-config-manager --disable mysql57-community
$> sudo yum-config-manager --enable mysql80-community
```

For dnf-enabled platforms:

```
$> sudo dnf config-manager --disable mysql57-community
```

```
$> sudo dnf config-manager --enable mysql80-community
```

Besides using `yum-config-manager` or the `dnf config-manager` command, you can also select a release series by editing manually the `/etc/yum.repos.d/mysql-community.repo` file. This is a typical entry for a release series' subrepository in the file:

```
[mysql57-community]
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql-2022
       file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

Find the entry for the subrepository you want to configure, and edit the `enabled` option. Specify `enabled=0` to disable a subrepository, or `enabled=1` to enable a subrepository. For example, to install MySQL 8.0, make sure you have `enabled=0` for the above subrepository entry for MySQL 5.7, and have `enabled=1` for the entry for the 8.0 series:

```
# Enable to use MySQL 8.0
[mysql80-community]
name=MySQL 8.0 Community Server
baseurl=http://repo.mysql.com/yum/mysql-8.0-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql-2022
       file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

You should only enable subrepository for one release series at any time. When subrepositories for more than one release series are enabled, Yum uses the latest series.

Verify that the correct subrepositories have been enabled and disabled by running the following command and checking its output (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> yum repolist enabled | grep mysql
```

3 Disabling the Default MySQL Module

(EL8 systems only) EL8-based systems such as RHEL8 and Oracle Linux 8 include a MySQL module that is enabled by default. Unless this module is disabled, it masks packages provided by MySQL repositories. To disable the included module and make the MySQL repository packages visible, use the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum module disable mysql
```

4 Installing MySQL

Install MySQL by the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum install mysql-community-server
```

This installs the package for MySQL server (`mysql-community-server`) and also packages for the components required to run the server, including packages for the client (`mysql-community-client`), the common error messages and character sets for client and server (`mysql-community-common`), and the shared client libraries (`mysql-community-libs`).

5 Starting the MySQL Server

Start the MySQL server with the following command:

```
$> systemctl start mysqld
```

You can check the status of the MySQL server with the following command:

```
$> systemctl status mysqld
```

If the operating system is systemd enabled, standard `systemctl` (or alternatively, `service` with the arguments reversed) commands such as `stop`, `start`, `status`, and `restart` should be used to manage the MySQL server service. The `mysqld` service is enabled by default, and it starts at system reboot. See Section 2.9, “Managing MySQL Server with systemd” for additional information.

At the initial start up of the server, the following happens, given that the data directory of the server is empty:

- The server is initialized.
- SSL certificate and key files are generated in the data directory.
- `validate_password` is installed and enabled.
- A superuser account `'root'@'localhost'` is created. A password for the superuser is set and stored in the error log file. To reveal it, use the following command:

  ```
  $> sudo grep 'temporary password' /var/log/mysqld.log
  ```

  Change the root password as soon as possible by logging in with the generated, temporary password and set a custom password for the superuser account:

  ```
  $> mysql -uroot -p
  ```

  ```
  mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
  ```

  > **Note**
  >
  > `validate_password` is installed by default. The default password policy implemented by `validate_password` requires that passwords contain at least one uppercase letter, one lowercase letter, one digit, and one special character, and that the total password length is at least 8 characters.

For more information on the postinstallation procedures, see Postinstallation Setup and Testing.

> **Note**
>
> *Compatibility Information for EL7-based platforms:* The following RPM packages from the native software repositories of the platforms are incompatible with the package from the MySQL Yum repository that installs the MySQL server. Once you have installed MySQL using the MySQL Yum repository, you cannot install these packages (and vice versa).
>
> - akonadi-mysql

Installing Additional MySQL Products and Components with Yum

You can use Yum to install and manage individual components of MySQL. Some of these components are hosted in sub-repositories of the MySQL Yum repository: for example, the MySQL Connectors are to be found in the MySQL Connectors Community sub-repository, and the MySQL Workbench in MySQL Tools Community. You can use the following command to list the packages for all the MySQL components available for your platform from the MySQL Yum repository (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum --disablerepo=\* --enablerepo='mysql*-community*' list available
```

Install any packages of your choice with the following command, replacing *`package-name`* with name of the package (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum install package-name
```

For example, to install MySQL Workbench on Fedora:

```
$> sudo dnf install mysql-workbench-community
```

To install the shared client libraries (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum install mysql-community-libs
```

Platform Specific Notes

ARM Support

ARM 64-bit (aarch64) is supported on Oracle Linux 7 and requires the Oracle Linux 7 Software Collections Repository (ol7_software_collections). For example, to install the server:

```
$> yum-config-manager --enable ol7_software_collections
$> yum install mysql-community-server
```

> **Note**
>
> ARM 64-bit (aarch64) is supported on Oracle Linux 7 as of MySQL 8.0.12.

> **Known Limitation**
>
> The 8.0.12 release requires you to adjust the *libstdc++7* path by executing `ln -s /opt/oracle/oracle-armtoolset-1/root/usr/lib64 /usr/lib64/gcc7` after executing the `yum install` step.

Updating MySQL with Yum

Besides installation, you can also perform updates for MySQL products and components using the MySQL Yum repository. See Upgrading MySQL with the MySQL Yum Repository for details.

2.2 Installing MySQL on Linux Using the MySQL APT Repository

The MySQL APT repository provides `deb` packages for installing and managing the MySQL server, client, and other components on the current Debian and Ubuntu releases.

Instructions for using the MySQL APT Repository are available in A Quick Guide to Using the MySQL APT Repository.

2.3 Installing MySQL on Linux Using the MySQL SLES Repository

The MySQL SLES repository provides RPM packages for installing and managing the MySQL server, client, and other components on SUSE Enterprise Linux Server.

Instructions for using the MySQL SLES repository are available in A Quick Guide to Using the MySQL SLES Repository.

2.4 Installing MySQL on Linux Using RPM Packages from Oracle

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages provided by Oracle. There are two sources for obtaining them, for the Community Edition of MySQL:

- From the MySQL software repositories:
  - The MySQL Yum repository (see Section 2.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details).

  - The MySQL SLES repository (see Section 2.3, “Installing MySQL on Linux Using the MySQL SLES Repository” for details).

- From the Download MySQL Community Server page in the MySQL Developer Zone.

> **Note**
>
> RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the installation instructions in this manual do not necessarily apply to them. The vendor's instructions should be consulted instead.

MySQL RPM Packages

Table 2.2 RPM Packages for MySQL Community Edition

Package Name	Summary
`mysql-community-client`	MySQL client applications and tools
`mysql-community-client-plugins`	Shared plugins for MySQL client applications
`mysql-community-common`	Common files for server and client libraries
`mysql-community-devel`	Development header files and libraries for MySQL database client applications
`mysql-community-embedded-compat`	MySQL server as an embedded library with compatibility for applications using version 18 of the library
`mysql-community-icu-data-files`	MySQL packaging of ICU data files needed by MySQL regular expressions
`mysql-community-libs`	Shared libraries for MySQL database client applications
`mysql-community-libs-compat`	Shared compatibility libraries for previous MySQL installations; only present if previous MySQL versions are supported by the platform
`mysql-community-server`	Database server and related tools
`mysql-community-server-debug`	Debug server and plugin binaries
`mysql-community-test`	Test suite for the MySQL server
`mysql-community`	The source code RPM looks similar to mysql-community-8.0.36-1.el7.src.rpm, depending on selected OS
Additional *debuginfo* RPMs	There are several `debuginfo` packages: mysql-community-client-debuginfo, mysql-community-libs-debuginfo mysql-community-server-debug-debuginfo mysql-community-server-debuginfo, and mysql-community-test-debuginfo.

Table 2.3 RPM Packages for the MySQL Enterprise Edition

Package Name	Summary
`mysql-commercial-backup`	MySQL Enterprise Backup (added in 8.0.11)
`mysql-commercial-client`	MySQL client applications and tools
`mysql-commercial-client-plugins`	Shared plugins for MySQL client applications
`mysql-commercial-common`	Common files for server and client libraries

Package Name	Summary
mysql-commercial-devel	Development header files and libraries for MySQL database client applications
mysql-commercial-embedded-compat	MySQL server as an embedded library with compatibility for applications using version 18 of the library
mysql-commercial-icu-data-files	MySQL packaging of ICU data files needed by MySQL regular expressions
mysql-commercial-libs	Shared libraries for MySQL database client applications
mysql-commercial-libs-compat	Shared compatibility libraries for previous MySQL installations; only present if previous MySQL versions are supported by the platform. The version of the libraries matches the version of the libraries installed by default by the distribution you are using.
mysql-commercial-server	Database server and related tools
mysql-commercial-test	Test suite for the MySQL server
Additional *debuginfo* RPMs	There are several debuginfo packages: mysql-commercial-client-debuginfo, mysql-commercial-libs-debuginfo mysql-commercial-server-debug-debuginfo mysql-commercial-server-debuginfo, and mysql-commercial-test-debuginfo.

The full names for the RPMs have the following syntax:

```
packagename-version-distribution-arch.rpm
```

The *distribution* and *arch* values indicate the Linux distribution and the processor type for which the package was built. See the table below for lists of the distribution identifiers:

Table 2.4 MySQL Linux RPM Package Distribution Identifiers

Distribution Value	Intended Use
el*{version}* where *{version}* is the major Enterprise Linux version, such as el8	EL6 (8.0), EL7, EL8, and EL9-based platforms (for example, the corresponding versions of Oracle Linux, Red Hat Enterprise Linux, and CentOS)
fc*{version}* where *{version}* is the major Fedora version, such as fc37	Fedora 38 and 39
sles12	SUSE Linux Enterprise Server 12

To see all files in an RPM package (for example, mysql-community-server), use the following command:

```
$> rpm -qpl mysql-community-server-version-distribution-arch.rpm
```

The discussion in the rest of this section applies only to an installation process using the RPM packages directly downloaded from Oracle, instead of through a MySQL repository.

Dependency relationships exist among some of the packages. If you plan to install many of the packages, you may wish to download the RPM bundle tar file instead, which contains all the RPM packages listed above, so that you need not download them separately.

In most cases, you need to install the mysql-community-server, mysql-community-client, mysql-community-client-plugins, mysql-community-libs, mysql-community-icu-data-files, mysql-community-common, and mysql-community-libs-compat packages to get a functional, standard MySQL installation. To perform such a standard, basic installation, go to the

folder that contains all those packages (and, preferably, no other RPM packages with similar names), and issue the following command:

```
$> sudo yum install mysql-community-{server,client,client-plugins,icu-data-files,common,libs}-*
```

Replace `yum` with `zypper` for SLES, and with `dnf` for Fedora.

While it is much preferable to use a high-level package management tool like `yum` to install the packages, users who prefer direct `rpm` commands can replace the `yum install` command with the `rpm -Uvh` command; however, using `rpm -Uvh` instead makes the installation process more prone to failure, due to potential dependency issues the installation process might run into.

To install only the client programs, you can skip `mysql-community-server` in your list of packages to install; issue the following command:

```
$> sudo yum install mysql-community-{client,client-plugins,common,libs}-*
```

Replace `yum` with `zypper` for SLES, and with `dnf` for Fedora.

A standard installation of MySQL using the RPM packages result in files and resources created under the system directories, shown in the following table.

Table 2.5 MySQL Installation Layout for Linux RPM Packages from the MySQL Developer Zone

Files or Resources	Location
Client programs and scripts	`/usr/bin`
`mysqld` server	`/usr/sbin`
Configuration file	`/etc/my.cnf`
Data directory	`/var/lib/mysql`
Error log file	For RHEL, Oracle Linux, CentOS or Fedora platforms: `/var/log/mysqld.log` For SLES: `/var/log/mysql/mysqld.log`
Value of `secure_file_priv`	`/var/lib/mysql-files`
System V init script	For RHEL, Oracle Linux, CentOS or Fedora platforms: `/etc/init.d/mysqld` For SLES: `/etc/init.d/mysql`
Systemd service	For RHEL, Oracle Linux, CentOS or Fedora platforms: `mysqld` For SLES: `mysql`
Pid file	`/var/run/mysql/mysqld.pid`
Socket	`/var/lib/mysql/mysql.sock`
Keyring directory	`/var/lib/mysql-keyring`
Unix manual pages	`/usr/share/man`
Include (header) files	`/usr/include/mysql`
Libraries	`/usr/lib/mysql`
Miscellaneous support files (for example, error messages, and character set files)	`/usr/share/mysql`

The installation also creates a user named `mysql` and a group named `mysql` on the system.

Notes

- The `mysql` user is created using the `-r` and `-s /bin/false` options of the `useradd` command, so that it does not have login permissions to your server

host (see Creating the mysql User and Group for details). To switch to the `mysql` user on your OS, use the `--shell=/bin/bash` option for the `su` command:

```
su - mysql --shell=/bin/bash
```

- Installation of previous versions of MySQL using older packages might have created a configuration file named `/usr/my.cnf`. It is highly recommended that you examine the contents of the file and migrate the desired settings inside to the file `/etc/my.cnf` file, then remove `/usr/my.cnf`.

MySQL is NOT automatically started at the end of the installation process. For Red Hat Enterprise Linux, Oracle Linux, CentOS, and Fedora systems, use the following command to start MySQL:

```
$> systemctl start mysqld
```

For SLES systems, the command is the same, but the service name is different:

```
$> systemctl start mysql
```

If the operating system is systemd enabled, standard `systemctl` (or alternatively, `service` with the arguments reversed) commands such as `stop`, `start`, `status`, and `restart` should be used to manage the MySQL server service. The `mysqld` service is enabled by default, and it starts at system reboot. Notice that certain things might work differently on systemd platforms: for example, changing the location of the data directory might cause issues. See Section 2.9, “Managing MySQL Server with systemd” for additional information.

During an upgrade installation using RPM and DEB packages, if the MySQL server is running when the upgrade occurs then the MySQL server is stopped, the upgrade occurs, and the MySQL server is restarted. One exception: if the edition also changes during an upgrade (such as community to commercial, or vice-versa), then MySQL server is not restarted.

At the initial start up of the server, the following happens, given that the data directory of the server is empty:

- The server is initialized.
- An SSL certificate and key files are generated in the data directory.
- `validate_password` is installed and enabled.
- A superuser account `'root'@'localhost'` is created. A password for the superuser is set and stored in the error log file. To reveal it, use the following command for RHEL, Oracle Linux, CentOS, and Fedora systems:

  ```
  $> sudo grep 'temporary password' /var/log/mysqld.log
  ```

  Use the following command for SLES systems:

  ```
  $> sudo grep 'temporary password' /var/log/mysql/mysqld.log
  ```

  The next step is to log in with the generated, temporary password and set a custom password for the superuser account:

```
$> mysql -uroot -p
```

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

Note

`validate_password` is installed by default. The default password policy implemented by `validate_password` requires that passwords contain at least

> one uppercase letter, one lowercase letter, one digit, and one special character, and that the total password length is at least 8 characters.

If something goes wrong during installation, you might find debug information in the error log file `/var/log/mysqld.log`.

For some Linux distributions, it might be necessary to increase the limit on number of file descriptors available to `mysqld`. See File Not Found and Similar Errors

Installing Client Libraries from Multiple MySQL Versions. It is possible to install multiple client library versions, such as for the case that you want to maintain compatibility with older applications linked against previous libraries. To install an older client library, use the `--oldpackage` option with `rpm`. For example, to install `mysql-community-libs-5.5` on an EL6 system that has `libmysqlclient.21` from MySQL 8.0, use a command like this:

```
$> rpm --oldpackage -ivh mysql-community-libs-5.5.50-2.el6.x86_64.rpm
```

Debug Package. A special variant of MySQL Server compiled with the debug package has been included in the server RPM packages. It performs debugging and memory allocation checks and produces a trace file when the server is running. To use that debug version, start MySQL with `/usr/sbin/mysqld-debug`, instead of starting it as a service or with `/usr/sbin/mysqld`. See The DBUG Package for the debug options you can use.

> **Note**
>
> The default plugin directory for debug builds changed from `/usr/lib64/mysql/plugin` to `/usr/lib64/mysql/plugin/debug` in MySQL 8.0.4. Previously, it was necessary to change `plugin_dir` to `/usr/lib64/mysql/plugin/debug` for debug builds.

Rebuilding RPMs from source SRPMs. Source code SRPM packages for MySQL are available for download. They can be used as-is to rebuild the MySQL RPMs with the standard `rpmbuild` tool chain.

2.5 Installing MySQL on Linux Using Debian Packages from Oracle

Oracle provides Debian packages for installing MySQL on Debian or Debian-like Linux systems. The packages are available through two different channels:

- The MySQL APT Repository. This is the preferred method for installing MySQL on Debian-like systems, as it provides a simple and convenient way to install and update MySQL products. For details, see Section 2.2, “Installing MySQL on Linux Using the MySQL APT Repository”.
- The MySQL Developer Zone's Download Area. For details, see How to Get MySQL. The following are some information on the Debian packages available there and the instructions for installing them:
  - Various Debian packages are provided in the MySQL Developer Zone for installing different components of MySQL on the current Debian and Ubuntu platforms. The preferred method is to use the tarball bundle, which contains the packages needed for a basic setup of MySQL. The tarball bundles have names in the format of `mysql-server_`*`MVER`*`-`*`DVER`*`_`*`CPU`*`.deb-bundle.tar`. *`MVER`* is the MySQL version and *`DVER`* is the Linux distribution version. The *`CPU`* value indicates the processor type or family for which the package is built, as shown in the following table:

    Table 2.6 MySQL Debian and Ubuntu Installation Packages CPU Identifiers

`CPU` Value	Intended Processor Type or Family
`i386`	Pentium processor or better, 32 bit
`amd64`	64-bit x86 processor

- After downloading the tarball, unpack it with the following command:

```
$> tar -xvf mysql-server_MVER-DVER_CPU.deb-bundle.tar
```

- You may need to install the `libaio` library if it is not already present on your system:

```
$> sudo apt-get install libaio1
```

- Preconfigure the MySQL server package with the following command:

```
$> sudo dpkg-preconfigure mysql-community-server_*.deb
```

  You are asked to provide a password for the root user for your MySQL installation. You might also be asked other questions regarding the installation.

  > **Important**
  >
  > Make sure you remember the root password you set. Users who want to set a password later can leave the **password** field blank in the dialogue box and just press **OK**; in that case, root access to the server is authenticated using the MySQL Socket Peer-Credential Authentication Plugin for connections using a Unix socket file. You can set the root password later using `mysql_secure_installation`.

- For a basic installation of the MySQL server, install the database common files package, the client package, the client metapackage, the server package, and the server metapackage (in that order); you can do that with a single command:

```
$> sudo dpkg -i mysql-{common,community-client-plugins,community-client-core,community-client,client,co
```

  There are also packages with `server-core` and `client-core` in the package names. These contain binaries only and are installed automatically by the standard packages. Installing them by themselves does not result in a functioning MySQL setup.

  If you are being warned of unmet dependencies by `dpkg` (such as libmecab2), you can fix them using `apt-get`:

```
sudo apt-get -f install
```

  Here are where the files are installed on the system:

  - All configuration files (like `my.cnf`) are under `/etc/mysql`
  - All binaries, libraries, headers, etc., are under `/usr/bin` and `/usr/sbin`
  - The data directory is under `/var/lib/mysql`

  > **Note**
  >
  > Debian distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

2.6 Deploying MySQL on Linux with Docker Containers

This section explains how to deploy MySQL Server using Docker containers.

While the `docker` client is used in the following instructions for demonstration purposes, in general, the MySQL container images provided by Oracle work with any container tools that are compliant with the OCI 1.0 specification.

> **Warning**
>
> Before deploying MySQL with Docker containers, make sure you understand the security risks of running containers and mitigate them properly.

2.6.1 Basic Steps for MySQL Server Deployment with Docker

> **Warning**
>
> The MySQL Docker images maintained by the MySQL team are built specifically for Linux platforms. Other platforms are not supported, and users using these MySQL Docker images on them are doing so at their own risk. See the discussion here for some known limitations for running these containers on non-Linux operating systems.

- Downloading a MySQL Server Docker Image
- Starting a MySQL Server Instance
- Connecting to MySQL Server from within the Container
- Container Shell Access
- Stopping and Deleting a MySQL Container
- Upgrading a MySQL Server Container
- More Topics on Deploying MySQL Server with Docker

Downloading a MySQL Server Docker Image

> **Important**
>
> *For users of MySQL Enterprise Edition*: A subscription is required to use the Docker images for MySQL Enterprise Edition. Subscriptions work by a Bring Your Own License model; see How to Buy MySQL Products and Services for details.

Downloading the server image in a separate step is not strictly necessary; however, performing this step before you create your Docker container ensures your local image is up to date. To download the MySQL Community Edition image from the Oracle Container Registry (OCR), run this command:

```
docker pull container-registry.oracle.com/mysql/community-server:tag
```

The *`tag`* is the label for the image version you want to pull (for example, `5.7`, `8.0`, or `latest`). If ***`:tag`*** is omitted, the `latest` label is used, and the image for the latest GA version of MySQL Community Server is downloaded.

To download the MySQL Enterprise Edition image from the OCR, you need to first accept the license agreement on the OCR and log in to the container repository with your Docker client. Follow these steps:

- Visit the OCR at https://container-registry.oracle.com/ and choose **MySQL**.
- Under the list of MySQL repositories, choose `enterprise-server`.
- If you have not signed in to the OCR yet, click the **Sign in** button on the right of the page, and then enter your Oracle account credentials when prompted to.
- Follow the instructions on the right of the page to accept the license agreement.
- Log in to the OCR with your container client using, for example, the `docker login` command:

```
# docker login container-registry.oracle.com
Username: Oracle-Account-ID
```

```
Password: password
Login successful.
```

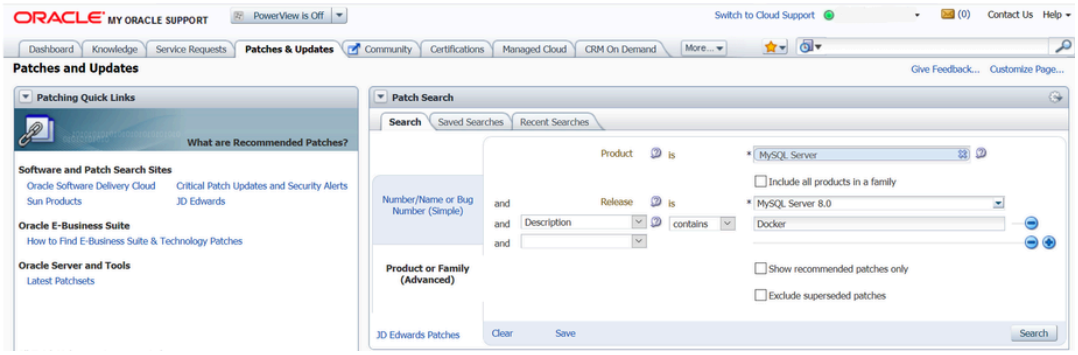
Download the Docker image for MySQL Enterprise Edition from the OCR with this command:

```
docker pull  container-registry.oracle.com/mysql/enterprise-server:tag
```

To download the MySQL Enterprise Edition image from My Oracle Support website, go onto the website, sign in to your Oracle account, and perform these steps once you are on the landing page:

- Select the **Patches and Updates** tab.
- Go to the **Patch Search** region and, on the **Search** tab, switch to the **Product or Family (Advanced)** subtab.
- Enter “MySQL Server” for the **Product** field, and the desired version number in the **Release** field.
- Use the dropdowns for additional filters to select **Description**—**contains**, and enter “Docker” in the text field.

  The following figure shows the search settings for the MySQL Enterprise Edition image for MySQL Server 8.0:



- Click the **Search** button and, from the result list, select the version you want, and click the **Download** button.
- In the **File Download** dialogue box that appears, click and download the `.zip` file for the Docker image.

Unzip the downloaded `.zip` archive to obtain the tarball inside (`mysql-enterprise-server-version.tar`), and then load the image by running this command:

```
docker load -i mysql-enterprise-server-version.tar
```

You can list downloaded Docker images with this command:

```
$> docker images
REPOSITORY                                          TAG       IMAGE ID       CREATED         SIZE
container-registry.oracle.com/mysql/community-server   latest    1d9c2219ff69   2 months ago    496MB
```

Starting a MySQL Server Instance

To start a new Docker container for a MySQL Server, use the following command:

```
docker run --name=container_name  --restart on-failure -d image_name:tag
```

image_name is the name of the image to be used to start the container; see Downloading a MySQL Server Docker Image for more information.

The `--name` option, for supplying a custom name for your server container, is optional; if no container name is supplied, a random one is generated.

The `--restart` option is for configuring the restart policy for your container; it should be set to the value `on-failure`, to enable support for server restart within a client session (which happens, for

example, when the RESTART statement is executed by a client or during the configuration of an InnoDB cluster instance). With the support for restart enabled, issuing a restart within a client session causes the server and the container to stop and then restart. *Support for server restart is available for MySQL 8.0.21 and later.*

For example, to start a new Docker container for the MySQL Community Server, use this command:

```
docker run --name=mysql1 --restart on-failure -d container-registry.oracle.com/mysql/community-server:l
```

To start a new Docker container for the MySQL Enterprise Server with a Docker image downloaded from the OCR, use this command:

```
docker run --name=mysql1 --restart on-failure -d container-registry.oracle.com/mysql/enterprise-server:
```

To start a new Docker container for the MySQL Enterprise Server with a Docker image downloaded from My Oracle Support, use this command:

```
docker run --name=mysql1 --restart on-failure -d mysql/enterprise-server:latest
```

If the Docker image of the specified name and tag has not been downloaded by an earlier `docker pull` or `docker run` command, the image is now downloaded. Initialization for the container begins, and the container appears in the list of running containers when you run the `docker ps` command. For example:

```
$> docker ps
CONTAINER ID   IMAGE                                                  COMMAND                  C
4cd4129b3211   container-registry.oracle.com/mysql/community-server:latest   "/entrypoint.sh mysq…"   8
```

The container initialization might take some time. When the server is ready for use, the `STATUS` of the container in the output of the `docker ps` command changes from `(health: starting)` to `(healthy)`.

The `-d` option used in the `docker run` command above makes the container run in the background. Use this command to monitor the output from the container:

```
docker logs mysql1
```

Once initialization is finished, the command's output is going to contain the random password generated for the root user; check the password with, for example, this command:

```
$> docker logs mysql1 2>&1 | grep GENERATED
GENERATED ROOT PASSWORD: Axegh3kAJyDLaRuBemecis&EShOs
```

Connecting to MySQL Server from within the Container

Once the server is ready, you can run the `mysql` client within the MySQL Server container you just started, and connect it to the MySQL Server. Use the `docker exec -it` command to start a `mysql` client inside the Docker container you have started, like the following:

```
docker exec -it mysql1 mysql -uroot -p
```

When asked, enter the generated root password (see the last step in Starting a MySQL Server Instance above on how to find the password). Because the `MYSQL_ONETIME_PASSWORD` option is true by default, after you have connected a `mysql` client to the server, you must reset the server root password by issuing this statement:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'password';
```

Substitute *`password`* with the password of your choice. Once the password is reset, the server is ready for use.

Container Shell Access

To have shell access to your MySQL Server container, use the `docker exec -it` command to start a bash shell inside the container:

```
$> docker exec -it mysql1 bash
bash-4.2#
```

You can then run Linux commands inside the container. For example, to view contents in the server's data directory inside the container, use this command:

```
bash-4.2# ls /var/lib/mysql
auto.cnf    ca.pem      client-key.pem  ib_logfile0  ibdata1  mysql       mysql.sock.lock    private_key.pe
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile1  ibtmp1   mysql.sock  performance_schema  public_k
```

Stopping and Deleting a MySQL Container

To stop the MySQL Server container we have created, use this command:

```
docker stop mysql1
```

`docker stop` sends a SIGTERM signal to the `mysqld` process, so that the server is shut down gracefully.

Also notice that when the main process of a container (`mysqld` in the case of a MySQL Server container) is stopped, the Docker container stops automatically.

To start the MySQL Server container again:

```
docker start mysql1
```

To stop and start again the MySQL Server container with a single command:

```
docker restart mysql1
```

To delete the MySQL container, stop it first, and then use the `docker rm` command:

```
docker stop mysql1
```

```
docker rm mysql1
```

If you want the Docker volume for the server's data directory to be deleted at the same time, add the `-v` option to the `docker rm` command.

Upgrading a MySQL Server Container

> **Important**
>
> - Before performing any upgrade to MySQL, follow carefully the instructions in Upgrading MySQL. Among other instructions discussed there, it is especially important to back up your database before the upgrade.
> - The instructions in this section require that the server's data and configuration have been persisted on the host. See Persisting Data and Configuration Changes for details.

Follow these steps to upgrade a Docker installation of MySQL 5.7 to 8.0:

- Stop the MySQL 5.7 server (container name is `mysql57` in this example):

  ```
  docker stop mysql57
  ```

- Download the MySQL 8.0 Server Docker image. See instructions in Downloading a MySQL Server Docker Image. Make sure you use the right tag for MySQL 8.0.

- Start a new MySQL 8.0 Docker container (named `mysql80` in this example) with the old server data and configuration (with proper modifications if needed—see Upgrading MySQL) that have been persisted on the host (by bind-mounting in this example). For the MySQL Community Server, run this command:

  ```
  docker run --name=mysql80 \
  ```

```
   --mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
   --mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
   -d container-registry.oracle.com/mysql/community-server:8.0
```

If needed, adjust `container-registry.oracle.com/mysql/community-server` to the correct image name—for example, replace it with `container-registry.oracle.com/mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from the OCR, or `mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from My Oracle Support.

- Wait for the server to finish startup. You can check the status of the server using the `docker ps` command (see Starting a MySQL Server Instance for how to do that).

Follow the same steps for upgrading within the 8.0 series (that is, from release 8.0.*x* to 8.0.*y*): stop the original container, and start a new one with a newer image on the old server data and configuration. If you used the `8.0` or the `latest` tag when starting your original container and there is now a new MySQL 8.0 release you want to upgrade to it, you must first pull the image for the new release with the command:

```
docker pull container-registry.oracle.com/mysql/community-server:8.0
```

You can then upgrade by starting a *new* container with the same tag on the old data and configuration (adjust the image name if you are using the MySQL Enterprise Edition; see Downloading a MySQL Server Docker Image):

```
docker run --name=mysql80new \
   --mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
   --mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
   -d container-registry.oracle.com/mysql/community-server:8.0
```

> **Note**
>
> *For MySQL 8.0.15 and earlier:* You need to complete the upgrade process by running the mysql_upgrade utility in the MySQL 8.0 Server container (the step is *not* required for MySQL 8.0.16 and later):
>
> - `docker exec -it mysql80 mysql_upgrade -uroot -p`
>
>   When prompted, enter the root password for your old server.
>
> - Finish the upgrade by restarting the new container:
>
>   `docker restart mysql80`

More Topics on Deploying MySQL Server with Docker

For more topics on deploying MySQL Server with Docker like server configuration, persisting data and configuration, server error log, and container environment variables, see Section 2.6.2, “More Topics on Deploying MySQL Server with Docker”.

2.6.2 More Topics on Deploying MySQL Server with Docker

> **Note**
>
> Most of the following sample commands have `container-registry.oracle.com/mysql/community-server` as the Docker image being used (like with the `docker pull` and `docker run` commands); change that if your image is from another repository—for example, replace it with `container-registry.oracle.com/mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from the Oracle Container Registry (OCR), or `mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from My Oracle Support.

- The Optimized MySQL Installation for Docker

The Optimized MySQL Installation for Docker

Docker images for MySQL are optimized for code size, which means they only include crucial components that are expected to be relevant for the majority of users who run MySQL instances in Docker containers. A MySQL Docker installation is different from a common, non-Docker installation in the following aspects:

- Only a limited number of binaries are included.
- All binaries are stripped; they contain no debug information.

> **Warning**
>
> Any software updates or installations users perform to the Docker container (including those for MySQL components) may conflict with the optimized MySQL installation created by the Docker image. Oracle does not provide support for MySQL products running in such an altered container, or a container created from an altered Docker image.

Configuring the MySQL Server

When you start the MySQL Docker container, you can pass configuration options to the server through the `docker run` command. For example:

```
docker run --name mysql1 -d container-registry.oracle.com/mysql/community-server:tag --character-set-server
```

The command starts the MySQL Server with `utf8mb4` as the default character set and `utf8mb4_col` as the default collation for databases.

Another way to configure the MySQL Server is to prepare a configuration file and mount it at the location of the server configuration file inside the container. See Persisting Data and Configuration Changes for details.

Persisting Data and Configuration Changes

Docker containers are in principle ephemeral, and any data or configuration are expected to be lost if the container is deleted or corrupted (see discussions here). Docker volumes provides a mechanism to persist data created inside a Docker container. At its initialization, the MySQL Server container creates a Docker volume for the server data directory. The JSON output from the `docker inspect` command on the container includes a `Mount` key, whose value provides information on the data directory volume:

```
$> docker inspect mysql1
...
 "Mounts": [
            {
                "Type": "volume",
                "Name": "4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652",
```

```
                "Source": "/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89
                "Destination": "/var/lib/mysql",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
...
```

The output shows that the source directory `/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652/_data`, in which data is persisted on the host, has been mounted at `/var/lib/mysql`, the server data directory inside the container.

Another way to preserve data is to bind-mount a host directory using the `--mount` option when creating the container. The same technique can be used to persist the configuration of the server. The following command creates a MySQL Server container and bind-mounts both the data directory and the server configuration file:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
-d container-registry.oracle.com/mysql/community-server:tag
```

The command mounts *`path-on-host-machine/my.cnf`* at `/etc/my.cnf` (the server configuration file inside the container), and *`path-on-host-machine/datadir`* at `/var/lib/mysql` (the data directory inside the container). The following conditions must be met for the bind-mounting to work:

- The configuration file *`path-on-host-machine/my.cnf`* must already exist, and it must contain the specification for starting the server by the user `mysql`:

  ```
  [mysqld]
  user=mysql
  ```

  You can also include other server configuration options in the file.

- The data directory *`path-on-host-machine/datadir`* must already exist. For server initialization to happen, the directory must be empty. You can also mount a directory prepopulated with data and start the server with it; however, you must make sure you start the Docker container with the same configuration as the server that created the data, and any host files or directories required are mounted when starting the container.

Running Additional Initialization Scripts

If there are any `.sh` or `.sql` scripts you want to run on the database immediately after it has been created, you can put them into a host directory and then mount the directory at `/docker-entrypoint-initdb.d/` inside the container. For example:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/scripts/,dst=/docker-entrypoint-initdb.d/ \
-d container-registry.oracle.com/mysql/community-server:tag
```

Connect to MySQL from an Application in Another Docker Container

By setting up a Docker network, you can allow multiple Docker containers to communicate with each other, so that a client application in another Docker container can access the MySQL Server in the server container. First, create a Docker network:

```
docker network create my-custom-net
```

Then, when you are creating and starting the server and the client containers, use the `--network` option to put them on network you created. For example:

```
docker run --name=mysql1 --network=my-custom-net -d container-registry.oracle.com/mysql/community-serve
```

```
docker run --name=myapp1 --network=my-custom-net -d myapp
```

The `myapp1` container can then connect to the `mysql1` container with the `mysql1` hostname and vice versa, as Docker automatically sets up a DNS for the given container names. In the following example, we run the `mysql` client from inside the `myapp1` container to connect to host `mysql1` in its own container:

```
docker exec -it myapp1 mysql --host=mysql1 --user=myuser --password
```

For other networking techniques for containers, see the Docker container networking section in the Docker Documentation.

Server Error Log

When the MySQL Server is first started with your server container, a server error log is NOT generated if either of the following conditions is true:

- A server configuration file from the host has been mounted, but the file does not contain the system variable `log_error` (see Persisting Data and Configuration Changes on bind-mounting a server configuration file).
- A server configuration file from the host has not been mounted, but the Docker environment variable `MYSQL_LOG_CONSOLE` is `true` (which is the variable's default state for MySQL 8.0 server containers). The MySQL Server's error log is then redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs mysqld-container` command.

To make MySQL Server generate an error log when either of the two conditions is true, use the `--log-error` option to configure the server to generate the error log at a specific location inside the container. To persist the error log, mount a host file at the location of the error log inside the container as explained in Persisting Data and Configuration Changes. However, you must make sure your MySQL Server inside its container has write access to the mounted host file.

Using MySQL Enterprise Backup with Docker

MySQL Enterprise Backup is a commercially-licensed backup utility for MySQL Server, available with MySQL Enterprise Edition. MySQL Enterprise Backup is included in the Docker installation of MySQL Enterprise Edition.

In the following example, we assume that you already have a MySQL Server running in a Docker container (see Section 2.6.1, “Basic Steps for MySQL Server Deployment with Docker” on how to start a MySQL Server instance with Docker). For MySQL Enterprise Backup to back up the MySQL Server, it must have access to the server's data directory. This can be achieved by, for example, bind-mounting a host directory on the data directory of the MySQL Server when you start the server:

```
docker run --name=mysqlserver \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

With this command, the MySQL Server is started with a Docker image of the MySQL Enterprise Edition, and the host directory */path-on-host-machine/datadir/* has been mounted onto the server's data directory (`/var/lib/mysql`) inside the server container. We also assume that, after the server has been started, the required privileges have also been set up for MySQL Enterprise Backup to access the server (see Grant MySQL Privileges to Backup Administrator, for details). Use the following steps to back up and restore a MySQL Server instance.

To back up a MySQL Server instance running in a Docker container using MySQL Enterprise Backup with Docker, follow the steps listed here:

1. On the same host where the MySQL Server container is running, start another container with an image of MySQL Enterprise Edition to perform a back up with the MySQL Enterprise Backup command `backup-to-image`. Provide access to the server's data directory using the bind mount

we created in the last step. Also, mount a host directory (*/path-on-host-machine/backups/* in this example) onto the storage folder for backups in the container (`/data/backups` in the example) to persist the backups we are creating. Here is a sample command for this step, in which MySQL Enterprise Backup is started with a Docker image downloaded from My Oracle Support:

```
$> docker run \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm mysql/enterprise-server:8.0 \
mysqlbackup -umysqlbackup -ppassword --backup-dir=/tmp/backup-tmp --with-timestamp \
--backup-image=/data/backups/db.mbi backup-to-image
[Entrypoint] MySQL Docker Image 8.0.11-1.1.5
MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08  07:06:
Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.
180921 17:27:25 MAIN    INFO: A thread created with Id '140594390935680'
180921 17:27:25 MAIN    INFO: Starting with following command line ...
...
------------------------------------------------------------
   Parameters Summary
------------------------------------------------------------
   Start LSN                  : 29615616
   End LSN                    : 29651854
------------------------------------------------------------
mysqlbackup completed OK!
```

It is important to check the end of the output by `mysqlbackup` to make sure the backup has been completed successfully.

2. The container exits once the backup job is finished and, with the `--rm` option used to start it, it is removed after it exits. An image backup has been created, and can be found in the host directory mounted in the last step for storing backups, as shown here:

```
$> ls /tmp/backups
db.mbi
```

To restore a MySQL Server instance in a Docker container using MySQL Enterprise Backup with Docker, follow the steps listed here:

1. Stop the MySQL Server container, which also stops the MySQL Server running inside:

```
docker stop mysqlserver
```

2. On the host, delete all contents in the bind mount for the MySQL Server data directory:

```
rm -rf /path-on-host-machine/datadir/*
```

3. Start a container with an image of MySQL Enterprise Edition to perform the restore with the MySQL Enterprise Backup command `copy-back-and-apply-log`. Bind-mount the server's data directory and the storage folder for the backups, like what we did when we backed up the server:

```
$> docker run \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm mysql/enterprise-server:8.0 \
mysqlbackup --backup-dir=/tmp/backup-tmp --with-timestamp \
--datadir=/var/lib/mysql --backup-image=/data/backups/db.mbi copy-back-and-apply-log
[Entrypoint] MySQL Docker Image 8.0.11-1.1.5
MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08  07:06:
Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.
180921 22:06:52 MAIN    INFO: A thread created with Id '139768047519872'
180921 22:06:52 MAIN    INFO: Starting with following command line ...
...
180921 22:06:52 PCR1    INFO: We were able to parse ibbackup_logfile up to
          lsn 29680612.
180921 22:06:52 PCR1    INFO: Last MySQL binlog file position 0 155, file name binlog.000003
180921 22:06:52 PCR1    INFO: The first data file is '/var/lib/mysql/ibdata1'
                              and the new created log files are at '/var/lib/mysql'
180921 22:06:52 MAIN    INFO: No Keyring file to process.
180921 22:06:52 MAIN    INFO: Apply-log operation completed successfully.
```

```
180921 22:06:52 MAIN    INFO: Full Backup has been restored successfully.
mysqlbackup completed OK! with 3 warnings
```

The container exits once the backup job is finished and, with the `--rm` option used when starting it, it is removed after it exits.

4. Restart the server container, which also restarts the restored server, using the following command:

```
docker restart mysqlserver
```

Or, start a new MySQL Server on the restored data directory, as shown here:

```
docker run --name=mysqlserver2 \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

Log on to the server to check that the server is running with the restored data.

Using `mysqldump` with Docker

Besides using MySQL Enterprise Backup to back up a MySQL Server running in a Docker container, you can perform a logical backup of your server by using the `mysqldump` utility, run inside a Docker container.

The following instructions assume that you already have a MySQL Server running in a Docker container and, when the container was first started, a host directory */path-on-host-machine/datadir/* has been mounted onto the server's data directory `/var/lib/mysql` (see bind-mounting a host directory on the data directory of the MySQL Server for details), which contains the Unix socket file by which `mysqldump` and `mysql` can connect to the server. We also assume that, after the server has been started, a user with the proper privileges (`admin` in this example) has been created, with which `mysqldump` can access the server. Use the following steps to back up and restore MySQL Server data:

Backing up MySQL Server data using `mysqldump` with Docker:

1. On the same host where the MySQL Server container is running, start another container with an image of MySQL Server to perform a backup with the `mysqldump` utility (see documentation of the utility for its functionality, options, and limitations). Provide access to the server's data directory by bind mounting */path-on-host-machine/datadir/*. Also, mount a host directory (*/path-on-host-machine/backups/* in this example) onto a storage folder for backups inside the container (`/data/backups` is used in this example) to persist the backups you are creating. Here is a sample command for backing up all databases on the server using this setup:

```
$> docker run --entrypoint "/bin/sh" \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm container-registry.oracle.com/mysql/community-server:8.0 \
-c "mysqldump -uadmin --password='password' --all-databases > /data/backups/all-databases.sql"
```

In the command, the `--entrypoint` option is used so that the system shell is invoked after the container is started, and the `-c` option is used to specify the `mysqldump` command to be run in the shell, whose output is redirected to the file `all-databases.sql` in the backup directory.

2. The container exits once the backup job is finished and, with the `--rm` option used to start it, it is removed after it exits. A logical backup been created, and can be found in the host directory mounted for storing the backup, as shown here:

```
$> ls /path-on-host-machine/backups/
all-databases.sql
```

Restoring MySQL Server data using `mysqldump` with Docker:

1. Make sure you have a MySQL Server running in a container, onto which you want your backed-up data to be restored.

2. Start a container with an image of MySQL Server to perform the restore with a `mysql` client. Bind-mount the server's data directory, as well as the storage folder that contains your backup:

```
$> docker run  \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm container-registry.oracle.com/mysql/community-server:8.0 \
mysql -uadmin --password='password' -e "source /data/backups/all-databases.sql"
```

   The container exits once the backup job is finished and, with the `--rm` option used when starting it, it is removed after it exits.

3. Log on to the server to check that the restored data is now on the server.

Known Issues

- When using the server system variable `audit_log_file` to configure the audit log file name, use the `loose` option modifier with it; otherwise, Docker cannot start the server.

Docker Environment Variables

When you create a MySQL Server container, you can configure the MySQL instance by using the `--env` option (short form `-e`) and specifying one or more environment variables. No server initialization is performed if the mounted data directory is not empty, in which case setting any of these variables has no effect (see Persisting Data and Configuration Changes), and no existing contents of the directory, including server settings, are modified during container startup.

Environment variables which can be used to configure a MySQL instance are listed here:

- The boolean variables including `MYSQL_RANDOM_ROOT_PASSWORD`, `MYSQL_ONETIME_PASSWORD`, `MYSQL_ALLOW_EMPTY_PASSWORD`, and `MYSQL_LOG_CONSOLE` are made true by setting them with any strings of nonzero lengths. Therefore, setting them to, for example, “0”, “false”, or “no” does not make them false, but actually makes them true. This is a known issue.

- `MYSQL_RANDOM_ROOT_PASSWORD`: When this variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` is set or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), a random password for the server's root user is generated when the Docker container is started. The password is printed to `stdout` of the container and can be found by looking at the container’s log (see Starting a MySQL Server Instance).

- `MYSQL_ONETIME_PASSWORD`: When the variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` is set or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), the root user's password is set as expired and must be changed before MySQL can be used normally.

- `MYSQL_DATABASE`: This variable allows you to specify the name of a database to be created on image startup. If a user name and a password are supplied with `MYSQL_USER` and `MYSQL_PASSWORD`, the user is created and granted superuser access to this database (corresponding to `GRANT ALL`). The specified database is created by a CREATE DATABASE IF NOT EXIST statement, so that the variable has no effect if the database already exists.

- `MYSQL_USER`, `MYSQL_PASSWORD`: These variables are used in conjunction to create a user and set that user's password, and the user is granted superuser permissions for the database specified by the `MYSQL_DATABASE` variable. Both `MYSQL_USER` and `MYSQL_PASSWORD` are required for a user to be created—if any of the two variables is not set, the other is ignored. If both variables are set but `MYSQL_DATABASE` is not, the user is created without any privileges.

  > **Note**
  >
  > There is no need to use this mechanism to create the root superuser, which is created by default with the password set by either one of the mechanisms discussed in the descriptions for

> `MYSQL_ROOT_PASSWORD` and `MYSQL_RANDOM_ROOT_PASSWORD`, unless `MYSQL_ALLOW_EMPTY_PASSWORD` is true.

- `MYSQL_ROOT_HOST`: By default, MySQL creates the `'root'@'localhost'` account. This account can only be connected to from inside the container as described in Connecting to MySQL Server from within the Container. To allow root connections from other hosts, set this environment variable. For example, the value `172.17.0.1`, which is the default Docker gateway IP, allows connections from the host machine that runs the container. The option accepts only one entry, but wildcards are allowed (for example, `MYSQL_ROOT_HOST=172.*.*.*` or `MYSQL_ROOT_HOST=%`).
- `MYSQL_LOG_CONSOLE`: When the variable is true (which is its default state for MySQL 8.0 server containers), the MySQL Server's error log is redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs mysqld-container` command.

  > **Note**
  >
  > The variable has no effect if a server configuration file from the host has been mounted (see Persisting Data and Configuration Changes on bind-mounting a configuration file).

- `MYSQL_ROOT_PASSWORD`: This variable specifies a password that is set for the MySQL root account.

  > **Warning**
  >
  > Setting the MySQL root user password on the command line is insecure. As an alternative to specifying the password explicitly, you can set the variable with a container file path for a password file, and then mount a file from your host that contains the password at the container file path. This is still not very secure, as the location of the password file is still exposed. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD` and `MYSQL_ONETIME_PASSWORD` both being true.

- `MYSQL_ALLOW_EMPTY_PASSWORD`. Set it to true to allow the container to be started with a blank password for the root user.

  > **Warning**
  >
  > Setting this variable to true is insecure, because it is going to leave your MySQL instance completely unprotected, allowing anyone to gain complete superuser access. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD` and `MYSQL_ONETIME_PASSWORD` both being true.

2.6.3 Deploying MySQL on Windows and Other Non-Linux Platforms with Docker

> **Warning**
>
> The MySQL Docker images provided by Oracle are built specifically for Linux platforms. Other platforms are not supported, and users running the MySQL Docker images from Oracle on them are doing so at their own risk. This section discusses some known issues for the images when used on non-Linux platforms.

Known Issues for using the MySQL Server Docker images from Oracle on Windows include:

- If you are bind-mounting on the container's MySQL data directory (see Persisting Data and Configuration Changes for details), you have to set the location of the server socket file with the `--socket` option to somewhere outside of the MySQL data directory; otherwise, the server fails to start. This is because the way Docker for Windows handles file mounting does not allow a host file from being bind-mounted on the socket file.

2.7 Installing MySQL on Linux from the Native Software Repositories

Many Linux distributions include a version of the MySQL server, client tools, and development components in their native software repositories and can be installed with the platforms' standard package management systems. This section provides basic instructions for installing MySQL using those package management systems.

> **Important**
>
> Native packages are often several versions behind the currently available release. You are also normally unable to install development milestone releases (DMRs), since these are not usually made available in the native repositories. Before proceeding, we recommend that you check out the other installation options described in Chapter 2, *Installing MySQL on Linux*.

Distribution specific instructions are shown below:

- **Red Hat Linux, Fedora, CentOS**

  > **Note**
  >
  > For a number of Linux distributions, you can install MySQL using the MySQL Yum repository instead of the platform's native software repository. See Section 2.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details.

  For Red Hat and similar distributions, the MySQL distribution is divided into a number of separate packages, `mysql` for the client tools, `mysql-server` for the server and associated tools, and `mysql-libs` for the libraries. The libraries are required if you want to provide connectivity from different languages and environments such as Perl, Python and others.

  To install, use the `yum` command to specify the packages that you want to install. For example:

```
#> yum install mysql mysql-server mysql-libs mysql-server
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package mysql.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-libs.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-server.x86_64 0:5.1.48-2.fc13 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.48-2.fc13.x86_64
--> Running transaction check
---> Package perl-DBD-MySQL.x86_64 0:4.017-1.fc13 set to be updated
--> Finished Dependency Resolution
Dependencies Resolved
================================================================================
 Package                Arch          Version                Repository      Size
================================================================================
Installing:
 mysql                  x86_64        5.1.48-2.fc13          updates        889 k
 mysql-libs             x86_64        5.1.48-2.fc13          updates        1.2 M
 mysql-server           x86_64        5.1.48-2.fc13          updates        8.1 M
Installing for dependencies:
 perl-DBD-MySQL         x86_64        4.017-1.fc13           updates        136 k
Transaction Summary
================================================================================
Install       4 Package(s)
Upgrade       0 Package(s)
Total download size: 10 M
Installed size: 30 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
```

```
Processing delta metadata
Package(s) data still to download: 10 M
(1/4): mysql-5.1.48-2.fc13.x86_64.rpm                    | 889 kB     00:04
(2/4): mysql-libs-5.1.48-2.fc13.x86_64.rpm               | 1.2 MB     00:06
(3/4): mysql-server-5.1.48-2.fc13.x86_64.rpm             | 8.1 MB     00:40
(4/4): perl-DBD-MySQL-4.017-1.fc13.x86_64.rpm            | 136 kB     00:00
--------------------------------------------------------------------------------
Total                                           201 kB/s |  10 MB     00:52
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : mysql-libs-5.1.48-2.fc13.x86_64                         1/4
  Installing     : mysql-5.1.48-2.fc13.x86_64                              2/4
  Installing     : perl-DBD-MySQL-4.017-1.fc13.x86_64                      3/4
  Installing     : mysql-server-5.1.48-2.fc13.x86_64                       4/4
Installed:
  mysql.x86_64 0:5.1.48-2.fc13           mysql-libs.x86_64 0:5.1.48-2.fc13
  mysql-server.x86_64 0:5.1.48-2.fc13
Dependency Installed:
  perl-DBD-MySQL.x86_64 0:4.017-1.fc13
Complete!
```

MySQL and the MySQL server should now be installed. A sample configuration file is installed into `/etc/my.cnf`. To start the MySQL server use `systemctl`:

```
$> systemctl start mysqld
```

The database tables are automatically created for you, if they do not already exist. You should, however, run `mysql_secure_installation` to set the root passwords on your server.

- **Debian, Ubuntu, Kubuntu**

  > **Note**
  >
  > For supported Debian and Ubuntu versions, MySQL can be installed using the MySQL APT Repository instead of the platform's native software repository. See Section 2.2, “Installing MySQL on Linux Using the MySQL APT Repository” for details.

  On Debian and related distributions, there are two packages for MySQL in their software repositories, `mysql-client` and `mysql-server`, for the client and server components respectively. You should specify an explicit version, for example `mysql-client-5.1`, to ensure that you install the version of MySQL that you want.

  To download and install, including any dependencies, use the `apt-get` command, specifying the packages that you want to install.

  > **Note**
  >
  > Before installing, make sure that you update your `apt-get` index files to ensure you are downloading the latest available version.

  > **Note**
  >
  > The `apt-get` command installs a number of packages, including the MySQL server, in order to provide the typical tools and application environment. This

can mean that you install a large number of packages in addition to the main MySQL package.

During installation, the initial database is created, and you are prompted for the MySQL root password (and confirmation). A configuration file is created in `/etc/mysql/my.cnf`. An init script is created in `/etc/init.d/mysql`.

The server should already be started. You can manually start and stop the server using:

```
#> service mysql [start|stop]
```

The service is automatically added to the 2, 3 and 4 run levels, with stop scripts in the single, shutdown and restart levels.

2.8 Installing MySQL on Linux with Juju

The Juju deployment framework supports easy installation and configuration of MySQL servers. For instructions, see https://jujucharms.com/mysql/.

2.9 Managing MySQL Server with systemd

If you install MySQL using an RPM or Debian package on the following Linux platforms, server startup and shutdown is managed by systemd:

- RPM package platforms:
  - Enterprise Linux variants version 7 and higher
  - SUSE Linux Enterprise Server 12 and higher
  - Fedora 29 and higher
- Debian family platforms:
  - Debian platforms
  - Ubuntu platforms

If you install MySQL from a generic binary distribution on a platform that uses systemd, you can manually configure systemd support for MySQL following the instructions provided in the post-installation setup section of the MySQL 8.0 Secure Deployment Guide.

If you install MySQL from a source distribution on a platform that uses systemd, obtain systemd support for MySQL by configuring the distribution using the `-DWITH_SYSTEMD=1` `CMake` option. See MySQL Source-Configuration Options.

The following discussion covers these topics:

- Overview of systemd
- Configuring systemd for MySQL
- Configuring Multiple MySQL Instances Using systemd
- Migrating from mysqld_safe to systemd

Note

On platforms for which systemd support for MySQL is installed, scripts such as `mysqld_safe` and the System V initialization script are unnecessary and are not installed. For example, `mysqld_safe` can handle server restarts, but

> systemd provides the same capability, and does so in a manner consistent with management of other services rather than by using an application-specific program.
>
> One implication of the non-use of `mysqld_safe` on platforms that use systemd for server management is that use of `[mysqld_safe]` or `[safe_mysqld]` sections in option files is not supported and might lead to unexpected behavior.
>
> Because systemd has the capability of managing multiple MySQL instances on platforms for which systemd support for MySQL is installed, `mysqld_multi` and `mysqld_multi.server` are unnecessary and are not installed.

Overview of systemd

systemd provides automatic MySQL server startup and shutdown. It also enables manual server management using the `systemctl` command. For example:

```
$> systemctl {start|stop|restart|status} mysqld
```

Alternatively, use the `service` command (with the arguments reversed), which is compatible with System V systems:

```
$> service mysqld {start|stop|restart|status}
```

> **Note**
>
> For the `systemctl` command (and the alternative `service` command), if the MySQL service name is not `mysqld` then use the appropriate name. For example, use `mysql` rather than `mysqld` on Debian-based and SLES systems.

Support for systemd includes these files:

- `mysqld.service` (RPM platforms), `mysql.service` (Debian platforms): systemd service unit configuration file, with details about the MySQL service.
- `mysqld@.service` (RPM platforms), `mysql@.service` (Debian platforms): Like `mysqld.service` or `mysql.service`, but used for managing multiple MySQL instances.
- `mysqld.tmpfiles.d`: File containing information to support the `tmpfiles` feature. This file is installed under the name `mysql.conf`.
- `mysqld_pre_systemd` (RPM platforms), `mysql-system-start` (Debian platforms): Support script for the unit file. This script assists in creating the error log file only if the log location matches a pattern (`/var/log/mysql*.log` for RPM platforms, `/var/log/mysql/*.log` for Debian platforms). In other cases, the error log directory must be writable or the error log must be present and writable for the user running the `mysqld` process.

Configuring systemd for MySQL

To add or change systemd options for MySQL, these methods are available:

- Use a localized systemd configuration file.
- Arrange for systemd to set environment variables for the MySQL server process.
- Set the `MYSQLD_OPTS` systemd variable.

To use a localized systemd configuration file, create the `/etc/systemd/system/mysqld.service.d` directory if it does not exist. In that directory, create a file that contains a `[Service]` section listing the desired settings. For example:

```
[Service]
```

```
LimitNOFILE=max_open_files
Nice=nice_level
LimitCore=core_file_limit
Environment="LD_PRELOAD=/path/to/malloc/library"
Environment="TZ=time_zone_setting"
```

The discussion here uses `override.conf` as the name of this file. Newer versions of systemd support the following command, which opens an editor and permits you to edit the file:

```
systemctl edit mysqld  # RPM platforms
systemctl edit mysql   # Debian platforms
```

Whenever you create or change `override.conf`, reload the systemd configuration, then tell systemd to restart the MySQL service:

```
systemctl daemon-reload
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

With systemd, the `override.conf` configuration method must be used for certain parameters, rather than settings in a `[mysqld]`, `[mysqld_safe]`, or `[safe_mysqld]` group in a MySQL option file:

- For some parameters, `override.conf` must be used because systemd itself must know their values and it cannot read MySQL option files to get them.
- Parameters that specify values otherwise settable only using options known to `mysqld_safe` must be specified using systemd because there is no corresponding `mysqld` parameter.

For additional information about using systemd rather than `mysqld_safe`, see Migrating from mysqld_safe to systemd.

You can set the following parameters in `override.conf`:

- To set the number of file descriptors available to the MySQL server, use `LimitNOFILE` in `override.conf` rather than the `open_files_limit` system variable for `mysqld` or `--open-files-limit` option for `mysqld_safe`.
- To set the maximum core file size, use `LimitCore` in `override.conf` rather than the `--core-file-size` option for `mysqld_safe`.
- To set the scheduling priority for the MySQL server, use `Nice` in `override.conf` rather than the `--nice` option for `mysqld_safe`.

Some MySQL parameters are configured using environment variables:

- `LD_PRELOAD`: Set this variable if the MySQL server should use a specific memory-allocation library.
- `NOTIFY_SOCKET`: This environment variable specifies the socket that `mysqld` uses to communicate notification of startup completion and service status change with systemd. It is set by systemd when the `mysqld` service is started. The `mysqld` service reads the variable setting and writes to the defined location.

  In MySQL 8.0, `mysqld` uses the `Type=notify` process startup type. (`Type=forking` was used in MySQL 5.7.) With `Type=notify`, systemd automatically configures a socket file and exports the path to the `NOTIFY_SOCKET` environment variable.
- `TZ`: Set this variable to specify the default time zone for the server.

There are multiple ways to specify environment variable values for use by the MySQL server process managed by systemd:

- Use `Environment` lines in the `override.conf` file. For the syntax, see the example in the preceding discussion that describes how to use this file.

- Specify the values in the `/etc/sysconfig/mysql` file (create the file if it does not exist). Assign values using the following syntax:

```
LD_PRELOAD=/path/to/malloc/library
TZ=time_zone_setting
```

  After modifying `/etc/sysconfig/mysql`, restart the server to make the changes effective:

```
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

To specify options for `mysqld` without modifying systemd configuration files directly, set or unset the `MYSQLD_OPTS` systemd variable. For example:

```
systemctl set-environment MYSQLD_OPTS="--general_log=1"
systemctl unset-environment MYSQLD_OPTS
```

`MYSQLD_OPTS` can also be set in the `/etc/sysconfig/mysql` file.

After modifying the systemd environment, restart the server to make the changes effective:

```
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

For platforms that use systemd, the data directory is initialized if empty at server startup. This might be a problem if the data directory is a remote mount that has temporarily disappeared: The mount point would appear to be an empty data directory, which then would be initialized as a new data directory. To suppress this automatic initialization behavior, specify the following line in the `/etc/sysconfig/mysql` file (create the file if it does not exist):

```
NO_INIT=true
```

Configuring Multiple MySQL Instances Using systemd

This section describes how to configure systemd for multiple instances of MySQL.

> **Note**
>
> Because systemd has the capability of managing multiple MySQL instances on platforms for which systemd support is installed, `mysqld_multi` and `mysqld_multi.server` are unnecessary and are not installed.

To use multiple-instance capability, modify the `my.cnf` option file to include configuration of key options for each instance. These file locations are typical:

- `/etc/my.cnf` or `/etc/mysql/my.cnf` (RPM platforms)
- `/etc/mysql/mysql.conf.d/mysqld.cnf` (Debian platforms)

For example, to manage two instances named `replica01` and `replica02`, add something like this to the option file:

RPM platforms:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysqld-replica01.log
[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
```

```
log-error=/var/log/mysqld-replica02.log
```

Debian platforms:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysql/replica01.log
[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
log-error=/var/log/mysql/replica02.log
```

The replica names shown here use `@` as the delimiter because that is the only delimiter supported by systemd.

Instances then are managed by normal systemd commands, such as:

```
systemctl start mysqld@replica01
systemctl start mysqld@replica02
```

To enable instances to run at boot time, do this:

```
systemctl enable mysqld@replica01
systemctl enable mysqld@replica02
```

Use of wildcards is also supported. For example, this command displays the status of all replica instances:

```
systemctl status 'mysqld@replica*'
```

For management of multiple MySQL instances on the same machine, systemd automatically uses a different unit file:

- `mysqld@.service` rather than `mysqld.service` (RPM platforms)
- `mysql@.service` rather than `mysql.service` (Debian platforms)

In the unit file, `%I` and `%i` reference the parameter passed in after the `@` marker and are used to manage the specific instance. For a command such as this:

```
systemctl start mysqld@replica01
```

systemd starts the server using a command such as this:

```
mysqld --defaults-group-suffix=@%I ...
```

The result is that the `[server]`, `[mysqld]`, and `[mysqld@replica01]` option groups are read and used for that instance of the service.

> **Note**
>
> On Debian platforms, AppArmor prevents the server from reading or writing `/var/lib/mysql-replica*`, or anything other than the default locations. To address this, you must customize or disable the profile in `/etc/apparmor.d/usr.sbin.mysqld`.

> **Note**
>
> On Debian platforms, the packaging scripts for MySQL uninstallation cannot currently handle `mysqld@` instances. Before removing or upgrading the package, you must stop any extra instances manually first.

Migrating from mysqld_safe to systemd

Because `mysqld_safe` is not installed on platforms that use systemd to manage MySQL, options previously specified for that program (for example, in an `[mysqld_safe]` or `[safe_mysqld]` option group) must be specified another way:

- Some `mysqld_safe` options are also understood by `mysqld` and can be moved from the `[mysqld_safe]` or `[safe_mysqld]` option group to the `[mysqld]` group. This does *not* include `--pid-file`, `--open-files-limit`, or `--nice`. To specify those options, use the `override.conf` systemd file, described previously.

  > **Note**
  >
  > On systemd platforms, use of `[mysqld_safe]` and `[safe_mysqld]` option groups is not supported and may lead to unexpected behavior.

- For some `mysqld_safe` options, there are alternative `mysqld` procedures. For example, the `mysqld_safe` option for enabling `syslog` logging is `--syslog`, which is deprecated. To write error log output to the system log, use the instructions at Error Logging to the System Log.

- `mysqld_safe` options not understood by `mysqld` can be specified in `override.conf` or environment variables. For example, with `mysqld_safe`, if the server should use a specific memory allocation library, this is specified using the `--malloc-lib` option. For installations that manage the server with systemd, arrange to set the `LD_PRELOAD` environment variable instead, as described previously.

Chapter 3 Installing MySQL on Solaris

Table of Contents

> **Note**
>
> MySQL 8.0 supports Solaris 11.4 and higher

MySQL on Solaris is available in a number of different formats.

- For information on installing using the native Solaris PKG format, see Section 3.1, “Installing MySQL on Solaris Using a Solaris PKG”.
- To use a standard `tar` binary installation, use the notes provided in Chapter 1, *Installing MySQL on Unix/Linux Using Generic Binaries*. Check the notes and hints at the end of this section for Solaris specific notes that you may need before or after installation.

  > **Note**
  >
  > MySQL 5.7 has a dependency on the Oracle Developer Studio Runtime Libraries; but this does not apply to MySQL 8.0.

To obtain a binary MySQL distribution for Solaris in tarball or PKG format, https://dev.mysql.com/downloads/mysql/8.0.html.

Additional notes to be aware of when installing and using MySQL on Solaris:

- If you want to use MySQL with the `mysql` user and group, use the `groupadd` and `useradd` commands:

  ```
  groupadd mysql
  useradd -g mysql -s /bin/false mysql
  ```

- If you install MySQL using a binary tarball distribution on Solaris, because the Solaris `tar` cannot handle long file names, use GNU `tar` (`gtar`) to unpack the distribution. If you do not have GNU `tar` on your system, install it with the following command:

  ```
  pkg install archiver/gnu-tar
  ```

- You should mount any file systems on which you intend to store `InnoDB` files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so causes a significant drop in performance when using the `InnoDB` storage engine on this platform.
- If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.
- If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

  ```
  Error in accept: Protocol error
  ```

  You might try starting the server with the `--back_log=50` option as a workaround for this.

- To configure the generation of core files on Solaris you should use the `coreadm` command. Because of the security implications of generating a core on a `setuid()` application, by default, Solaris does not support core files on `setuid()` programs. However, you can modify this behavior using `coreadm`. If you enable `setuid()` core files for the current user, they are generated using mode 600 and are owned by the superuser.

3.1 Installing MySQL on Solaris Using a Solaris PKG

You can install MySQL on Solaris using a binary package of the native Solaris PKG format instead of the binary tarball distribution.

> **Note**
>
> MySQL 5.7 has a dependency on the Oracle Developer Studio Runtime Libraries; but this does not apply to MySQL 8.0.

To use this package, download the corresponding `mysql-VERSION-solaris11-PLATFORM.pkg.gz` file, then uncompress it. For example:

```
$> gunzip mysql-8.0.36-solaris11-x86_64.pkg.gz
```

To install a new package, use `pkgadd` and follow the onscreen prompts. You must have root privileges to perform this operation:

```
$> pkgadd -d mysql-8.0.36-solaris11-x86_64.pkg
The following packages are available:
  1  mysql     MySQL Community Server (GPL)
               (i86pc) 8.0.36
Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

The PKG installer installs all of the files and tools needed, and then initializes your database if one does not exist. To complete the installation, you should set the root password for MySQL as provided in the instructions at the end of the installation. Alternatively, you can run the `mysql_secure_installation` script that comes with the installation.

By default, the PKG package installs MySQL under the root path `/opt/mysql`. You can change only the installation root path when using `pkgadd`, which can be used to install MySQL in a different Solaris zone. If you need to install in a specific directory, use a binary `tar` file distribution.

The `pkg` installer copies a suitable startup script for MySQL into `/etc/init.d/mysql`. To enable MySQL to startup and shutdown automatically, you should create a link between this file and the init script directories. For example, to ensure safe startup and shutdown of MySQL you could use the following commands to add the right links:

```
$> ln /etc/init.d/mysql /etc/rc3.d/S91mysql
$> ln /etc/init.d/mysql /etc/rc0.d/K02mysql
```

To remove MySQL, the installed package name is `mysql`. You can use this in combination with the `pkgrm` command to remove the installation.

To upgrade when using the Solaris package file format, you must remove the existing installation before installing the updated package. Removal of the package does not delete the existing database information, only the server, binaries and support files. The typical upgrade sequence is therefore:

```
$> mysqladmin shutdown
$> pkgrm mysql
$> pkgadd -d mysql-8.0.36-solaris11-x86_64.pkg
$> mysqld_safe &
$> mysql_upgrade   # prior to MySQL 8.0.16 only
```

You should check the notes in Upgrading MySQL before performing any upgrade.

Chapter 4 Installing MySQL on FreeBSD

This section provides information about installing MySQL on variants of FreeBSD Unix.

You can install MySQL on FreeBSD by using the binary distribution provided by Oracle. For more information, see Chapter 1, *Installing MySQL on Unix/Linux Using Generic Binaries*.

The easiest (and preferred) way to install MySQL is to use the `mysql-server` and `mysql-client` ports available at http://www.freebsd.org/. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

The MySQL build process requires GNU make (`gmake`) to work. If GNU `make` is not available, you must install it first before compiling MySQL.

> **Note**
>
> Prerequisite libraries as per `ldd mysqld`: libthr, libcrypt, libkrb5, libm, librt, libexecinfo, libunwind, and libssl.

To install using the ports system:

```
# cd /usr/ports/databases/mysql80-server
# make
...
# cd /usr/ports/databases/mysql80-client
# make
...
```

The standard port installation places the server into `/usr/local/libexec/mysqld`, with the startup script for the MySQL server placed in `/usr/local/etc/rc.d/mysql-server`.

Some additional notes on the BSD implementation:

- To remove MySQL after installation using the ports system:

```
# cd /usr/ports/databases/mysql80-server
# make deinstall
...
# cd /usr/ports/databases/mysql80-client
# make deinstall
...
```

- If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Environment Variables.

Chapter 5 Initializing the Data Directory

After MySQL is installed, the data directory must be initialized, including the tables in the `mysql` system schema:

- For some MySQL installation methods, data directory initialization is automatic, as described in Postinstallation Setup and Testing.
- For other installation methods, you must initialize the data directory manually. These include installation from generic binary and source distributions on Unix and Unix-like systems, and installation from a ZIP Archive package on Windows.

This section describes how to initialize the data directory manually for MySQL installation methods for which data directory initialization is not automatic. For some suggested commands that enable testing whether the server is accessible and working properly, see Testing the Server.

> **Note**
>
> In MySQL 8.0, the default authentication plugin has changed from `mysql_native_password` to `caching_sha2_password`, and the `'root'@'localhost'` administrative account uses `caching_sha2_password` by default. If you prefer that the `root` account use the previous default authentication plugin (`mysql_native_password`), see caching_sha2_password and the root Administrative Account.

- Data Directory Initialization Overview
- Data Directory Initialization Procedure
- Server Actions During Data Directory Initialization
- Post-Initialization root Password Assignment

Data Directory Initialization Overview

In the examples shown here, the server is intended to run under the user ID of the `mysql` login account. Either create the account if it does not exist (see Create a mysql User and Group), or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location to the top-level directory of your MySQL installation, which is typically `/usr/local/mysql` (adjust the path name for your system as necessary):

   ```
   cd /usr/local/mysql
   ```

   Within this directory you can find several files and subdirectories, including the `bin` subdirectory that contains the server, as well as client and utility programs.

2. The `secure_file_priv` system variable limits import and export operations to a specific directory. Create a directory whose location can be specified as the value of that variable:

   ```
   mkdir mysql-files
   ```

   Grant directory user and group ownership to the `mysql` user and `mysql` group, and set the directory permissions appropriately:

   ```
   chown mysql:mysql mysql-files
   chmod 750 mysql-files
   ```

3. Use the server to initialize the data directory, including the `mysql` schema containing the initial MySQL grant tables that determine how users are permitted to connect to the server. For example:

   ```
   bin/mysqld --initialize --user=mysql
   ```

For important information about the command, especially regarding command options you might use, see Data Directory Initialization Procedure. For details about how the server performs initialization, see Server Actions During Data Directory Initialization.

Typically, data directory initialization need be done only after you first install MySQL. (For upgrades to an existing installation, perform the upgrade procedure instead; see Upgrading MySQL.) However, the command that initializes the data directory does not overwrite any existing `mysql` schema tables, so it is safe to run in any circumstances.

4. If you want to deploy the server with automatic support for secure connections, use the `mysql_ssl_rsa_setup` utility to create default SSL and RSA files:

   ```
   bin/mysql_ssl_rsa_setup
   ```

   For more information, see mysql_ssl_rsa_setup — Create SSL/RSA Files.

   > **Note**
   >
   > The `mysql_ssl_rsa_setup` utility is deprecated as of MySQL 8.0.34.

5. In the absence of any option files, the server starts with its default settings. (See Server Configuration Defaults.) To explicitly specify options that the MySQL server should use at startup, put them in an option file such as `/etc/my.cnf` or `/etc/mysql/my.cnf`. (See Using Option Files.) For example, you can use an option file to set the `secure_file_priv` system variable.

6. To arrange for MySQL to start without manual intervention at system boot time, see Starting and Stopping MySQL Automatically.

7. Data directory initialization creates time zone tables in the `mysql` schema but does not populate them. To do so, use the instructions in MySQL Server Time Zone Support.

Data Directory Initialization Procedure

Change location to the top-level directory of your MySQL installation, which is typically `/usr/local/mysql` (adjust the path name for your system as necessary):

```
cd /usr/local/mysql
```

To initialize the data directory, invoke `mysqld` with the `--initialize` or `--initialize-insecure` option, depending on whether you want the server to generate a random initial password for the `'root'@'localhost'` account, or to create that account with no password:

- Use `--initialize` for “secure by default” installation (that is, including generation of a random initial `root` password). In this case, the password is marked as expired and you must choose a new one.
- With `--initialize-insecure`, no `root` password is generated. This is insecure; it is assumed that you intend to assign a password to the account in a timely fashion before putting the server into production use.

For instructions on assigning a new `'root'@'localhost'` password, see Post-Initialization root Password Assignment.

> **Note**
>
> The server writes any messages (including any initial password) to its standard error output. This may be redirected to the error log, so look there if you do not see the messages on your screen. For information about the error log, including where it is located, see The Error Log.
>
> On Windows, use the `--console` option to direct messages to the console.

On Unix and Unix-like systems, it is important for the database directories and files to be owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this, start `mysqld` from the system `root` account and include the `--user` option as shown here:

```
bin/mysqld --initialize --user=mysql
bin/mysqld --initialize-insecure --user=mysql
```

Alternatively, execute `mysqld` while logged in as `mysql`, in which case you can omit the `--user` option from the command.

On Windows, use one of these commands:

```
bin\mysqld --initialize --console
bin\mysqld --initialize-insecure --console
```

> **Note**
>
> Data directory initialization might fail if required system libraries are missing. For example, you might see an error like this:
>
> ```
> bin/mysqld: error while loading shared libraries:
> libnuma.so.1: cannot open shared object file:
> No such file or directory
> ```
>
> If this happens, you must install the missing libraries manually or with your system's package manager. Then retry the data directory initialization command.

It might be necessary to specify other options such as `--basedir` or `--datadir` if `mysqld` cannot identify the correct locations for the installation directory or data directory. For example (enter the command on a single line):

```
bin/mysqld --initialize --user=mysql
  --basedir=/opt/mysql/mysql
  --datadir=/opt/mysql/mysql/data
```

Alternatively, put the relevant option settings in an option file and pass the name of that file to `mysqld`. For Unix and Unix-like systems, suppose that the option file name is `/opt/mysql/mysql/etc/my.cnf`. Put these lines in the file:

```
[mysqld]
basedir=/opt/mysql/mysql
datadir=/opt/mysql/mysql/data
```

Then invoke `mysqld` as follows (enter the command on a single line, with the `--defaults-file` option first):

```
bin/mysqld --defaults-file=/opt/mysql/mysql/etc/my.cnf
  --initialize --user=mysql
```

On Windows, suppose that `C:\my.ini` contains these lines:

```
[mysqld]
basedir=C:\\Program Files\\MySQL\\MySQL Server 8.0
datadir=D:\\MySQLdata
```

Then invoke `mysqld` as follows (again, you should enter the command on a single line, with the `--defaults-file` option first):

```
bin\mysqld --defaults-file=C:\my.ini
   --initialize --console
```

> **Important**
>
> When initializing the data directory, you should not specify any options other than those used for setting directory locations such as `--basedir` or `--`

> `datadir`, and the `--user` option if needed. Options to be employed by the MySQL server during normal use can be set when restarting it following initialization. See the description of the `--initialize` option for further information.

Server Actions During Data Directory Initialization

> **Note**
>
> The data directory initialization sequence performed by the server does not substitute for the actions performed by `mysql_secure_installation` and `mysql_ssl_rsa_setup`. See mysql_secure_installation — Improve MySQL Installation Security, and mysql_ssl_rsa_setup — Create SSL/RSA Files.

When invoked with the `--initialize` or `--initialize-insecure` option, `mysqld` performs the following actions during the data directory initialization sequence:

1. The server checks for the existence of the data directory as follows:

   - If no data directory exists, the server creates it.
   - If the data directory exists but is not empty (that is, it contains files or subdirectories), the server exits after producing an error message:

     ```
     [ERROR] --initialize specified but the data directory exists. Aborting.
     ```

     In this case, remove or rename the data directory and try again.

     An existing data directory is permitted to be nonempty if every entry has a name that begins with a period (`.`).

2. Within the data directory, the server creates the `mysql` system schema and its tables, including the data dictionary tables, grant tables, time zone tables, and server-side help tables. See The mysql System Schema.

3. The server initializes the system tablespace and related data structures needed to manage `InnoDB` tables.

   > **Note**
   >
   > After `mysqld` sets up the `InnoDB` system tablespace, certain changes to tablespace characteristics require setting up a whole new instance. Qualifying changes include the file name of the first file in the system tablespace and the number of undo logs. If you do not want to use the default values, make sure that the settings for the `innodb_data_file_path` and `innodb_log_file_size` configuration parameters are in place in the MySQL configuration file *before* running `mysqld`. Also make sure to specify as necessary other parameters that affect the creation and location of `InnoDB` files, such as `innodb_data_home_dir` and `innodb_log_group_home_dir`.
   >
   > If those options are in your configuration file but that file is not in a location that MySQL reads by default, specify the file location using the `--defaults-extra-file` option when you run `mysqld`.

4. The server creates a `'root'@'localhost'` superuser account and other reserved accounts (see Reserved Accounts). Some reserved accounts are locked and cannot be used by clients, but `'root'@'localhost'` is intended for administrative use and you should assign it a password.

   Server actions with respect to a password for the `'root'@'localhost'` account depend on how you invoke it:

- With `--initialize` but not `--initialize-insecure`, the server generates a random password, marks it as expired, and writes a message displaying the password:

```
[Warning] A temporary password is generated for root@localhost:
iTag*AfrH5ej
```

- With `--initialize-insecure`, (either with or without `--initialize` because `--initialize-insecure` implies `--initialize`), the server does not generate a password or mark it expired, and writes a warning message:

```
[Warning] root@localhost is created with an empty password ! Please
consider switching off the --initialize-insecure option.
```

For instructions on assigning a new `'root'@'localhost'` password, see Post-Initialization root Password Assignment.

5. The server populates the server-side help tables used for the `HELP` statement (see HELP Statement). The server does not populate the time zone tables. To do so manually, see MySQL Server Time Zone Support.

6. If the `init_file` system variable was given to name a file of SQL statements, the server executes the statements in the file. This option enables you to perform custom bootstrapping sequences.

   When the server operates in bootstrap mode, some functionality is unavailable that limits the statements permitted in the file. These include statements that relate to account management (such as `CREATE USER` or `GRANT`), replication, and global transaction identifiers.

7. The server exits.

Post-Initialization root Password Assignment

After you initialize the data directory by starting the server with `--initialize` or `--initialize-insecure`, start the server normally (that is, without either of those options) and assign the `'root'@'localhost'` account a new password:

1. Start the server. For instructions, see Starting the Server.

2. Connect to the server:

   - If you used `--initialize` but not `--initialize-insecure` to initialize the data directory, connect to the server as `root`:

     ```
     mysql -u root -p
     ```

     Then, at the password prompt, enter the random password that the server generated during the initialization sequence:

     ```
     Enter password: (enter the random root password here)
     ```

     Look in the server error log if you do not know this password.

   - If you used `--initialize-insecure` to initialize the data directory, connect to the server as `root` without a password:

     ```
     mysql -u root --skip-password
     ```

3. After connecting, use an `ALTER USER` statement to assign a new `root` password:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
```

See also Securing the Initial MySQL Account.

Note

Attempts to connect to the host `127.0.0.1` normally resolve to the `localhost` account. However, this fails if the server is run with `skip_name_resolve` enabled. If you plan to do that, make sure that an account exists that can accept a connection. For example, to be able to connect as `root` using `--host=127.0.0.1` or `--host=::1`, create these accounts:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@'::1' IDENTIFIED BY 'root-password';
```

It is possible to put those statements in a file to be executed using the `init_file` system variable, as discussed in Server Actions During Data Directory Initialization.