MySQL Router 8.0

Abstract

MySQL Router is part of InnoDB cluster, and is lightweight middleware that provides transparent routing between your application and back-end MySQL Servers. It can be used for a wide variety of use cases, such as providing high availability and scalability by effectively routing database traffic to appropriate back-end MySQL Servers. The pluggable architecture also enables developers to extend MySQL Router for custom use cases. For additional details about how MySQL Router is part of InnoDB cluster, see InnoDB Cluster.

MySQL Router 8.0 is highly recommended for use with MySQL Server 8.0 and 5.7.

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

If you have not yet installed MySQL Router, download it from the download site.

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

Document generated on: 2020-03-17 (revision: 65332)

Table of Contents

Preface and Legal Notices

This is the MySQL Router manual. This document covers MySQL Router.

Licensing information. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL Router, see MySQL Router Commercial 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 Router, see >MySQL Router Community 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

your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Chapter 1 General Information

Table of Contents

MySQL Router is a building block for high availability (HA) solutions. It simplifies application development by intelligently routing connections to MySQL servers for increased performance and reliability.

MySQL Router 8 fully supports MySQL 5.7 and MySQL 8, and it replaces the MySQL Router 2.x series. If you currently use Router 2.0 or 2.1 then we recommend upgrading your installation to MySQL Router 8.

1.1 Routing for MySQL InnoDB cluster

MySQL Router is part of InnoDB cluster and is lightweight middleware that provides transparent routing between your application and back-end MySQL Servers. It is used for a wide variety of use cases, such as providing high availability and scalability by routing database traffic to appropriate back-end MySQL servers. The pluggable architecture also enables developers to extend MySQL Router for custom use cases.

For additional details about how Router is part of InnoDB cluster, see InnoDB Cluster.

Introduction

For client applications to handle failover, they need to be aware of the InnoDB cluster topology and know which MySQL instance is the PRIMARY. While it is possible for applications to implement that logic, MySQL Router can provide and handle this functionality for you.

MySQL uses Group Replication to replicate databases across multiple servers while performing automatic failover in the event of a server failure. When used with a MySQL InnoDB cluster, MySQL Router acts as a proxy to hide the multiple MySQL instances on your network and map the data requests to one of the cluster instances. As long as there are enough online replicas and communication between the components is intact, applications will be able to contact one of them. MySQL Router also makes this possible by having applications connect to MySQL Router instead of directly to MySQL.

Deploying Router with MySQL InnoDB cluster

The recommended deployment model for MySQL Router is with InnoDB cluster, with Router sitting on the same host as the application.

The steps for deploying MySQL Router with an InnoDB cluster after configuring the cluster are:

1. Install MySQL Router.
2. Bootstrap InnoDB cluster, and test.

   Bootstrapping automatically configures MySQL Router for an existing InnoDB cluster by using `--bootstrap` and other command-line options. During bootstrap, Router connects to the cluster, fetches its metadata, and configures itself for use. Bootstrapping is optional.

For additional information, see Chapter 3, *Deploying MySQL Router*.

3. Set up MySQL Router for automatic startup.

   Configure your system to automatically start MySQL Router when the host is rebooted, a process similar to how the MySQL server is configured to start automatically. For additional details, see Section 5.1, “Starting MySQL Router”.

For example, after creating a MySQL InnoDB cluster, you might configure MySQL Router using:

```
shell> mysqlrouter --bootstrap localhost:3310 --directory /opt/myrouter --user snoopy
```

This example bootstraps MySQL Router to an existing InnoDB cluster where:

- `localhost:3310` is a member of an InnoDB cluster, and either the PRIMARY or bootstrap will redirect to a PRIMARY in the cluster.
- Because the optional `--directory` bootstrap option was used, this example creates a self-contained installation with all generated directories and files at `/opt/myrouter/`. These files include `start.sh`, `stop.sh`, `log/`, and a fully functional MySQL Router configuration file named `mysqlrouter.conf`.
- Only the host's system user named `snoopy` will have access to `/opt/myrouter/*`.

See `--bootstrap` and related options for ways to modify the bootstrap configuration process. For example, passing in `--conf-use-sockets` enables Unix domain socket connections because only TCP/IP connections are enabled by default.

1.2 Cluster Metadata and State

MySQL Router works by sitting in between applications and MySQL servers. Applications connect to Router normally as if they were connecting to an ordinary MySQL server. Whenever an application connects to Router, Router chooses a suitable MySQL server from the pool of candidates that it knows about, and then connects to it. From that moment on, Router forwards all network traffic between the application and MySQL, including responses coming back from it.

MySQL Router keeps a cached list of the online MySQL servers, or the topology and state of the configured InnoDB cluster. Initially, the list is loaded from Router's configuration file when Router is started. This list was generated with InnoDB cluster servers when Router was bootstrapped using the `--bootstrap` option.

To keep the cache updated, the metadata cache component keeps an open connection to one of the InnoDB cluster servers that contains metadata. It does so by querying the metadata database and live state information from MySQL's performance schema. The cluster metadata is changed whenever the InnoDB cluster is modified, such as adding or removing a MySQL server using the MySQL Shell, and the performance_schema tables are updated in real-time by the MySQL server's Group Replication plugin whenever a cluster state change is detected. For example, if one of the MySQL servers had an unexpected shutdown.

When Router detects that a connected MySQL server shuts down, for example because the metadata cache has lost its connection and can not connect again, it attempts to connect to a different MySQL server to fetch metadata and InnoDB cluster state from the new MySQL server.

Note

Dropping cluster metadata using MySQL Shell, such as `dba.dropMetadataSchema()`, causes Router to drop all current connections and forbid new connections. This causes a full outage.

Application connections to a MySQL server that shuts down are automatically closed. They must then reconnect to Router, which redirects them to an online MySQL server.

1.3 Connection Routing

Connection routing means redirecting MySQL connections to an available MySQL server. MySQL packets are routed in their entirety without inspection. For an example deployment using basic connection routing, see Section 3.3, “Basic Connection Routing”.

Applications connect to MySQL Router and not directly to MySQL Server, and if the connection fails then applications are designed to retry the connection because MySQL Router selects a new MySQL server after failed attempts. This is also called simple redirect connection routing because it requires the application to retry the connection. That is, if a connection from MySQL Router to the MySQL server is interrupted, the application encounters a connection failure. However, a new connection attempt triggers Router to find and connect to another MySQL server.

Routed servers and routing strategies are defined in a configuration file. For example, the following section tells MySQL Router to listen for connections on port 7002 of the localhost, and then redirect those connections to a MySQL instance defined by the `destinations` option, including servers running on the localhost listening on ports 3306, 3307, and 3308. We also use the `routing_strategy` option to use the round robin form of load-balancing. For additional information, see Section 4.3, “Configuration Options”

```
[routing:simple_redirect]
bind_port = 7002
routing_strategy = round-robin
destinations = localhost:3306,localhost:3307,localhost:3308
```

This example section is titled `routing:simple_redirect`. The first part, `routing`, is the section name used internally to determine which plugin to load. The second part, `simple_redirect`, is an optional section key to differentiate between other routing strategies.

When a server is no longer reachable, MySQL Router moves to the next server destination in the list and circles back to the first server destination if the list is exhausted as per the round-robin strategy.

Note

Before MySQL Router 8.0, the now deprecated `mode` option was used instead of the `routing_strategy` option that was added in MySQL Router 8.0.

1.4 Application Considerations

MySQL Router usage does not require specific libraries or interfaces. Aside from managing the MySQL Router instance, write your application as if MySQL Router was a typical MySQL instance.

The only difference when using MySQL Router is how you make connections to the MySQL server. Applications using a single MySQL connection at startup that does not test for connection errors must be updated. This is because MySQL Router redirects connections when the connection is attempted and does not read packets or perform an analysis. If a MySQL server fails, Router returns the connection error to the application.

For these reasons, the application should be written to test for connection errors and, if encountered, retry the connection. If this technique or one similar is employed in your application then using MySQL Router will not require any extra effort.

The following gives a better sense of why you may want to use MySQL Router and looks into how it is used from an application's point of view.

Scenarios

There are several possible scenarios for MySQL Router, including:

- As a developer, I want my application to connect to a service so it gets a connection to, by default, the current primary of a group replication cluster.
- As an administrator, I want to set up multiple services so MySQL Router listens on a different port for each highly available replica set.
- As an administrator, I want to be able to run a connection routing service on port 3306 so it is more transparent to a user or application.
- As an administrator, I want to configure a mode for each connection routing service so I can specify whether a primary or secondary is returned.

Workflow with MySQL Router

The workflow for using MySQL Router is as follows:

1. MySQL Client or Connector connects to MySQL Router to, for example, port 6446.
2. Router checks for an available MySQL server.
3. Router opens a connection to a suitable MySQL server.
4. Router forwards packets back and forth, between the application and the MySQL server
5. Router disconnects the application if the connected MySQL server fails. The application can then retry connecting to Router, and Router then chooses a different and available MySQL server.

Connections using MySQL Router

An application connects to MySQL Router, and Router connects the application to an available MySQL server.

This example demonstrates that a connection transparently connects to one of the InnoDB cluster instances. Because this example uses a sandboxed InnoDB cluster where all instances run on the same host, we check the `port` status variable to see which MySQL instance is connected.

Make a connection to MySQL Router using the MySQL client, for example:

```
shell> mysql -u root -h 127.0.0.1 -P 6446 -p
```

These port numbers depend on your configuration, but compare ports in this example:

```
mysql> select @@port;
+--------+
| @@port |
+--------+
|   3310 |
+--------+
1 row in set (0.00 sec)
```

To summarize, the client (application) connected to port 6446 but is connected to a MySQL instance on port 3310.

Recommendations

The following are recommendations for using MySQL Router.

- Install and run MySQL Router on the same host as the application. For a list of reasons, see Chapter 3, *Deploying MySQL Router*.
- Bind Router to localhost using `bind_port = 127.0.0.1:<port>` in the configuration file. Alternatively, on Linux, disable TCP connections (see `--conf-skip-tcp`) and limit this to only using Unix socket connections (see `--conf-use-sockets`).

1.5 What's New in MySQL Router 8.0

This section summarizes many of the new features added to MySQL Router 8.0, in relation to MySQL Router 2.1.

Version Numbering

MySQL Router 8.0.3 is the first 8.0.x release to use the new numbering, and is the successor to MySQL Router 2.1.4.

MySQL Connectors and other MySQL client tools and applications now synchronize the first digit of their version number with the (highest) MySQL server version they support. This change makes it easy and intuitive to decide which client version to use for which server version. Similarly, MySQL Router now uses the same version number as MySQL Server.

New Features and Changes

- The optional `routing_strategy` configuration option was added. The available values are `first-available`, `next-available`, `round-robin`, and `round-robin-with-fallback`.

  Previously, these strategies were described as scheduling modes by the `mode` configuration option where the read-write mode defaults to the first-available strategy, and the read-only mode defaults to the round-robin strategy. This preserves previous behavior for these modes.

- The `--ssl-key` and `--ssl-cert` optional bootstrap command-line options were added. They directly use their MySQL client's counterparts, and specify the client-side certificate and private key to facilitate client-side authentication. This is useful when the root account used during bootstrap was created with REQUIRE X509, which requires the client to authenticate itself when logging in.

- The new `connect_timeout` and `read_timeout` metadata configuration file options were added. These are defined under the [DEFAULT] namespace and affect internal operations, such as metadata server connections.

- Bootstrap now accepts any member of an InnoDB cluster and automatically finds and reconnects to a writable primary. Before, only the primary was accepted.

- Bootstrap now accepts the `--config` option and reads the *[logger]* `level` option's definition.

- The maximum number of concurrent client connections was increased from about 500 to over 5000, a limit now dependent on the operation system. To achieve this, select() based fd event calls were replaced by poll() (or WSAPoll() on Windows).

- A new `mysqlrouter_plugin_info` utility was added to help debug MySQL Router plugins. It provides information such as the plugin version, description, ABI version, requirements, and function pointers.

Additional Changes

For complete list of all changes introduced in MySQL Router 8.0, see the MySQL Router 8.0 Release Notes

Chapter 2 Installing MySQL Router

Table of Contents

This chapter describes how to obtain and install MySQL Router. Downloads are available from the download site.

2.1 Installing MySQL Router on Linux

There are binary distributions of MySQL Router available for several variants of Linux, including Fedora, Oracle Linux, Red Hat, and Ubuntu.

Installation options include:

- **Official MySQL Yum or APT repository packages**: These binaries are built by the MySQL Release team. For additional information about installing these, see the quick guides for installing them using Yum or APT.
- **Download official MySQL packages**: Downloads are available at https://dev.mysql.com/downloads/router. Download and install using your preferred package manager.

  Alternatively, MySQL Router is included in MySQL Server's source and monolithic binary packages as of MySQL Router 8.0.13.
- **Download the source code and compile yourself**: The source code is available at https://dev.mysql.com/downloads/router as a `tar.gz` or RPM package. Alternatively, the source code is also available on GitHub.

  For information about compiling MySQL Router, see Installing MySQL Router from Source Code.

The procedure for installing on Linux depends on your Linux distribution.

Installing MySQL Router using an official DEB or RPM package creates a local system user and group named "mysqlrouter" on the host that MySQL Router runs as by default. For additional information, see the system `user`'s configuration option.

Installing DEB packages

On Ubuntu, and other systems that use the Debian package scheme, you can either download and install .deb packages or use the APT package manager.

Using the APT Package Manager

1. Install the MySQL APT repository as described in the MySQL APT Repository documentation. For example:

Note

Download the APT configuration package from here.

```
shell> sudo dpkg -i mysql-apt-config_0.8.8-1_all.deb
```

Enable the "MySQL Tools & Connectors" on the configuration screen.

2. Update your APT repository:

```
shell> sudo apt-get update
```

3. Install MySQL Router. For example:

```
shell> sudo apt-get install mysql-router
```

Manually Installing a Package

You can also download the .deb package and install it from the command line similarly to

```
shell> sudo dpkg -i package.deb
```

package`.deb` is the MySQL Router package name; for example, `mysql-router-`*version*`-1ubu1604-amd64.deb`, where *version* is the MySQL Router version number.

Installing RPM packages

On RPM-based systems, you can either download and install RPM packages or use the Yum package manager.

Using the Yum Package Manager

- First, install the MySQL Yum repository as described in the MySQL Yum Repository documentation. For example:

Note

Download the Yum configuration package from here.

```
shell> sudo rpm -Uvh mysql57-community-release-el7-11.noarch.rpm
```

- Next, install MySQL Router. For example:

```
shell> sudo yum install mysql-router
```

Manually Installing an RPM Package

```
shell> sudo rpm -i package.rpm
```

package`.rpm` is the MySQL Router package name; for example, `mysql-router-`*version*`-el7.x86_64.rpm`, where *version* is the MySQL Router version number.

Uninstalling

The procedure for uninstalling MySQL Router on Linux depends on the package you are using.

Uninstalling DEB packages

To uninstall a Debian package, use this command:

```
shell> sudo dpkg -r mysql-router
```

This command does not remove the configuration files. To also remove them and the data directory, use:

```
shell> sudo dpkg --purge mysql-router
```

Note

Alternatively, use `apt-get remove mysql-router` or `apt-get purge mysql-router`.

Uninstalling RPM packages

To uninstall an RPM package, use this command:

```
shell> sudo rpm -e mysql-router
```

Note

Similarly, use `yum remove mysql-router`.

This command does not remove the configuration files.

What Is Not Removed

When not purging, the uninstallation process does not remove your configuration files. On Debian systems, this might include files such as:

```
/etc/init.d/mysqlrouter
/etc/mysqlrouter/mysqlrouter.conf
/etc/apparmor.d/usr.sbin.mysqlrouter
```

2.2 Installing MySQL Router on macOS

Download the DMG archive from https://dev.mysql.com/downloads/router/, and execute it to install MySQL Router.

Alternatively, download, unpack, and manually install the compressed `.tar.gz` file.

2.3 Installing MySQL Router on Windows

MySQL Router for Windows can be installed using the MySQL Installer that installs and updates all MySQL products on Windows, or by downloading the ZIP Archive.

Windows Prerequisites

For the Community version of MySQL Router: The Visual C++ Redistributable for Visual Studio 2015 (available at the Microsoft Download Center) is required. Install it before installing MySQL Router on Windows.

Installing Using MySQL Installer

The general MySQL Installer download is available at https://dev.mysql.com/downloads/windows/installer/. The MySQL Installer application can install, upgrade, and manage most MySQL products, including MySQL Router. MySQL Installer also includes an option to bootstrap MySQL Router with a MySQL InnoDB cluster.

Recommended Approach

Managing all of your MySQL products, including MySQL Router, with MySQL Installer is the recommended approach. It handles all requirements, prerequisites, configuration procedures, and upgrades.

When executing MySQL Installer, you may choose MySQL Router as one of the products to install or upgrade.

MySQL Router is typically installed in `C:\%PROGRAMFILES%\MySQL\MySQL Router 8.0`, where *`%PROGRAMFILES%`* is the default directory for programs for your locale. The `%PROGRAMFILES%` directory is defined as `C:\Program Files\` on most systems.

For information about installing and starting Router as a Windows service, see Section 5.1, “Starting MySQL Router”.

Installing the ZIP Archive

The ZIP Archive download is available at https://dev.mysql.com/downloads/router/.

Unlike installing with MySQL Installer, unpacking the MySQL Router ZIP archive does not check for dependencies on your system, such as the required VC++ 2015 runtime. When installing MySQL Router using the ZIP archive, download and install Visual C++ Redistributable for Visual Studio 2015 before using MySQL Router.

After installing the prerequisites, unzip the ZIP Archive and execute `bin/mysqlrouter.exe` as you normally would.

For information about installing and using MySQL Router as a Windows service, see Section 5.1, “Starting MySQL Router”.

2.4 Installing MySQL Router from Source Code

The MySQL Router is written using the C++11 standard. As such, you must compile the code before you can install it. Compilation is typical of most C++ applications, as demonstrated below.

The CMake program provides control over how you configure a MySQL Router source distribution. Typically, you do this using options on the CMake command line. For information about options supported by CMake, run either of these commands in the top-level MySQL Router source directory:

```
shell> cmake . -LH
shell> ccmake .
```

The default CMake installation prefixes are used. It is different for each platform, but for most Unix-like platforms it is "`/usr/local`". It is possible to alter the installation path with the CMake variable "**CMAKE_INSTALL_PREFIX**". For example:

```
shell> mkdir build && cd build
shell> cmake .. -DINSTALL_LAYOUT=STANDALONE -DCMAKE_INSTALL_PREFIX=/opt/mysql/router8.0
```

Notice we use the `-DINSTALL_LAYOUT=STANDALONE` option to use the same installation layout as used for .tar.gz and .zip packages. This is the recommended setting for building the source.

Note

The CMake options are not documented here, but they are similar to the MySQL Server CMake options. For additional (related) information, see MySQL Source-Configuration Options.

Download and unpack the source files, and then follow the steps specific to your platform.

Linux and macOS

```
shell> tar xzf mysql-router-8.0.21-src.tar.gz
shell> cd mysql-router-8.0.21-src
```

Once this is complete, you need to configure and compile MySQL Router using cmake. Our examples use the default installation location of `/usr/local`.

Note

Installing MySQL Router generates a file named `install_manifest.txt` that lists all files (with paths) installed on the system. This file is useful for uninstalling MySQL Router.

However, there are prerequisites for compiling the MySQL Router source code.

2.4.1 Prerequisites

The following components and libraries are required to compile MySQL Router on Linux:

- An operating system with a compiler that supports **C++11**.

  Example systems that include this support are Ubuntu 14.04 and later, Oracle Linux 7, and macOS 10.10 and later.

  Note

  Enterprise Linux 6: compilation works but requires the Software Collection Library 1.2. For Oracle Linux, see Docs and Downloads. For RedHat and CentOS, see Docs and Downloads.

- MySQL Server 5.5 or higher client libraries and header files. For example, on Ubuntu this is the `libmysqlclient-dev` package.
- Code development tools including gcc, make, and assorted utilities for C++ 11 including GCC 4.8 and later, glibc 2.17 and later, and clang 3.3 and later
- SLES 12 considerations: as of MySQL 8.0.13, these binaries are built using GCC 7; and the lowest supported GCC version on this platform is now 5.3. This change means the GCC Devel repository is required. For example:

```
shell> cd /etc/zypp/repos.d/
shell> wget https://download.opensuse.org/repositories/devel:/gcc/SLE-12/devel:gcc.repo
```

- CMake 2.8.9 or later.
- Protobuf 3.0

Note

If your MySQL Server installation does not include the header files and compiled client libraries, then you may need to download the MySQL Server source code.

2.4.2 Compiling the Source Code

To compile the source code, you should create a folder to contain the compiled binaries and executables, run cmake to create the make file, then compile the code. The following demonstrates the steps needed on a Ubuntu machine. Other platforms are similar.

Note

For some platforms, such as Oracle Enterprise Linux 6, you may also need to install the *devtoolset* software collection.

If you get an error stating that the MySQL libraries cannot be found, then check the listed paths. If the client libraries or the `include` folder does not exist, you may need to reference a compiled copy of the MySQL Server source code by using the `-DWITH_MYSQL=<path to server code>` option. More specifically, the compiler needs to be able to find the MySQL client libraries and

include files. If libmysqlclient is stored elsewhere, then `-DMySQL_CLIENT_LIB=/path/to/libmysqlclient.so` can also be used. A compiled server source code tree will have these files. So too will most installations of the MySQL server.

For example, on Debian and RPM-based platforms, you would need the packages which contain the libraries and the development (include) files. If you installed MySQL from a platform-specific repository, you would need to install the `mysql-community-libs` and `mysql-community-devel` packages.

Note

If you change anything and need to recompile from scratch, be sure to delete the `CMakeCache.txt` file before running the `cmake` command.

Begin by running the `cmake` command to create the makefile. The following commands are run from the root of the MySQL Router source code tree. You should see similar results with the appropriate paths for your system.

```
shell>  mkdir build
shell>  cd build
shell>  cmake ..  -DWITH_MYSQL=<path to binaries and libraries>
-- The C compiler identification is GNU 4.9.2
-- The CXX compiler identification is GNU 4.9.2
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Loading internal repository
-- Installation layout set to DEFAULT
-- Adding MySQL Harness from /home/cbell/source/git/mysql-router-2.0.2/mysql_harness
-- Harness will install plugins in lib/mysqlrouter
-- MySQL Harness CPU Descriptor is x86_64
-- MySQL Harness OS Descriptor is linux
-- MySQL Harness Compiler Descriptor is gnu-3
-- MySQL Harness Runtime Descriptor is *
-- Found Doxygen: /usr/bin/doxygen (found version "1.8.9.1")
-- Performing Test COMPILER_SUPPORTS_CXX11
-- Performing Test COMPILER_SUPPORTS_CXX11 - Success
-- Performing Test COMPILER_SUPPORTS_CXX0X
-- Performing Test COMPILER_SUPPORTS_CXX0X - Success
-- Looking for include file pthread.h
-- Looking for include file pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Performing Test support_11
-- Performing Test support_11 - Success
-- Performing Test support_0x
-- Performing Test support_0x - Success
-- Found MySQL Libraries 5.6.27; using <path to server code>/lib/libmysqlclient.so
-- Loading module 'router'
-- Loading module 'routing'
-- Configuring done
-- Generating done
-- Build files have been written to: <path to router code>/build
```

Next, compile the code. For this we only need the `make` command as shown. Again, you should see similar results on your system.

```
shell>  make
Scanning dependencies of target harness-archive
[  2%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/loader.cc.o
```

```
[  5%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/utilities.cc.o
[  8%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/config_parser.cc.o
[ 11%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/designator.cc.o
[ 14%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/filesystem-posix.cc.o
Linking CXX static library libmysqlharness.a
[ 14%] Built target harness-archive
Scanning dependencies of target harness-library
[ 17%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/loader.cc.o
[ 20%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/utilities.cc.o
[ 22%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/config_parser.cc.o
[ 25%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/designator.cc.o
[ 28%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/filesystem-posix.cc.o
Linking CXX shared library libmysqlharness.so
[ 28%] Built target harness-library
Scanning dependencies of target logger
[ 31%] Building CXX object harness/plugins/logger/CMakeFiles/logger.dir/logger.cc.o
Linking CXX shared library ../../../stage/lib/mysqlrouter/logger.so
[ 31%] Built target logger
Scanning dependencies of target keepalive
[ 34%] Building CXX object harness/plugins/keepalive/CMakeFiles/keepalive.dir/src/keepalive.cc.o
Linking CXX shared library ../../../stage/lib/mysqlrouter/keepalive.so
[ 34%] Built target keepalive
Scanning dependencies of target router_lib
[ 37%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/router_app.cc.o
[ 40%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/arg_handler.cc.o
[ 42%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/utils.cc.o
[ 45%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/datatypes.cc.o
[ 48%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/plugin_config.cc.o
Linking CXX shared library ../../../stage/lib/libmysqlrouter.so
[ 48%] Built target router_lib
Scanning dependencies of target mysqlrouter
[ 51%] Building CXX object src/router/src/CMakeFiles/mysqlrouter.dir/main.cc.o
Linking CXX executable ../../../stage/bin/mysqlrouter
[ 51%] Built target mysqlrouter
Scanning dependencies of target routing
[ 77%] Building CXX object src/routing/CMakeFiles/routing.dir/src/routing_plugin.cc.o
[ 80%] Building CXX object src/routing/CMakeFiles/routing.dir/src/plugin_config.cc.o
[ 82%] Building CXX object src/routing/CMakeFiles/routing.dir/src/mysql_routing.cc.o
[ 85%] Building CXX object src/routing/CMakeFiles/routing.dir/src/utils.cc.o
[ 88%] Building CXX object src/routing/CMakeFiles/routing.dir/src/destination.cc.o
[ 94%] Building CXX object src/routing/CMakeFiles/routing.dir/src/dest_first_available.cc.o
[ 97%] Building CXX object src/routing/CMakeFiles/routing.dir/src/uri.cc.o
[100%] Building CXX object src/routing/CMakeFiles/routing.dir/src/routing.cc.o
Linking CXX shared library ../../stage/lib/mysqlrouter/routing.so
[100%] Built target routing
```

2.4.3 Installing from Source Code

Once the source code is compiled, you can install the MySQL Router on your system with the following command. Note that you may need elevated privileges (e.g. sudo) to install.

```
shell>  sudo make install
[ 14%] Built target harness-archive
[ 28%] Built target harness-library
[ 31%] Built target logger
[ 34%] Built target keepalive
[ 48%] Built target router_lib
[ 51%] Built target mysqlrouter
[100%] Built target routing
Install the project...
-- Install configuration: ""
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/loader.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/filesystem.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/plugin.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/config_parser.h
-- Installing: /usr/local/lib/libmysqlharness.a
-- Installing: /usr/local/lib/libmysqlharness.so.0
-- Up-to-date: /usr/local/lib/libmysqlharness.so
-- Set runtime path of "/usr/local/lib/libmysqlharness.so.0" to "$ORIGIN/../lib"
-- Installing: /usr/local/lib/mysqlrouter/keepalive.so
-- Set runtime path of "/usr/local/lib/mysqlrouter/keepalive.so" to "$ORIGIN"
```

```
-- Installing: /usr/local/lib/mysqlrouter/logger.so
-- Set runtime path of "/usr/local/lib/mysqlrouter/logger.so" to "$ORIGIN"
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/logger.h
-- Up-to-date: /usr/local/share/doc/mysqlrouter/README.txt
-- Up-to-date: /usr/local/share/doc/mysqlrouter/License.txt
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/plugin_config.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/utils.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/datatypes.h
-- Installing: //var
-- Installing: //var/local
-- Installing: //var/local/mysqlrouter
-- Installing: //var/local/mysqlrouter/log
-- Installing: //var
-- Installing: //var/local
-- Installing: //var/local/mysqlrouter
-- Installing: //var/local/mysqlrouter/run
-- Installing: /usr/local/etc
-- Installing: /usr/local/etc/mysqlrouter
-- Installing: /usr/local/bin/mysqlrouter
-- Set runtime path of "/usr/local/bin/mysqlrouter" to "$ORIGIN/../lib"
-- Installing: /usr/local/lib/libmysqlrouter.so.1
-- Up-to-date: /usr/local/lib/libmysqlrouter.so
-- Set runtime path of "/usr/local/lib/libmysqlrouter.so.1" to "$ORIGIN/../lib"
-- Installing: /usr/local/lib/mysqlrouter/routing.so
-- Set runtime path of "/usr/local/lib/mysqlrouter/routing.so" to "$ORIGIN"
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/routing.h
```

2.4.4 Testing the Installation

You can ensure the installation succeeded by running the following command. You should see a similar output on your system. An example of setting the Router for simple routing is available at Section 3.2, “Trying out MySQL Router in a Sandbox”

Note

Our example assumes that `mysqlrouter` is in the system's PATH. In this case, PATH includes `/usr/local/bin`.

```
shell> mysqlrouter --help

...

Start MySQL Router.

Configuration read from the following files in the given order (enclosed
in parentheses means not available for reading):
  (/etc/mysqlrouter/mysqlrouter.conf)
  /home/philip/.mysqlrouter.conf

...
```

Note

Use the `mysqlrouter --version` command to check the version.

Chapter 3 Deploying MySQL Router

Table of Contents

Performance Recommendations

For best performance, MySQL Router is typically installed on the same host as the application that uses it. Possible reasons include:

- To allow local UNIX domain socket connections to the application, instead of TCP/IP.

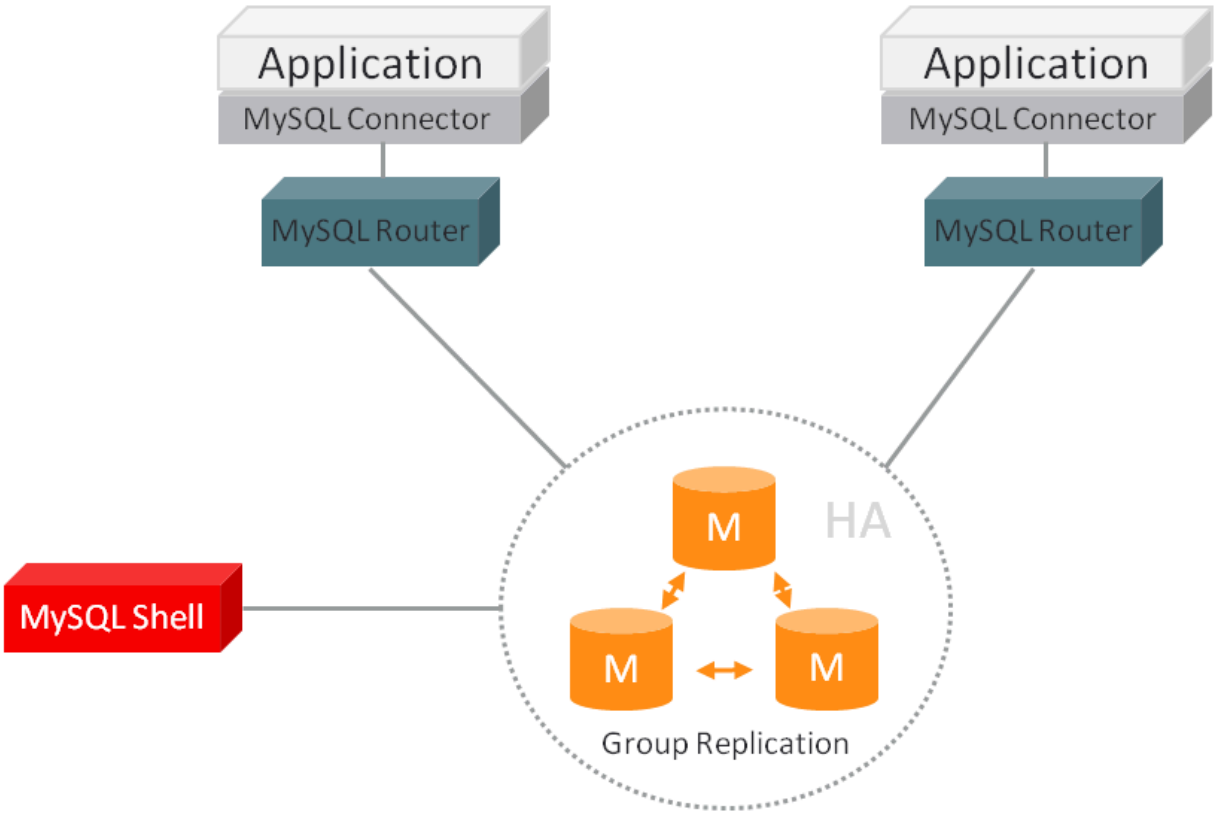
Note

Unix domain sockets can function with applications connecting to MySQL Router, but not for MySQL Router connecting to a MySQL Server.

- To decrease network latency.
- To allow MySQL Router to connect to MySQL without requiring extra accounts for the Router's host, for MySQL accounts that are created specifically for application hosts such as *myapp@198.51.100.45* instead of a value like *myapp@%*.
- Typically application servers are easiest to scale.

You can run multiple MySQL Router instances on your network, and you do not need to isolate MySQL Router to a single machine. This is because MySQL Router has no affinity for any particular server or host.

Figure 3.1 Example MySQL Router Deployment



3.1 Bootstrapping MySQL Router

Here is a brief example to demonstrate how MySQL Router can be deployed to use an InnoDB cluster using bootstrapping. For additional information, see `--bootstrap` and the other bootstrap options.

This example creates a standalone MySQL Router instance using the `--directory` option, enables sockets, and assumes that an InnoDB cluster named `myCluster` already exists:

```
shell> mysqlrouter --bootstrap root@localhost:3310 --directory /tmp/myrouter --conf-use-sockets

Please enter MySQL password for root:

# Bootstrapping MySQL Router instance at '/tmp/myrouter'...

- Checking for old Router accounts
  - No prior Router accounts found
- Creating mysql account 'mysql_router1_b8z6shsk9uyp'@'%' for cluster management
- Storing account in keyring
- Adjusting permissions of generated files
- Creating configuration /tmp/myrouter/mysqlrouter.conf

# MySQL Router configured for the InnoDB cluster 'myCluster'

After this MySQL Router has been started with the generated configuration

    $ mysqlrouter -c /tmp/myrouter/mysqlrouter.conf

the cluster 'myCluster' can be reached by connecting to:

## MySQL Classic protocol

- Read/Write Connections: localhost:6446
- Read/Only Connections:  localhost:6447

## MySQL X protocol

- Read/Write Connections: localhost:64460
- Read/Only Connections:  localhost:64470
```

At this point the bootstrap process has created a `mysqlrouter.conf` file with the required files at the directory specified, and the result shows you how to start this MySQL Router instance. A generated MySQL Router directory looks similar to:

```
shell> ls -l | awk '{print $9}'

data/
log/
mysqlrouter.conf
mysqlrouter.key
run/
start.sh*
stop.sh*
```

A generated MySQL Router configuration file (`mysqlrouter.conf`) looks similar to:

```
# File automatically generated during MySQL Router bootstrap
[DEFAULT]
logging_folder=/tmp/myrouter/log
runtime_folder=/tmp/myrouter/run
data_folder=/tmp/myrouter/data
keyring_path=/tmp/myrouter/data/keyring
master_key_path=/tmp/myrouter/mysqlrouter.key
connect_timeout=15
read_timeout=30
dynamic_state=/tmp/myrouter/data/state.json

[logger]
level = INFO
```

```
[metadata_cache:myCluster]
router_id=5
user=mysql_router5_p79mg6q6ytrw
metadata_cluster=myCluster
ttl=0.5

[routing:myCluster_default_rw]
bind_address=0.0.0.0
bind_port=6446
socket=/tmp/myrouter/mysql.sock
destinations=metadata-cache://myCluster/default?role=PRIMARY
routing_strategy=round-robin
protocol=classic

[routing:myCluster_default_ro]
bind_address=0.0.0.0
bind_port=6447
socket=/tmp/myrouter/mysqlro.sock
destinations=metadata-cache://myCluster/default?role=SECONDARY
routing_strategy=round-robin
protocol=classic

[routing:myCluster_default_x_rw]
bind_address=0.0.0.0
bind_port=64460
socket=/tmp/myrouter/mysqlx.sock
destinations=metadata-cache://myCluster/default?role=PRIMARY
routing_strategy=round-robin
protocol=x

[routing:myCluster_default_x_ro]
bind_address=0.0.0.0
bind_port=64470
socket=/tmp/myrouter/mysqlxro.sock
destinations=metadata-cache://myCluster/default?role=SECONDARY
routing_strategy=round-robin
protocol=x
```

In this example, MySQL Router configured four ports and four sockets. Ports are added by default, and sockets were added by passing in `--conf-use-sockets`. The InnoDB cluster named "myCluster" is the source of the metadata, and the `destinations` are using the InnoDB cluster metadata cache to dynamically configure host information. The related command line options:

- `--conf-use-sockets`: Optionally enable UNIX domain sockets for all four connection types, as demonstrated in the example.
- `--conf-skip-tcp`: Optionally disable TCP ports, an option to pass in with `--conf-use-sockets` if you only want sockets.
- `--conf-base-port`: Optionally change the range of ports rather than using the default ports. This sets the port for classic read-write (PRIMARY) connections, and defaults to *6446*.
- `--conf-bind-address`: Optionally change the bind_address value for each route.

To demonstrate MySQL Router's behavior, the following client (application) connects to port 6446 but is connected to a MySQL instance on port 3310.

```
shell> mysql -u root -h 127.0.0.1 -P 6446 -p

...

mysql> select @@port;
+--------+
| @@port |
+--------+
|   3310 |
+--------+
1 row in set (0.00 sec)
```

For additional examples, see Set Up a MySQL Server Sandbox and Sandbox Deployment of InnoDB Cluster.

3.2 Trying out MySQL Router in a Sandbox

Test a MySQL Router installation by setting up a Router sandbox with InnoDB cluster. In this case, Router acts as an intermediate node redirecting client connections to a list of servers. If one server fails, clients are redirected to the next available server in the list.

Set Up a MySQL Server Sandbox

Begin by starting three MySQL Servers. You can do this in a variety of ways, including:

- Using the MySQL Shell AdminAPI interface that InnoDB cluster provides. This is the recommended and simplest approach, and is documented in this section. For additional information, see InnoDB Cluster.

  For a scripted approach, see either Scripting AdminAPI or https://github.com/mattlord/Docker-InnoDB-Cluster.

- By installing three MySQL Server instances on three different hosts, or on the same host.
- Using the `mysql-test-run.pl` script that is part of the MySQL Test Suite framework. For additional information, see The MySQL Test Suite.
- Using the `mysqlcloneserver` MySQL Utility.

The following example uses the AdminAPI method to set up our cluster sandbox. This is a brief overview, so see Sandbox Deployment of InnoDB Cluster in the InnoDB cluster manual for additional details. The following assumes you have a current version of MySQL Shell, MySQL Server, and MySQL Router installed.

Deploy a Sandbox cluster

This example uses MySQL Shell AdminAPI to set up a InnoDB cluster with three MySQL instances (one primary and two secondaries), and a bootstrapped standalone MySQL Router with a generate configuration file. Output was shortened using "...".

```
shell> mysqlsh

mysql-js> dba.deploySandboxInstance(3310)
  ...
mysql-js> dba.deploySandboxInstance(3320)
  ...
mysql-js> dba.deploySandboxInstance(3330)
  ...

mysql-js> \connect root@localhost:3310
  ...

mysql-js> cluster = dba.createCluster("myCluster")
  ...

mysql-js> cluster.addInstance("root@localhost:3320")
  ...
mysql-js> cluster.addInstance("root@localhost:3330")
  ...

mysql-js> cluster.status()
{
    "clusterName": "myCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "localhost:3310",
        "ssl": "REQUIRED",
        "status": "OK",
```

```
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "localhost:3310": {
                "address": "localhost:3310",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "localhost:3320": {
                "address": "localhost:3320",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "localhost:3330": {
                "address": "localhost:3330",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    },
    "groupInformationSourceMember": "mysql://root@localhost:3310"
}

mysql-js> \q

Bye!
```

Set Up the Router

Next, set up MySQL Router to redirect to these MySQL instances. We'll use bootstrapping (using `--bootstrap`), and create a self-contained MySQL Router installation using `--directory`. This uses the metadata cache plugin to securely store the credentials.

```
shell> mysqlrouter --bootstrap root@localhost:3310 --directory /tmp/myrouter

Please enter MySQL password for root:

Bootstrapping MySQL Router instance at '/tmp/mysqlrouter'...
MySQL Router  has now been configured for the InnoDB cluster 'myCluster'.

The following connection information can be used to connect to the cluster.

Classic MySQL protocol connections to cluster 'myCluster':
- Read/Write Connections: localhost:6446
- Read/Only Connections: localhost:6447

X protocol connections to cluster 'myCluster':
- Read/Write Connections: localhost:64460
- Read/Only Connections: localhost:64470

shell> cd /tmp/myrouter

shell> ./start.sh
```

MySQL Router is now configured and running, and is using the **myCluster** cluster that we set up earlier.

Testing the Router

Now connect to MySQL Router as you would any other MySQL Server by connecting to a configured MySQL Router port.

The following example connects to MySQL Router on port 6446, the port we configured for read-write connections:

```
shell> mysql -u root -h 127.0.0.1 -P 6446 -p
mysql> SELECT @@port;

+--------+
| @@port |
+--------+
|   3310 |
+--------+
```

As demonstrated, we connected to MySQL Router using port 6446 but see we are connected to our MySQL instance on port 3310 (our PRIMARY). Next let's connect to a read-only MySQL instance:

```
shell> mysql -u root -h 127.0.0.1 -P 6447 -p
mysql> SELECT @@port;

+--------+
| @@port |
+--------+
|   3320 |
+--------+
```

As demonstrated, we connected to MySQL Router using port 6447 but see we are connected to our MySQL instance on port 3320 (one of our secondaries). The read-only mode defaults to the round-robin strategy where the next connection refers to a different secondary:

```
shell> mysql -u root -h 127.0.0.1 -P 6447 -p
mysql> SELECT @@port;

+--------+
| @@port |
+--------+
|   3330 |
+--------+
```

As demonstrated, our second read-only connection to port 6447 connected to a different MySQL secondary, in this case to port 3330 instead of 3320.

Now test failover by first killing the primary MySQL instance (port 3310) that we connected to above.

```
shell> mysqlsh --uri root@127.0.0.1:6446

mysql-js> dba.killSandboxInstance(3310)

The MySQL sandbox instance on this host in
/home/philip/mysql-sandboxes/3310 will be killed

Killing MySQL instance...

Instance localhost:3310 successfully killed.
```

You can continue using MySQL Shell to check the connection but let us use the same `mysql` client example we did above:

```
shell> mysql -u root -h 127.0.0.1 -P 6446 -p
mysql> SELECT @@port;

+--------+
| @@port |
+--------+
|   3320 |
+--------+

shell> mysql -u root -h 127.0.0.1 -P 6447 -p
mysql> SELECT @@port;

+--------+
| @@port |
+--------+
|   3330 |
+--------+
```

As shown, despite connecting to the same ports (6446 for the primary and 6447 for a secondary), the underlying ports changed. Our new primary server changed from port 3310 to 3320 while our secondary changed from 3320 to 3330.

We have now demonstrated MySQL Router performing simple redirects to a list of primary and secondary MySQL instances.

3.3 Basic Connection Routing

The *Connection Routing* plugin performs connection-based routing, meaning it forwards packets to the server without inspecting them. This is a simplistic approach that provides high throughput. For additional general information about connection routing, see Section 1.3, “Connection Routing”.

A simple connection-based routing setup is shown below. These and additional options are documented under Section 4.3.2, “Configuration File Options”.

```
[logger]
level = INFO

[routing:secondary]
bind_address = localhost
bind_port = 7001
destinations = foo.example.org:3306,bar.example.org:3306,baz.example.org:3306
routing_strategy = round-robin

[routing:primary]
bind_address = localhost
bind_port = 7002
destinations = foo.example.org:3306,bar.example.org:3306
routing_strategy = first-available
```

Here we use connection routing to round-robin MySQL connections to three MySQL servers on port 7001 as defined by *round-robin* `routing_strategy`. This example also configures the *first-available* strategy for two of the servers using port 7002. The first-available strategy uses the first available server from the destinations list. The number of MySQL instances assigned to each `destinations` is up to you as this is only an example. Router does not inspect the packets and does not restrict connections based on assigned strategy or mode, so it is up the application to determine where to send read and write requests, which is either port 7001 or 7002 in our example.

Note

Before MySQL Router 8.0, the now deprecated `mode` option was used instead of the `routing_strategy` option that was added in MySQL Router 8.0.

Assuming all three MySQL instances are running, next start MySQL Router by passing in the configuration file:

```
shell> ./bin/mysqlrouter -config=/etc/mysqlrouter-config.conf
```

Now MySQL Router is listening to port's 7001 and 7002 and sends requests to the appropriate MySQL instances. For example:

```
shell> ./bin/mysql --user=root --port 7001 --protocol=TCP
```

That will first connect to foo.example.org, and then bar.example.org next, then baz.example.org, and the fourth call goes back to foo.example.org. Instead, we configured port 7002 behavior differently:

```
shell> ./bin/mysql --user=root --port 7002 --protocol=TCP
```

That first connects to foo.example.org, and additional requests will continue connecting to foo.example.org until there is a failure, at which point bar.example.org is now used. For additional information about this behavior, see `mode`.

Chapter 4 Configuration

Table of Contents

MySQL Router is configured using a required configuration file, additional optional configuration files, and some options are also available from the command line.

Bootstrapping is the preferred and common approach to generating a MySQL Router configuration file. For additional information, see `--bootstrap`. Bootstrapping generates a fully functional `mysqlrouter.conf` file.

4.1 Configuration File Syntax

The configuration file format resembles the traditional INI file format with sections and options, but with a few additional extensions.

Note

Both forward slashes and backslashes are supported. Backslashes are unconditionally copied, as they do not escape characters.

Comments

The configuration file can contain comment lines. Comment lines start with a hash (`#`) or semicolon (`;`) and continue to the end of the line. Trailing comments are *not* supported.

Sections

Each configuration file consists of a list of *configuration sections* where each section contains a sequence of *configuration options*. Each configuration option has a name and value. For example:

```
[section name]
option = value
option = value
option = value

[section name:optional section key]

option = value
option = value
option = value
```

A configuration file section header starts with an opening bracket (`[`) and ends with a closing bracket (`]`). There can be leading and trailing space characters on the line, which are ignored, but no space inside the section brackets.

The section header inside the brackets consists of a *section name* and an optional *section key* that is separated from the section header with a colon (`:`). The combination of section name and section key is unique for a configuration.

The section names and section keys consist of a sequence of one or more letters, digits, or underscores (`_`). No other characters are allowed in the section name or section key.

A section is similar to a namespace. For example, the `user` option's meaning depends on its associated section. A *user* in the *[DEFAULT]* section refers to the system user that MySQL Router is run as, which is also controlled by the `--user` command line option. Unrelated to that is defining *user* in the *[metadata_cache]* section, which refers to the MySQL user that accesses a MySQL server's metadata.

Default Section

The special section name `DEFAULT` (any case) is used for default values for options. Options not found in a section are looked up in the default section. The default section does not accept a section key.

Options

After a section's start header, there can be a sequence of zero or more *option lines* where each option line is of the form:

```
name = value
```

Any leading or trailing blank characters on the option name or option value are removed before being handled. Option names are case-insensitive. Trailing comments are not supported, so in this example the option `mode` is given the value "read-only # Read only mode" and will therefore generate an error when starting the router.

```
[routing:round-robin]
# Trailing comments are not supported so the following is incorrect
routing_strategy=round-robin # Circles back to first server
```

Variable Interpolation

Option values support (*variable interpolation*) using an option name given within braces `{` and `}`. Interpolation is done on retrieval of the option value and not when it is read from the configuration file. If a variable is not defined then no substitutions are done and the option value is read literally.

Consider this sample configuration file:

```
[DEFAULT]
prefix = /usr/

[sample]
bin = {prefix}bin/{name}
lib = {prefix}lib/{name}
name = magic
directory = C:\foo\bar\{3a339172-6898-11e6-8540-9f7b235afb23}
```

Here the value of `bin` is "/usr/bin/magic", the value of `lib` is "/usr/lib/magic", and the value of `directory` is "C:\foo\bar\{3a339172-6898-11e6-8540-9f7b235afb23}" because a variable named "{3a339172-6898-11e6-8540-9f7b235afb23}" is not defined.

Predefined variables

MySQL Router defines predefined variables that are available to the configuration file. Variables use braces, such as `{program}` for the `program` predefined variable.

Table 4.1 Predefined variables

Name	Description
`program`	Name of the program, normally `mysqlrouter`
`origin`	Path to directory where binary is located
`logging_folder`	Path to folder for log files
`plugin_folder`	Path to folder for plugins
`runtime_folder`	Path to folder for runtime data

Name	Description
`config_folder`	Path to folder for configuration files

4.2 Configuration File Locations

MySQL Router scans for the default configuration files at startup, and optionally loads user-defined configuration files at runtime from the command line.

Default Configuration File Locations

By default, MySQL Router scans specific locations for its configuration files that depend on the platform and how MySQL Router was set up.

You can alter the default locations at compile time by using the `-DROUTER_CONFIGDIR=<path>` option. You could also edit `cmake/settings.cmake` to change the default locations before compiling MySQL Router, thus adding new locations or exceptions for specific platforms.

Execute `mysqlrouter --help` to see the default configuration file locations (and their availability) on your system. For example:

```
shell> mysqlrouter --help

...

Start MySQL Router.

Configuration read from the following files in the given order (enclosed
in parentheses means not available for reading):
  (/usr/local/mysql-router/mysqlrouter.conf)
  /Users/philip/.mysqlrouter.conf
Plugins Path:
  /usr/local/lib/mysqlrouter
Default Log Directory:
  /usr/local/mysql-router
Default Persistent Data Directory:
  /usr/local/mysql-router/data
Default Runtime State Directory:
  /usr/local/mysql-router/run

Usage: mysqlrouter [-v|--version] [-h|--help]

...
```

Important

The default configuration file is not loaded if a user-defined configuration file is passed in with the `--config` option.

On Linux, MySQL Router scans the following locations by default, although these locations are system dependent:

1. `/etc/mysqlrouter/mysqlrouter.conf`

Note

Unlike MySQL server, the backward compatible path "`/etc/mysqlrouter.conf`" is not supported.

2. `$HOME/.mysqlrouter.conf`

Note

For backward compatibility, MySQL Router also looks for the *.ini* variant in each directory. In doing so, Router looks in the initial directory for the *.conf* version,

then checks for a *.ini* version, and then repeats the process in the next directory which is typically the user's home directory on the system.

User-Defined and Extra Configuration Files

Two command line options help control these configuration file locations:

- `--config` (or `-c`): Read the base configuration from this file, and not use or scan the default file paths.

  Example use: when generating a standalone MySQL Router installation with the `--directory` bootstrap option, the generated `start.sh` passes this option to the generated `mysqlrouter.conf` inside that directory.

- `--extra-config` (or `-a`): Read this additional configuration file after the configuration files are read from either the default locations, or from files specified using the `--config` option.

For example:

```
shell> mysqlrouter --config /custom/path/to/router.conf --extra-config /another/config.conf
```

Multiple extra configuration options can be passed in and the files are loaded in the order they are entered, with `--config` options being loaded before the `--extra-config` options. For example:

```
shell> mysqlrouter --extra-config a.conf --config b.conf --extra-config c.conf
```

In the above example, `b.conf` is loaded first, and then `a.conf` and `c.conf`, in that order. Also, the default configuration file, such as `/etc/mysqlrouter/mysqlrouter.conf`, is not loaded because `--config` was used.

Each loaded configuration file overrides configuration settings from the previously read configuration files.

Default Configuration File Locations (Linux)

The following lists default file location for the router to read configuration files on popular Linux platforms.

Note

Execute `mysqlrouter --help` to see the default configuration file locations (and their availability) on your system.

- Default system-wide installation under `/usr/local` : `/usr/local/etc/mysqlrouter.conf`
- RPM and Debian : `/etc/mysqlrouter/mysqlrouter.conf`
- On all systems, a bootstrapped standalone installation using `--directory` adds `mysqlrouter.conf` into the directory defined by *--directory*.

Default Configuration File Locations (Windows)

Default file locations that MySQL Router searches for configuration files on Windows.

Note

Execute `mysqlrouter.exe --help` to see the default configuration file locations (and their availability) on your system.

- Default system-wide installation under `C:\ProgramData\MySQL\MySQL Router` : `C:\ProgramData\MySQL\MySQL Router\mysqlrouter.conf`
- In addition: `C:\Users\`*`username`*`\AppData\Roaming\mysqlrouter.conf` where *`username`* is replaced with your system's user.

- In addition to *mysqlrouter.conf*, for backwards compatibility the system also looks for *mysqlrouter.ini*
- With `--directory`: a bootstrapped standalone installation using `--directory` adds `mysqlrouter.conf` into the directory defined by *--directory*.

4.3 Configuration Options

Configuration file options and command-line options serve different purposes and are documented in separate locations.

When `bootstrapping`, the generated configuration file's settings depend on the bootstrap options passed into `mysqlrouter`. For example, passing in `--conf-use-sockets` enables socket connections by defining `socket` for each route in the generated configuration file. Or, `--directory` adds all generated files and subdirectories to a single directory and adjusts the generated configuration file values accordingly.

4.3.1 MySQL Router Command Line Programs

This section describes the MySQL Router commands. The `mysqlrouter` command is used for most tasks, including bootstrapping and running MySQL Router, and `mysqlrouter_plugin_info` is an optional debugging tool.

4.3.1.1 `mysqlrouter` — Command Line Options

- mysqlrouter Option Summaries
- mysqlrouter Option Descriptions

MySQL Router accepts command line options that are passed into `mysqlrouter` to affect its behavior, or to bootstrap router based on an InnoDB cluster.

When starting Router, you can optionally use `--config` to pass in the main configuration file's location (otherwise the default location is used) and `--extra-config` for an additional configuration file.

Bootstrapping command line options affect the generated files and directories that are used when starting MySQL Router.

mysqlrouter Option Summaries

Table 4.2 General Options

Option Name	Description	Introduced
--config	Read configuration options from the provided file	
--extra-config	Read this file after configuration files are read from either default locations or from files specified by the --config option	
--help	Display help text and exit	
--pid-file	Location to store the PID file	8.0.20
--user	Run mysqlrouter as the user having the defined user name or numeric user id	
--version	Display version information and exit	

Table 4.3 Bootstrapping Options

Option Name	Description	Introduced
--account	The MySQL user account used by Router after boostrapping	8.0.19
--account-create	Bootstrapped account creation behavior	8.0.19

Option Name	Description	Introduced
--account-host	The host pattern used for bootstrapped accounts	8.0.12
--bootstrap	Bootstrap and configure Router for operation with a MySQL InnoDB cluster	
--bootstrap-socket	Connect to the MySQL metadata server through a Unix domain socket, used in conjunction with --bootstrap	
--conf-base-port	Base port to use for listening Router ports	
--conf-bind-address	IP address of the interface to which router's listening sockets should bind	
--conf-skip-tcp	Whether to disable binding of a TCP port for incoming connections	
--conf-use-gr-notifications	Enables Group Replication notifications	8.0.17
--conf-use-sockets	Whether to use Unix domain sockets	
--connect-timeout	Number of seconds before connection attempts to a metadata server are considered timed out	8.0.4
--directory	Creates a self-contained directory for a new instance of the Router	
--force	Force reconfiguration of a possibly existing instance of the router	
--force-password-validation	When creating a user account automatically, do not skip the validate_password mechanism	
--master-key-reader	Script that returns the master key to STDOUT	8.0.12
--master-key-writer	Script that reads the master key from STDIN	8.0.12
--name	Gives a symbolic name for the router instance	
--password-retries	The number of retries to use for generating the Router's user password	
--read-timeout	Number of seconds before read operations to a metadata server are considered timed out	8.0.4
--report-host	Router's hostname; overrides auto-detection	8.0.12
--strict	Enables bootstrap strict mode	8.0.19

Table 4.4 SSL Options

Option Name	Description	Introduced
--ssl-ca	Path to SSL Certificate Authority file to verify server's certificate against	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	The client-side SSL certificate to facilitate client-side authentication during bootstrap	8.0.4
--ssl-cipher	A colon-separated list of SSL ciphers to allow, if SSL is enabled	
--ssl-crl	Path to SSL CRL file to use when verifying server certificate	
--ssl-crlpath	Path to directory containing SSL CRL files to use when verifying server certificate	
--ssl-key	The private SSL key to facilitate client-side authentication during bootstrap	8.0.4

Option Name	Description	Introduced
--ssl-mode	Desired security state when connecting to the metadata server during bootstrap and normal operation. Analogous to --ssl-mode in mysql client	
--tls-version	Comma-separated list of TLS versions to request, if SSL is enabled	

Table 4.5 Windows Services Options

Option Name	Description
--clear-all-credentials	Clear all stored credentials
--install-service	Install MySQL Router as service named MySQLRouter, set it to automatically start when Windows restarts (Windows only)
--install-service-manual	Install MySQL Router as service named MySQLRouter that can be manually started (Windows only)
--remove-credentials-section	Remove a section's credentials
--remove-service	Remove MySQL Router as a Windows service
--service	Start MySQL Router as a Windows service
--update-credentials-section	Update a section's credentials

mysqlrouter Option Descriptions

- `--version`, `-V`

Property	Value
Command-Line Format	`--version , -V`

Displays the version number and related information of the application, and exits. For example:

```
shell> mysqlrouter --version

MySQL Router v8.0.21 on Linux (64-bit) (GPL community edition)
```

- `--help`, `-?`

Property	Value
Command-Line Format	`--help , -?`

Display help and informative information, and exit.

The `--help` option has an added benefit. Along with the explanation of each of the options, the `--help` option also displays the paths used to find the configuration file, and also several default paths. The following excerpt of the `--help` output shows an example from a Ubuntu 16.04 machine:

```
shell> mysqlrouter --help

...

Start MySQL Router.

Configuration read from the following files in the given order (enclosed
in parentheses means not available for reading):
  (/etc/mysqlrouter/mysqlrouter.conf)
  /home/philip/.mysqlrouter.conf
Plugin Path:
  /usr/lib/x86_64-linux-gnu/mysqlrouter
Default Log Directory:
  /var/log/mysqlrouter
Default Persistent Data Directory:
```

```
  /var/lib/mysqlrouter
Default Runtime State Directory:
  /run/mysqlrouter

Usage: mysqlrouter [-V|--version] [-?|--help]
...
```

The configuration section shows the order for the paths that may be used for reading the configuration file. In this case, only the second file is accessible.

- `--bootstrap` *`URI`*, `-B` *`URI`*

Property	Value
Command-Line Format	`--bootstrap URI, -B URI`
Type	String

The main option to perform a bootstrap of MySQL Router by connecting to the InnoDB cluster metadata server at the URI provided. MySQL Router configures itself based on the information retrieved from the InnoDB cluster metadata server. A password is prompted for if needed. If a username is not provided as part of the URI then the default user name "root" is used. See Connecting Using URI-Like Connection Strings for information on using a path to specify a server instance.

Note

While `--bootstrap` accepts a URI for TCP/IP connections, using the `--bootstrap-socket` option with a local Unix domain socket name replaces the "host:port" part of the URI passed to the `--bootstrap` option with the socket on the same machine.

By default, the bootstrap process performs a system-wide configuration of MySQL Router. Only one instance of MySQL Router can be configured for system-wide operation. The system instance of MySQL Router has a `router_name` of "system". If additional instances are desired, use the `--directory` option to create self-contained MySQL Router installations.

`URI`: a server instance from an InnoDB cluster to fetch metadata information from. If the provided *`URI`* is a read-only instance, MySQL Router automatically reconnects to a read-write instance in the InnoDB cluster so it can register MySQL Router.

If a configuration file already exists when you start MySQL Router with the `--bootstrap`, the existing `router_id` in that file is reused, and a reconfiguration process occurs. The configuration file is regenerated from scratch and the MySQL Router's metadata server account is recreated, although with the same name.

During the reconfiguration process, all changes made to an existing configuration file are discarded. To customize a configuration file and still retain the ability of automatic reconfiguration (bootstrapping), you can use the `--extra-config` command line option to specify an additional configuration file that is read after the main configuration file. These configuration options are used because this extra configuration file is loaded after the main configuration file.

The bootstrap process creates a new MySQL user account with a randomly generated password to use by that specific MySQL Router instance. This account is used by MySQL Router when connecting to the metadata server and InnoDB cluster to fetch information about its current state. For detailed information about this user including how its password is stored and the MySQL privilege it requires, see documentation for the `MySQL user option`.

The generated configuration file is named `mysqlrouter.conf`, and its location depends on the type of instance being configured, the system, and the package. For system-wide installations, the generated configuration file is added to the system's configuration directory such as `/etc` or

`%PROGRAMDATA%\MySQL\MySQL Router\`. Executing `mysqlrouter --help` will display this location.

The `--user` option is required if executing a bootstrap with a super user (uid=0). Although not recommended, forcing the super user is possible by passing its name as an argument such as *--user=root*.

Note

The minimum GRANT permissions required to execute `--boostrap` are:

```
GRANT CREATE USER ON *.* TO 'bootstrapuser'@'%' WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON mysql_innodb_cluster_metadata.* TO
'bootstrapuser'@'%';
GRANT SELECT ON mysql.user TO 'bootstrapuser'@'%';
GRANT SELECT ON performance_schema.replication_group_members TO
'bootstrapuser'@'%';
GRANT SELECT ON performance_schema.replication_group_member_stats TO
'bootstrapuser'@'%';
```

Using `--bootstrap` adds default values to the generated MySQL Router configuration file, and some of these default values depend on other conditions. Listed below are some of the conditions that affect the generated default values, where default is defined by passing in `--bootstrap` by itself.

Table 4.6 Conditions that affect default --bootstrap values

Condition	Description
`--conf-base-port`	Modifies generated `bind_port` values for each connection type. By default, generated `bind_port` values are as follows: For the classic protocol, Read-Write uses 6446 and Read-Only uses 6447, and for the X protocol Read-Write uses 64460 and Read-Only uses 64470.
`--conf-use-sockets`	Inserts `socket` definitions for each connection type.
`--conf-skip-tcp`	TCP/IP connection definitions are not defined.
`--directory`	Affects all file paths, and also generates additional files.
Other	This list is not exhaustive, other options and conditions also affect the generated values.

- `--bootstrap-socket` *`socket_name`*

Property	Value
Command-Line Format	`--bootstrap-socket socket_name`
Platform Specific	Linux

Used in conjunction with `--bootstrap` to bootstrap using a local Unix domain socket instead of TCP/IP. The `--bootstrap-socket` value replaces the "host:port" part in the `--bootstrap` definition with the assigned socket name for connecting to the MySQL metadata server using Unix domain sockets. This is the MySQL instance that is being bootstrapped from, and this instance must be on the same machine if sockets are used. For additional details about how bootstrapping works, see `--bootstrap`.

This option is different than the `--conf-use-sockets` command line option that sets the `socket` configuration file option during the bootstrap process.

This option is not available on Windows.

- `--directory dir_path`, `-d dir_path`

Property	Value
Command-Line Format	`--directory dir_path, -d dir_path`
Type	String

Specifies that a self-contained MySQL Router installation will be created at the defined directory instead of configuring the system-wide router instance. This also allows multiple router instances to be created on the same system.

The self-contained directory structure for Router is:

```
$path/start.sh
$path/stop.sh
$path/mysqlrouter.pid
$path/mysqlrouter.conf
$path/mysqlrouter.key
$path/run
$path/run/keyring
$path/data
$path/log
$path/log/mysqlrouter.log
```

If this option is specified, the keyring file is stored under the runtime state directory of that instance, under `run/` in the specified directory, as opposed to the system-wide runtime state directory.

If `--conf-use-sockets` is also enabled then the generated socket files are also added to this directory.

- `--master-key-writer`

Property	Value
Command-Line Format	`--master-key-writer file_path`
Introduced	8.0.12
Type	String

This optional bootstrap option accepts a script that reads the master key from *STDIN*. It also uses the *ROUTER_ID* environment variable set by MySQL Router before the `master-key-writer` script is called.

The `master-key-writer` and `master-key-reader` options must be used together, and using them means the `master_key_file` option must not be defined in `mysqlrouter.conf` as the master key is not written to the `mysqlrouter.key` master key file.

This is also written to the generated MySQL Router configuration file as the `master-key-writer` [DEFAULT] option.

Example contents of a bash script named `writer.sh` used in our example:

```

#!/bin/bash

KID_=$(keyctl padd user ${ROUTER_ID} @us <&0)
```

Example usage:

```
shell> mysqlrouter --bootstrap=127.0.0.1:3310 --master-key-reader=./reader.sh --master-key-writer=./write
```

This also affects the generated `mysqlrouter.conf`, for example:

```
[DEFAULT]
...
```

```
master-key-reader=reader.sh
master-key-writer=writer.sh
```

- `--master-key-reader`

Property	Value
Command-Line Format	`--master-key-reader file_path`
Introduced	8.0.12
Type	String

This optional bootstrap option accepts a script that writes the master key to *STDOUT*. It also uses the *ROUTER_ID* environment variable set by MySQL Router before the `master-key-reader` script is called.

The `master-key-reader` and `master-key-writer` options must be used together, and using them means the `master_key_file` option must not be defined in `mysqlrouter.conf` as the master key is not written to the `mysqlrouter.key` master key file, and instead uses the value provided by this option's script.

This is also written to the generated MySQL Router configuration file as the `master-key-reader` [DEFAULT] option.

Example contents of a bash script named `reader.sh` used in our example:

```
#!/bin/bash

KID_=$(keyctl search @us user ${ROUTER_ID} 2>/dev/null)
if [ ! -z $KID_ ]; then
  keyctl pipe $KID_
fi
```

Example usage:

```
shell> mysqlrouter --bootstrap=127.0.0.1:3310 --master-key-reader=./reader.sh --master-key-writer=./w
```

This also affects the generated `mysqlrouter.conf`, for example:

```
[DEFAULT]
...
master-key-reader=reader.sh
master-key-writer=writer.sh
```

- `--strict`

Property	Value
Command-Line Format	`--strict`
Introduced	8.0.19
Type	String

Enables strict mode, which for example causes the bootstrap `--account` user verification check to stop the bootstrap process rather than only emit a warning and continue if the supplied user does not pass the check.

- `--account`

Property	Value
Command-Line Format	`--account username`
Introduced	8.0.19

Property	Value
Type	String

A bootstrap option to specify the MySQL user to use, which either reuses an existing MySQL user account or creates one; behavior controlled by the related `--account-create` option.

With `--account`, usage favors ease of management over ease of deployment as multiple routers may share the same account, and the username and password are manually defined rather than auto-generated.

Setting this option triggers a password prompt for this account regardless of whether the password is available in the keyring.

Bootstrapping without passing in `--account` does not recreate an existing MySQL server account. Prior to MySQL Router 8.0.18, bootstrapping would DROP the existing user and reCREATE it.

Using this option assumes the user has sufficient access rights for Router because the bootstrap process does not attempt to add missing grants to existing accounts. The bootstrap process does verify the permissions and outputs information to the console of the failed check. The bootstrap process continues despite these failed checks unless the optional `--strict` option is also used. Example required permissions:

```
GRANT SELECT ON mysql_innodb_cluster_metadata.* TO `theuser`
GRANT SELECT ON performance_schema.replication_group_members TO `theuser`
GRANT SELECT ON performance_schema.replication_group_member_stats TO `theuser`
```

A password is not accepted from the command-line. For example, passing in "foo:bar" assumes "foo:bar" is the desired username rather than user *foo* with the password *bar*.

- `--account-create`

Property	Value
Command-Line Format	`--account-create behavior`
Introduced	8.0.19
Type	String
Default Value	`if-not-exists`

Valid Values	`if-not-exists`
	`always`
	`never`

Specify the account creation policy to help guard against accidentally bootstrapping with the wrong user account. Potential values are:

- `if-not-exists` (default): Bootstrap either way; reuse the account if it exists, otherwise create it.
- `always`: Only bootstrap if the account does not already exist; and create it.
- `never`: Only bootstrap if the account already exists; and reuse it.

This option requires that the `--account` option is also used, and that `--account-host` is not used.

- `--account-host`

Property	Value
Command-Line Format	`--account-host host_pattern`

Property	Value
Introduced	8.0.12
Type	String
Default Value	`%`

The host pattern used for accounts created by MySQL Router during the bootstrap process. This is optional and defaults to '%'.

Pass in this option multiple times to define multiple patterns, in which case the generated MySQL accounts use the same password.

Note

Router does not perform sanity checking and does not ensure that the pattern authorizes Router to connect.

Note

Bootstrapping reuses existing Router accounts by dropping and recreating the user, and this user recreation process applies to every host.

Examples:

```
# One host
shell> mysqlrouter --bootstrap localhost:3310 --account-host host1

# Or, multiple hosts
shell> mysqlrouter --bootstrap localhost:3310 --account-host host1 --account-host host2 --account-hos
```

- `--conf-use-sockets`

Property	Value
Command-Line Format	`--conf-use-sockets`
Platform Specific	Linux

Enables local Unix domain sockets.

This option is used while bootstrapping, and enabling it adds the `socket` option to the generated configuration file.

The name of the generated socket file depends on the `mode` and `protocol` options. With the classic protocol enabled, the file is named `mysql.sock` in read-write mode, and `mysqlro.sock` in read-only mode. With the X Protocol enabled, the file is named `mysqlx.sock` in read-write mode, and `mysqlxro.sock` in read-only mode.

This option is not available on Windows.

- `--conf-use-gr-notifications`

Property	Value
Command-Line Format	`--conf-use-gr-notifications`
Introduced	8.0.17

Enables the `use_gr_notifications` [metadata_cache] option during bootstrap.

- `--pid-file path`

Property	Value
Command-Line Format	`--pid-file path`
Introduced	8.0.20
Type	String

Sets location of the PID file. This can be set in three different ways (in order of precedence): this `--pid-file` command-line option, setting `pid_file` in Router's configuration file, or defining the `ROUTER_PID` environment variable.

If `--bootstrap` is specified, then setting --pid-file causes Router to fail. This is unlike ROUTER_PID and the pid_file configuration option, which are ignored if --bootstrap is specified.

If `--bootstrap` is not specified, then the following cause Router to fail: the --pid-file already exists, pid_file or ROUTER_PID are set but empty, or if Router can't write the PID file.

- `--report-host`

Property	Value
Command-Line Format	`--report-host hostname`
Introduced	8.0.12
Type	String

Optionally define Router's hostname instead of relying on auto-detection to determine the externally visible hostname registered to metadata during the bootstrap process.

Router does not check or confirm that the supplied hostname is reachable, and it only checks the hostname string for illegal characters. Only alphanumeric, '-', '.', and '_' characters are allowed.

The supplied hostname is written to the host_name field of the mysql_innodb_cluster_metadata.hosts table in the MySQL InnoDB cluster metadata store.

- `--conf-skip-tcp`

Property	Value
Command-Line Format	`--conf-skip-tcp`
Platform Specific	Linux

Skips configuration of a TCP port for listening to incoming connections. See also `--conf-use-sockets`.

This option is not available on Windows.

- `--conf-base-port port_num`

Property	Value
Command-Line Format	`--conf-base-port port_num`
Type	Integer

Base (first) value used for the listening TCP ports by setting `bind_port` for each bootstrapped route.

This value is used for the classic read-write route, and each additional allocated port is incremented by a value of one. The port order set is classic read-write / read-only, and then x read-write / read-only.

Example usage:

```
# Example without --conf-base-port
shell> mysqlrouter --bootstrap root@localhost:3310
...
Classic MySQL protocol connections to cluster 'devCluster':
- Read/Write Connections: localhost:6446
- Read/Only Connections: localhost:6447

X protocol connections to cluster 'devCluster':
- Read/Write Connections: localhost:64460
- Read/Only Connections: localhost:64470

# Example demonstrating --conf-base-port behavior
shell> mysqlrouter --bootstrap root@localhost:3310 --conf-base-port 6446
...
Classic MySQL protocol connections to cluster 'devCluster':
- Read/Write Connections: localhost:6446
- Read/Only Connections: localhost:6447

X protocol connections to cluster 'devCluster':
- Read/Write Connections: localhost:6448
- Read/Only Connections: localhost:6449
```

- `--conf-bind-address address`

Property	Value
Command-Line Format	`--conf-bind-address address`
Type	String
Default Value	`0.0.0.0`

Modifies the `bind_address` value set by `--bootstrap` in the generated Router configuration file. By default, bootstrapping sets `bind_address=0.0.0.0` for each route, and this option changes that value.

Note

The default `bind_address` value is *127.0.0.1* if `bind_address` is not defined.

- `--read-timeout num_seconds`

Property	Value
Command-Line Format	`--read-timeout num_seconds`
Introduced	8.0.4
Type	Integer
Default Value	`30`

Number of seconds before read operations to a metadata server are considered timed out.

This affects read operations during both the bootstrap process, and also affects normal MySQL Router operations by setting the associated `read_timeout` option in the generated `mysqlrouter.conf`.

This option is set under the `[DEFAULT]` namespace.

- `--connect-timeout num_seconds`

Property	Value
Command-Line Format	`--connect-timeout num_seconds`

Property	Value
Introduced	8.0.4
Type	Integer
Default Value	`30`

Number of seconds before connection attempts to a metadata server are considered timed out.

This affects connections during both the bootstrap process, and also affects normal MySQL Router operations by setting the associated `connect_timeout` option in the generated `mysqlrouter.conf`.

There are two *connect_timeout* variants. The metadata server variant is defined under the `[DEFAULT]` namespace, while the MySQL server variant is defined under the `[routing]` namespace.

- `--user {`*`user_name`*`|`*`user_id`*`}`, `-u {`*`user_name`*`|`*`user_id`*`}`

Property	Value
Command-Line Format	`--user {user_name|user_id}, -u {user_name|user_id}`
Platform Specific	Linux
Type	String

Run `mysqlrouter` as the user having the name *`user_name`* or the numeric user ID *`user_id`*. “User” in this context refers to a system login account, not a MySQL user listed in the grant tables. When bootstrapping, all generated files are owned by this user, and this also sets the associated `user` option.

This system `user` is defined in the configuration file under the `[DEFAULT]` namespace. For additional information, see the `user` option's documentation that `--user` configures.

The `--user` option is required if executing a bootstrap as a super user (uid=0). Although not recommended, forcing the super user is possible by passing its name as an argument, such as *--user=root*.

This option is not available on Windows.

- `--name `*`router_name`*

Property	Value
Command-Line Format	`--name router_name`
Type	String
Default Value	`system`

On initial bootstrap, specifies a symbolic name for a self-contained Router instance. This option is optional, and is used with `--directory`. When creating multiple instances, the names must be unique.

- `--force-password-validation`

Property	Value
Command-Line Format	`--force-password-validation`
Platform Specific	Linux

By default, MySQL Router skips the MySQL Server's validate_password mechanism and instead Router generates and uses a STRONG password based on known validate_password default

settings. This is because validate_password can be configured by the user and Router can not take into account unusual custom settings.

This option ensures that password validation (validate_password) is not skipped for generated passwords, and it is disabled by default.

- `--password-retries num_retries`

Property	Value
Command-Line Format	`--password-retries num_retries`
Type	Integer
Default Value	`20`
Minimum Value	`1`
Maximum Value	`10000`

Specifies the number of times MySQL Router should attempt to generate a password when creating user account with the password validation rules. The default value is 20. The valid range is 1 to 10000.

The most likely reason for failure is due to custom validate_password settings with unusual requirements such as a 50 character minimum. In that fail scenario, either `--force-password-validation` is set to true and/or the `mysql_native_password` MySQL Server plugin is disabled (this plugin allows bypassing validation).

- `--force`

Property	Value
Command-Line Format	`--force`

Force a reconfiguration over a previously configured router instance on the host.

- `--ssl-mode mode`

Property	Value
Command-Line Format	`--ssl-mode mode`
Type	String
Default Value	`PREFERRED`

Valid Values	`PREFERRED`
	`DISABLED`
	`REQUIRED`
	`VERIFY_CA`
	`VERIFY_IDENTITY`

SSL connection mode for use during bootstrap and normal operation when connecting to the metadata server. Analogous to `--ssl-mode` in the `mysql` client.

During bootstrap, all connections to metadata servers made by the Router will use the SSL options specified. If `ssl_mode` is not specified in the configuration, it will default to PREFERRED. During

normal operation, after Router is launched, its Metadata Cache plugin will read and honor all configured SSL settings.

When set to a value other than the default (PREFERRED), an `ssl_mode` entry is inserted under the `[metadata_cache]` section in the generated configuration file.

Available values are DISABLED, PREFERRED, REQUIRED, VERIFY_CA, and VERIFY_IDENTITY. PREFERRED is the default value. As with the `mysql` client, this value is case-insensitive.

The configuration file equivalent is documented separately at `ssl_mode`.

- `--ssl-cert file_path`

Property	Value
Command-Line Format	`--ssl-key file_path`
Introduced	8.0.4
Type	String

The path name of the SSL public key certificate file in PEM format. This is used to facilitate client-side authentication during the bootstrap process. This directly matches and uses functionality of the MySQL client's `--ssl-cert` option.

Like `--ssl-key`, this option is only used during bootstrap that uses a root account. It is useful when the root account was created with REQUIRE X509, and therefore logging in as root requires the client to authenticate itself.

- `--ssl-key file_path`

Property	Value
Command-Line Format	`--ssl-key file_path`
Introduced	8.0.4
Type	String

The path name of the SSL private key file in PEM format. This is used to facilitate client-side authentication during the bootstrap process. This directly matches and uses functionality of the MySQL client's `--ssl-key` option.

Like `--ssl-cert`, this option is only used during a bootstrap process that uses a root account. It is useful when the root account was created with REQUIRE X509, and therefore logging in as root requires the client to authenticate itself.

- `--ssl-cipher ciphers`

Property	Value
Command-Line Format	`--ssl-cipher ciphers`
Type	String
Default Value	

A colon-separated (":") list of SSL ciphers to allow, if SSL is enabled.

- `--tls-version versions`

Property	Value
Command-Line Format	`--tls-version versions`
Type	String

Property	Value
Default Value	

A comma-separated (",") list of TLS versions to request, if SSL is enabled.

- `--ssl-ca file_path`

Property	Value
Command-Line Format	`--ssl-ca file_path`
Type	String
Default Value	

Path to the SSL CA file to verify a server's certificate against.

- `--ssl-capath dir_path`

Property	Value
Command-Line Format	`--ssl-capath dir_path`
Type	String
Default Value	

Path to directory containing the SSL CA files to verify a server's certificate against.

- `--ssl-crl file_path`

Property	Value
Command-Line Format	`--ssl-crl file_path`
Type	String
Default Value	

Path to the SSL CRL file to use when verifying a server's certificate.

- `--ssl-crlpath dir_path`

Property	Value
Command-Line Format	`--ssl-crlpath dir_path`
Type	String
Default Value	

Path to the directory containing SSL CRL files to use when verifying a server's certificate.

- `--config file_path`, `-c file_path`

Property	Value
Command-Line Format	`--config file_path, -c file_path`

Used to provide a path and file name for the configuration file to use. Use this option if you want to use a configuration file located in a folder other than the default locations.

When used with `--bootstrap`, and if the configuration file already exists, a copy of the current file is saved with a *.bak* extension if the generated configuration file contents is different than the original. Existing *.bak* files are overwritten.

- `--extra-config file_path`, `-a file_path`

Property	Value
Command-Line Format	`--extra-config file_path, -a file_path`

Used to provide an optional, additional configuration file to use. Use this option if you want to split the configuration file into two parts for testing, multiple instances of the application running on the same machine, etc.

This configuration file is read after the main configuration file. If there are conflicts (an option is set in multiple configuration files), values from the file that is loaded last is used.

- `--install-service`

Property	Value
Command-Line Format	`--install-service`
Platform Specific	Windows

Install Router as a Windows service that automatically starts when Windows starts. The service name is *MySQLRouter*.

This installation process does not validate configuration files passed in via `--config`.

This option is only available on Windows.

- `--install-service-manual`

Property	Value
Command-Line Format	`--install-service-manual`
Platform Specific	Windows

Install MySQL Router as a Windows service that can be manually started. The service name is *MySQLRouter*.

This option is only available on Windows.

- `--remove-service`

Property	Value
Command-Line Format	`--remove-service`
Platform Specific	Windows

Remove the Router Windows service.

This option is only available on Windows.

- `--service`

Property	Value
Command-Line Format	`--service`
Platform Specific	Windows

Start Router as a Windows service.

This option is only available on Windows.

- `--update-credentials-section`

Property	Value
Command-Line Format	`--update-credentials-section section_name`
Platform Specific	Windows

This option is only available on Windows, and refers to its password vault.

- `--remove-credentials-section` *`section_name`*

Property	Value
Command-Line Format	`--remove-credentials-section section_name`
Platform Specific	Windows

Remove the credentials for a given section.

This option is only available on Windows, and refers to its password vault.

- `--clear-all-credentials`

Property	Value
Command-Line Format	`--clear-all-credentials`
Platform Specific	Windows

Clear the password vault by removing all credentials stored in it.

This option is only available on Windows, and refers to its password vault.

4.3.1.2 `mysqlrouter_plugin_info` — Command Line Options

The `mysqlrouter_plugin_info` utility is a debugging tool that inspects a MySQL Router plugin for potential conflicts and general problems.

Usage information:

```
shell> ./mysqlrouter_plugin_info --help

Usage:
  ./mysqlrouter_plugin_info <mysqlrouter_plugin_file> <mysql_plugin_name>

Example:
  ./mysqlrouter_plugin_info /usr/lib/mysqlrouter/routing.so routing

To print help information:
  ./mysqlrouter_plugin_info --help
To print application version:
  ./mysqlrouter_plugin_info --version

shell> ./bin/mysqlrouter_plugin_info --version

MySQLRouter Plugin Info App 8.0.3
```

Example usage:

```
shell> ./bin/mysqlrouter_plugin_info lib/mysqlrouter/routing.so routing
{
    "abi-version": "2.0",
    "arch-descriptor": "i386/darwin//",
    "brief": "Routing MySQL connections between MySQL clients/connectors and servers",
    "plugin-version": "0.0.1",
    "requires": [],
    "conflicts": []
```

```
}
```

4.3.1.3 `mysqlrouter_passwd` — Command Line Options

The `mysqlrouter_passwd` utility is a a command line application to manage the accounts in the passwd file.

Note

This feature was added in MySQL Router 8.0.16.

Usage information:

```
Usage: ./mysqlrouter_passwd [--delete] [-?|--help] [--kdf=<name>] [--list] [--verify]
                            [-V|--version] [--work-factor=<num>] [filename username]
Options:

  --delete
      Delete username if it exists.
  -?, --help
      Display this help and exit.
  --kdf <name>
      Key Derivation Function. One of pbkdf2-sha256, pbkdf2-sha512, sha256-crypt, sha512-crypt. default: sh
  --list
      List account(s).
  --verify
      Verify password against stored hash for username. Exit-code is 0 if password matches, 1 otherwise
  -V, --version
      Display version information and exit.
  --work-factor <num>
      Work-factor hint for KDF if account is updated.
```

4.3.1.4 `mysqlrouter_keyring` — Command Line Options

The `mysqlrouter_keyring` utility is a a command line application to manage MySQL Router key rings.

Note

This feature was added in MySQL Router 8.0.18.

Usage information:

Generic commands

- `--help`: usage information.
- `--version`: the tool's version.

Keyring commands; all commands also accept --master-key-reader and --master-key-writer instead of --master-key-file.

- `--init`: Initialize keyring with a master-key-file.

  Creates a keyring and master-key-file if they doo not exist; and adds keyring to master-key-file if it does not yet exist there.

- `--list`: List usernames stored in the keyring; or list properties of a user stored in the keyring.
- `--get`: Get property of user from the keyring.
- `--export`: Export keyring as JSON.
- `--set`: Set property in the keyring.

- `-delete-`: Delete user from the keyring.

Master-key commands

- `--master-key-list`: List keyring-ids from master-key-file.
- `--master-key-delete`: Delete master-ley from "keyring" from master-key-file.
- `--master-key-rename`: Rename keyring-id in a master-key-file.

Examples:

```
shell> mysqlrouter_keyring init --master-key-file=mysqlrouter.key data/keyring

shell> mysqlrouter_keyring list --master-key-file=mysqlrouter.key data/keyring
shell> mysqlrouter_keyring list --master-key-file=mysqlrouter.key data/keyring user

shell> mysqlrouter_keyring get --master-key-file=mysqlrouter.key data/keyring someuser key

shell> mysqlrouter_keyring export --master-key-file=mysqlrouter.key data/keyring

shell> mysqlrouter_keyring set --master-key-file=mysqlrouter.key data/keyring user key value

shell> mysqlrouter_keyring delete --master-key-file=mysqlrouter.key data/keyring user
shell> mysqlrouter_keyring delete --master-key-file=mysqlrouter.key data/keyring user key

shell> mysqlrouter_keyring master-key-list --master-key-file=mysqlrouter.key

shell> mysqlrouter_keyring master-key-delete --master-key-file=mysqlrouter.key data/keyring

shell> mysqlrouter_keyring master-key-rename --master-key-file=mysqlrouter.key data/keyring other/data/
```

4.3.2 Configuration File Options

When started, MySQL Router reads a list of *configuration files* that together make up the configuration of the router. At least one configuration file is required.

MySQL Router reads options from configuration files that closely resemble the traditional INI file format, with sections and options. These specify the options set when MySQL Router starts. For file syntax information, see Section 4.1, “Configuration File Syntax”.

Options are defined under sections, that dictate the option's meaning. For example, `user` under the *[DEFAULT]* section refers to the system user running router, while `user` under the *[metadata_cache]* section refers to the MySQL user that accesses metadata.

The following tables are separated by section, and summarize the MySQL Router options defined in a MySQL Router configuration file. Detailed information about each of these options, such as descriptions and allowed values, is documented below these tables.

General Options

Table 4.7 [DEFAULT]

Option Name	Description	Type
`config_folder`	Path to configuration files	String
`connect_timeout`	Number of seconds before connection attempts to a metadata server are considered timed out	Integer
`keyring_path`	Path to keyring file	String
`logging_folder`	Path to router logs	String
`master_key_path`	Path to master keyring file	String
`master-key-reader`	Script that returns the master key to STDOUT	String

Option Name	Description	Type
master-key-writer	Script that reads the master key from STDIN	String
pid_file	Location to store the PID file	String
plugin_folder	Path to router plugins	String
runtime_folder	Path to runtime files	String
sinks	Logging method(s) to receive configured log data	String
thread_stack_size	Size in KB of memory allocated to each thread stack	Integer
user	System user that router is run as	String

Routing Options

Table 4.8 [routing]

Option Name	Description	Type
bind_address	Address router is bound to, also uses bind_port if a port is not defined	String
bind_port	Default port used by bind_address	Integer
client_connect_timeout	Maximum number of seconds to receive packets from MySQL server	Integer
connect_timeout	Number of seconds before connection attempts to a MySQL server are considered timed out	Integer
destinations	Routing destinations as either a comma-separated list of MySQL servers, or a metadata-cache definition	String
dynamic_state	Path to generated JSON file used to track and store active MySQL InnoDB cluster Metadata server addresses	String
max_connect_errors	Maximum number of failed MySQL server connections before giving up	Integer
max_connections	Maximum number of connections assigned to a routed destination MySQL server	Integer
mode	Routing mode, how router chooses destination MySQL servers	String
protocol	Protocol for connecting to MySQL Server	String
read_timeout	Number of seconds before read operations to a metadata server are considered timed out	Integer
routing_strategy	Routing strategy (optional), how router chooses destination MySQL servers	String
socket	Path to Unix domain socket file	String

Metadata Cache Options

Table 4.9 [metadata_cache]

Option Name	Description	Type
auth_cache_refresh_interval	Time between auth-cache refresh attempts	Numeric
auth_cache_ttl	Time until the cache becomes invalid if not refreshed	Numeric
bootstrap_server_addresses	MySQL servers with metadata, as a comma-separated list	String
cluster_type	Object Router was bootstrapped against	String
metadata_cluster	InnoDB cluster name	String
router_id	Router ID	Integer
ssl_mode	SSL connection mode for connecting to the metadata server, defaults to PREFERRED if not set	String

Option Name	Description	Type
ttl	Time To Live, in seconds	Integer
use_gr_notifications	Group Replication notifications behavior	Integer
user	MySQL user that accesses the MySQL Server's metadata schema	String

Logging Options

Table 4.10 [logger]

Option Name	Description	Type
level	Logging level	String
timestamp_precision	Logger timestamp precision	String

HTTP Server Options

Table 4.11 [http_server]

Option Name	Description	Type
bind_address	IP address bound to the HTTP port	String
port	HTTP server TCP port	Integer
require_realm	[http_auth_realm] name	String
ssl_cert	SSL certification file name	String
ssl_cipher	Approved SSL ciphers	String
ssl_dh_param	DH parameter file name	String
ssl	Enables TLSv1.2 or later support	Integer
ssl_key	SSL key filename	String
static_folder	Directory for HTTP server static file requests	String

Table 4.12 [http_auth_realm]

Option Name	Description	Type
backend	Name of the [http_auth_backend] section	String
method	The HTTP authentication method	String
name	Realm name for authenticated user	String
require	Require authentication validation	String

Table 4.13 [http_auth_backend]

Option Name	Description	Type
backend	Backend type	String
filename	Backend storage file name	String

MySQL Router Configuration File Option Descriptions

- logging_folder

Property	Value
Type	String
Default Value	$router_basepath

Path to the MySQL Router log file directory. The log file is named mysqlrouter.log, and it is either generated or appended to if this file already exists.

Setting `logging_folder` to an empty value sends the messages to the console (**stdout**).

Note

The default `logging_folder` value changed from "" to Router's base path in MySQL Router 2.1.

An example that sends logs to `/var/log/mysqlrouter/mysqlrouter.log`:

```
[DEFAULT]
logging_folder = /var/log/mysqlrouter
```

When the `--directory` bootstrap option is used, the generated configuration file sets it to $directory/log/.

- `plugin_folder`

Property	Value
Type	String
Default Value (Other)	`/usr/local/lib/mysqlrouter`
Default Value (Windows)	

Path to the MySQL Router plugins. This folder must match the MySQL Router installation directory. You should only set this if you have a custom installation where the plugins are not in the standard installation location.

Default value: `/usr/local/lib/mysqlrouter`

- `runtime_folder`

Property	Value
Type	String
Default Value (Other)	`/run/mysqlrouter`
Default Value (Windows)	

Path to the MySQL Router runtime files.

Default value: `/run/mysqlrouter`

- `master-key-writer`

Property	Value
Command-Line Format	`--master-key-writer file_path`
Introduced	8.0.12
Type	String

Script that reads the master key from STDIN. Set using the `--master-key-writer` command-line bootstrap option.

- `master-key-reader`

Property	Value
Command-Line Format	`--master-key-reader file_path`
Introduced	8.0.12
Type	String

Script that returns the master key to STDOUT. Set using the `--master-key-reader` command-line bootstrap option.

- `config_folder`

Property	Value
Type	String
Default Value (Other)	`/usr/local/etc/mysqlrouter`
Default Value (Windows)	

Path to the MySQL Router configuration files.

Note

The `config_folder` is currently set at compile time. The option could be used by future plugins when they have their own configuration files.

Default value: `/usr/local/etc/mysqlrouter`

- `sinks`

Property	Value
Introduced	8.0.16
Type	String

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

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

The sink(s) (different logging methods) that a defined log level are sent to.

Supported sink values are: `consolelog`, `filelog`, `eventlog` (on Windows), and `syslog` (on Unix-based systems). Use a comma-separated list to define multiple values.

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

For example, to configure logger to use the file, console and the event log each using the debug log level configured in the *[logger]* section:

```
[logger]
level=debug
sinks=consolelog,eventlog,filelog
```

- `keyring_path`

Property	Value
Type	String
Default Value (Other)	`/run/mysql-router/keyring-data`

Property	Value
Default Value (Windows)	`%PROGRAMDATA%\MySQL\MySQL Router\keyring-data`

Points to the keyring file's location.

A system-wide bootstrap does not add this option to the generated configuration file, and assumes the keyring file is located in the system-wide runtime state directory. If `--directory` is also used, then the keyring file is stored under the runtime state directory of that instance, under `run/` in the specified directory.

System-wide default paths are used if this option is not defined.

Example usage:

```
keyring_path = /opt/myrouter/data/keyring
master_key_path = /opt/myrouter/mysqlrouter.key
```

- `master_key_path`

Property	Value
Type	String
Default Value (Other)	`/run/mysql-router/mysqlrouter.key`
Default Value (Windows)	`%PROGRAMDATA%\MySQL\MySQL Router\mysqlrouter.key`

The master key file's location. This option allows unattended decryption, as otherwise its location is requested at startup.

System-wide default paths are used if this option is not specified.

Example usage:

```
keyring_path = /opt/myrouter/data/keyring
master_key_path = /opt/myrouter/mysqlrouter.key
```

- `user (system)`

Property	Value
Type	String

Run `mysqlrouter` as the user having the name *`user_name`* or the numeric user ID *`user_id`*. “User” in this context refers to a system login account, not a MySQL user listed in the grant tables. This can also be assigned at runtime using the `--user` command line option.

On Linux, installing Router with official DEB or RPM packages creates a local system user and group named "mysqlrouter" on the host, and MySQL Router runs as this user by default. This account does not have shell access and its home directory points to the directory where the default configuration file is stored.

The purpose of this option is to run MySQL Router as a user with restricted system privileges. If the user does not exist on the system, or if an attempt to start Router as root is made, an error is emitted and Router exits.

MySQL Router can be bootstrapped and executed under any Operating System user and does not require special privileges other than read and write access to its own files. The files it accesses

include plugins (read/execute), configuration file, logs, UNIX domain socket files (if enabled), and more.

By default, the configuration and log files are written to a system-wide location such as `/etc` and `/var/log`. Alternatively, Router can be bootstrapped to a self-contained directory of its own by using the `--directory` option. For example:

```
shell> sudo mysqlrouter --bootstrap localhost:3310 --directory /a/path/myrouter --user snoopy
```

In this example, Router creates */a/path/myrouter* and adds all of the generated files and directories here, and these are only writable by the system user `snoopy`. Additionally, `user` is defined in the generated configuration file `/a/path/myrouter/mysqlrouter.conf`:

```
[DEFAULT]
user=snoopy
```

Note

This is different from the `user` definition defined in the `[metadata_cache]` section, which is a MySQL user.

- `bind_address`

Property	Value
Type	String
Default Value	`127.0.0.1`

Information related to the optional `bind_address` option:

- Routing entries can be bound to a network interface (NIC). The default `bind_address` is *127.0.0.1*. If a port is not defined here, then setting `bind_port` is required.
- By default, `--bootstrap` sets `bind_address=0.0.0.0` for each route in the generated Router configuration file. This value can be changed using `--conf-bind-address`.
- Binding to a specific IPv4 or IPv6 address allows and ensures that MySQL Router is not starting and routing the service on an NIC on which nothing is allowed to execute.
- It is not possible to specify more than one binding address per routing configuration group. However, using 0.0.0.0:*$port* (where you define $port) binds all network interfaces (IPs) on the host. IPv6 addresses can also be used.

Example usage:

```
bind_address = 127.0.0.1:7001
```

Note

The `bind_address` cannot be listed in the `destinations` list.

- `bind_port`

Property	Value
Type	Integer

Optionally, you can define a default port for `bind_address` using `bind_port`. If a port is not configured in `bind_address`, then `bind_port` is required and used.

The three examples below all result in *bind_address = 127.0.0.1:7001*

```
[routing:example_1]
bind_port = 7001
```

```
[routing:example_2]
bind_port = 7001
bind_address = 127.0.0.1
```

```
[routing:example_3]
bind_address = 127.0.0.1:7001
```

- `socket`

Property	Value
Platform Specific	Linux
Type	String

Sockets are enabled using the `socket` option, which can be specified with or without the TCP `bind_port` and `bind_address` options. An example:

```
[routing]
socket = /tmp/mysqlrouter.sock
destinations = a.example.com:3306,b.example.com:3307
```

When launching MySQL Router, Router will refuse to run if either the socket file already exists or it cannot be written to.

Relative paths are acceptable and based on the current working directory where Router is launched.

Router can listen to both TCP sockets and Unix sockets simultaneously. For example, the following *[routing]* configuration example is valid and configures Router to listen for connections on both **localhost:1234** and `/tmp/mysqlrouter.sock`:

```
[routing:my_redirect]
bind_address = localhost:1234
socket = /tmp/mysqlrouter.sock
mode = read-write
destinations = localhost:57121, localhost:57122, localhost:57123
```

Note

A Unix domain socket length limit is platform-specific and should not exceed the system's allowed length.

- `protocol`

Property	Value
Type	String
Default Value	`classic`

Valid Values	`classic`
	
	`x`

Used by the routing plugin when connecting to the destination MySQL server, and can be set to either "classic" (default), or "x" (X Protocol).

Example usage:

```
[routing:basic_failover]
bind_port = 7001
mode = read-write
destinations = 10.20.200.1:33060, 10.20.200.2:33060
protocol = x
```

The `protocol` option also affects the default port used by by each destination. If a destination port is not configured, then the default port is 3306 for "classic" (default), 33060 for "x" (X Protocol).

- `pid_file`

Property	Value
Introduced	8.0.20
Type	String

Sets location of the PID file. This can be set in three different ways (in order of precedence): the `--pid-file` command-line option, setting this `pid_file` option in Router's configuration file, or defining the `ROUTER_PID` environment variable.

If `--bootstrap` is specified, then the `pid_file` and ROUTER_PID definitions are ignored. This is unlike the `--pid-file` command-line option which causes Router to fail.

If `--bootstrap` is not specified, then the following cause Router to fail: the --pid-file already exists, pid_file or ROUTER_PID are set but empty, or if Router can't write the PID file.

- `connect_timeout`

Property	Value
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`65536`

Timeout value used by the MySQL Router when connecting to the destination MySQL server. The default value is 1 second. The value cannot be unlimited, and an invalid value results in a configuration error. The valid range is between 1 and 65536. You should keep this value low.

For example, when using `read-write` mode, the value can be a little higher to wait for the PRIMARY to become available. When using `read-only` mode for secondary connections, a lower value makes more sense because Router selects a new server during connection routing.

Example usage:

```
[routing]
connect_timeout = 1
```

- `connect_timeout`

Property	Value
Type	Integer
Default Value (>= 8.0.14)	`15`
Default Value (<= 8.0.13)	`30`

Timeout value used by the MySQL Router when connecting to the MySQL metadata server. The default value is 30 seconds.

Example usage:

```
[DEFAULT]
connect_timeout = 30
```

- `read_timeout`

Property	Value
Type	Integer
Default Value	`30`

Timeout value used by the MySQL Router when reading from the MySQL metadata server. The default value is 30 seconds.

Example usage:

```
[DEFAULT]
read_timeout = 30
```

- `destinations`

Property	Value
Type	String

Provides host information for establishing connections. It accepts either a comma-separated list of destination addresses or a metadata-cache link to an InnoDB cluster.

Example usage with specific hosts (static routing):

```
destinations = a.example.com,b.example.com,c.example.com
```

Note

If a destination's port is not explicitly set, then the default port is 3306 if `protocol` is set to "classic" or not set (default), or port 33060 if `protocol` is set to "x".

Example usage with InnoDB cluster metadata cache:

```
destinations=metadata-cache://mycluster/default?role=PRIMARY
```

The `metadata-cache` URI options are:

- `role`: Determines the type of instances available to the connection. Acceptable values are PRIMARY, SECONDARY, or PRIMARY_AND_SECONDARY.

  The `routing_strategy` `mysqlrouter.conf` option defines the specific strategy, and the default metadata-cache routing strategy is *round-robin*.

- `disconnect_on_promoted_to_primary`: Controls whether existing client connections to a secondary are closed when the secondary is promoted as a primary. The default value is "no",

meaning existing client connections to the promoted secondary are not closed after promotion. Set *disconnect_on_promoted_to_primary=yes* in the URI to close these existing connections.

This option was added in MySQL Router 8.0.12.

- `disconnect_on_metadata_unavailable`: Controls whether existing client connections are closed when the group is overloaded. The default value is "no", meaning existing client connections are not closed when the group is overloaded. Set *disconnect_on_metadata_unavailable=yes* in the URI to close these existing connections.

  This option was added in MySQL Router 8.0.12.

Note

Related, these conditions cause disconnections: connections to a primary after the primary is downgraded to a secondary, and connections to a node that are no longer part of the cluster.

- `dynamic_state`

Property	Value
Introduced	8.0.14
Type	String

This option tracks and stores active MySQL InnoDB cluster Metadata server addresses and loads them if Router is restarted. This functionality is activated by `--bootstrap` and is preferred over the deprecated static `bootstrap_server_addresses` option.

Bootstrapping defines the `dynamic_state` option in `mysqlrouter.conf` file under the [DEFAULT] section. The value is a path to a JSON file named `state.json`, which is created when Router has been bootstrapped. The `state.json` is initialized with InnoDB cluster Metadata server addresses and the Group Replication ID (the group_replication_name returned by the InnoDB cluster); additional information is added and updated while Router is running.

Example `mysqlrouter.conf` entry:

```
[DEFAULT]
dynamic_state=/opt/myrouter/data/state.json
```

Example `state.json` generated by `--bootstrap`:

```
{
    "metadata-cache": {
        "group-replication-id": "4b9e817a-0254-11e9-9cc0-080027bb5030",
        "cluster-metadata-servers": [
            "mysql://localhost:3310",
            "mysql://localhost:3320",
            "mysql://localhost:3330"
        ]
    },
    "version": "1.0.0"
}
```

The `dynamic_state` and deprecated `bootstrap_server_addresses` options cannot be set at the same time. For backwards compatibility, if only `bootstrap_server_addresses` is set then it functions as it did in previous Router versions and this dynamic configuration functionality is not used.

This option was added in MySQL Router 8.0.14.

- `mode`

Property	Value
Type	String

Valid Values	`read-write`
	
	`read-only`

The deprecated mode option sets Router's scheduling, and the two supported mode values are:

Important

MySQL Router 8.0.4 introduced the `routing_strategy` option as a more flexible way to configure the **mode schedule**.

Both `mode` and `routing_strategy` cannot be set at the same time. Setting one is required for static routing while they are optional with InnoDB cluster.

- **read-write**: Typically used for routing to a master or primary MySQL instance.

  Mode Schedule: In *read-write* mode, all traffic is directed to the initial address on the list. If that fails, then MySQL Router will try the next entry on the list continues trying each MySQL server on the list. If no more MySQL servers are available on the list then routing is aborted.

  Note

  With `routing_strategy`, this same behavior can be defined using `routing_strategy=next-available` instead of `mode=read-write`.

  The first successful MySQL server contacted is saved in memory as the first to try for future incoming connections. This is a temporary state, meaning this is forgotten after MySQL Router is restarted.

```
[routing:example_strategy_mode]
bind_port = 7001
destinations = primary1.example.com,primary2.example.com,primary3.example.com
mode = read-write
```

  Because `mode` is deprecated, the previous example should use `routing_strategy` instead:

```
[routing:example_strategy]
bind_port = 7001
destinations = primary1.example.com,primary2.example.com,primary3.example.com
```

```
routing_strategy = next-available
```

- **read-only**: Typically used for routing to a slave or secondary MySQL instance.

  Mode Schedule: Mode *read-only* uses a simple **round-robin** method to go through the list of MySQL Servers. It sends the first connection to the first address on the list, the next connection to the second address, and so on, and circles back to the first address after the list is exhausted.

  Note

  With `routing_strategy`, this same behavior can be defined using `routing_strategy=round-robin` instead of `mode=read-only`.

  If a MySQL server is not available then the next server is tried. When none of the MySQL servers on the list are available then the routing is aborted.

  Unavailable MySQL servers are quarantined. Their availability is rechecked and when available they are put back onto the available `destinations` list. The destinations order is maintained.

  ```
  [routing:ro_route_mode]
  bind_port = 7002
  destinations = secondary1.example.com,secondary2.example.com,secondary3.example.com
  mode = read-only
  ```

  Because `mode` is deprecated, the previous example should use `routing_strategy` instead:

  ```
  [routing:ro_route]
  bind_port = 7002
  destinations = secondary1.example.com,secondary2.example.com,secondary3.example.com
  routing_strategy=round-robin
  ```

Alternatively, the previous `destinations` example could use *metadata-cache* to utilize InnoDB cluster's metadata cache that dynamically configures host information. For example: .

```
[routing:ro_route]
bind_port = 7002
destinations=metadata-cache://myCluster/default?role=SECONDARY
routing_strategy=round-robin
```

- `routing_strategy`

Property	Value
Introduced	8.0.4
Type	String

Valid Values	`first-available`
	
	`next-available`
	
	`round-robin`

Property	Value
	`round-robin-with-fallback`

The routing strategy defines how MySQL Router chooses MySQL servers to connect to.

Important

MySQL Router 8.0.4 introduced the `routing_strategy` option as a more flexible way to define the strategy. Previously this behavior was defined using the now deprecated `mode` option.

Both `routing_strategy` and `mode` cannot be set at the same time. Setting one is required for static routing while they are optional with InnoDB cluster.

Available strategies:

- `round-robin`: for load-balancing, each new connection is made to the next available server in a round-robin fashion.
- `round-robin-with-fallback`: for load-balancing, each new connection is made to the next available secondary server in a round-robin fashion. If a secondary server is not available then servers from the primary list are used in round-robin fashion.
- `first-available`: the new connection is routed to the first available server from the destinations list. In case of failure, the next available server is used. This cycle continues until all servers are unavailable.
- `next-available`: like `first-available`, in that the new connection is routed to the first available server from the destinations list. Unlike `first-available`, if a server is marked as unreachable then it gets discarded and is never used again as a destination.

  This strategy is backward compatible with MySQL Router 2.x's `mode`'s *read-write* behavior. Its limitations include:

  - After all nodes of the selection are discarded, there is no way to add servers back to the list.
  - After restarting MySQL Router, all knowledge of what servers are discarded is lost and all servers are available again.

Defaults

- `PRIMARY`: `round-robin` is default behavior (if routing_strategy is not set), whereas bootstrapping adds `routing_strategy=first-available` to the generated MySQL Router configuration file. The available strategy values are *first-available* and *round-robin*.

  Note

  The bootstrap value changed from *round-robin* to *first-available* in v8.0.16.

- `SECONDARY`: `round-robin` is default behavior (if routing_strategy is not set), whereas bootstrapping adds `routing_strategy=round-robin-with-fallback` to the generated

MySQL Router configuration file. The available strategy values are *first-available*, *round-robin* and *round-robin-with-fallback*.

Note

The bootstrap value changed from *round-robin* to *round-robin-with-fallback* in v8.0.16.

- `PRIMARY_AND_SECONDARY`: `round-robin` is default behavior (if routing_strategy is not set). The available strategy values are *first-available*, *round-robin*.

- `max_connections`

Property	Value
Type	Integer
Default Value	`512`
Minimum Value	`1`
Maximum Value	`65536`

Each routing can limit the number of routes or connections. One possible use is to help prevent possible Denial-Of-Service (DOS) attacks. The default value is 512, and the valid range is between 1 and 65536.

This is similar to MySQL Server's max_connections server system variable.

```
max_connections = 512
```

Note

MySQL Router 2.1.5 and 8.0.4 introduced functionality that increases the concurrent connection limit from around 500 to 5000 connections. This operating system dependent limitation was changed to use a poll() implementation instead of select().

- `thread_stack_size`

Property	Value
Introduced	8.0.12
Type	Integer
Default Value	`64`
Minimum Value	`1`
Maximum Value	`65535`

The stack size allocated for each thread. It is measured in kilobytes, and defaults to 64.

```
[DEFAULT]
thread_stack_size=128
```

- `max_connect_errors`

Property	Value
Type	Integer
Default Value	`100`
Minimum Value	`1`

Property	Value
Maximum Value	`4294967295`

The default value is 100, and the valid range is between 1 and 2^32 (4294967295, an unsigned int).

This is similar to MySQL Server's max_connect_errors server system variable.

This can cause a slight performance penalty if an application performs frequent reconnections, because MySQL Router attempts to discover if connection-related errors are present.

A successful connection resets the error counter (as of 8.0.14).

Each routing has its own list of blocked hosts. Blocked clients receive the MySQL Server error 1129 code with a slightly different error message: "1129: Too many connection errors from fail.example.com". The Router logs contain extra information for blocked clients, such as: INFO [...] 1 authentication errors for fail.example.com (max 100) WARNING [...] blocking client host fail.example.com

```
max_connect_errors = 100
```

- `client_connect_timeout`

Property	Value
Type	Integer
Default Value	`9`
Minimum Value	`2`
Maximum Value	`31536000`

This is similar to MySQL Server's connect_timeout server system variable.

The default value is 9, which is one less than the MySQL 5.7 default. The valid range is between 2 and 31536000.

```
client_connect_timeout = 9
```

- `auth_cache_refresh_interval`

Property	Value
Introduced	8.0.20
Type	Numeric
Default Value	`2`
Minimum Value	`0.001`
Maximum Value	`3600`

Time (in seconds) between the auth-cache refresh attempts. Defaults to 2. The value must be smaller than `auth_cache_ttl` and `ttl` else Router won't start.

This option is applied if the *http_auth_backend* section's `backend` option is set to *metadata_cache*; which is a Router REST API feature.

- `auth_cache_ttl`

Property	Value
Introduced	8.0.20
Type	Numeric

Property	Value
Default Value	`-1`
Minimum Value	`0.001`
Maximum Value	`3600`

Time (in seconds) until the cache becomes invalid if not refreshed. Defaults to -1 (infinite). The value must be larger than `auth_cache_refresh_interval` else Router won't start.

This option is applied if the *http_auth_backend* section's `backend` option is set to *metadata_cache*; which is a Router REST API feature.

- `router_id`

Property	Value
Type	Integer

The MySQL Router ID.

- `ssl_mode`

Property	Value
Type	String
Default Value	`PREFERRED`

Valid Values	`PREFERRED`
	`DISABLED`
	`REQUIRED`
	`VERIFY_CA`
	`VERIFY_IDENTITY`

SSL mode for connecting to the MySQL metadata server. It defaults to `PREFERRED` if not set.

When set to PREFERRED (the default), bootstrapping will warn when SSL is not used and connection to the metadata server is unencrypted.

Available values are DISABLED, PREFERRED, REQUIRED, VERIFY_CA, and VERIFY_IDENTITY. As with the `mysql` client, this value is case-insensitive.

There is also a runtime option for bootstrapping; see `--ssl-mode`.

- `bootstrap_server_addresses`

Property	Value
Deprecated	8.0.14
Type	String

Points to a list of MySQL servers with metadata that can be connected to. After the metadata has been accessed, the metadata cache switches to the servers that are present in the primary ReplicaSet to fetch the metadata. They are also known as bootstrap servers.

This option is deprecated in MySQL Router 8.0.14 and no longer generated by the bootstrap process. Instead, the `dynamic_state` option was added as a replacement.

- `user (MySQL)`

Property	Value
Type	String

A generated MySQL user with privileges to access the MySQL server's metadata schema. This user's password is auto-generated and stored in an encrypted keyring. By default, the encryption key for this keyring is stored in a read protected master key store file, which is defined in the configuration file. Most commonly, this user and associated password are automatically generated during bootstrap. Related command line options are `--force-password-validation` and `--password-retries`. By default, the generated password passes the STRONG validate_password strength.

The password is entirely managed by Router and never exposed, and is stored in a local keyring system using the operating system's account that MySQL Router is running as. It can then be used by Router to connect to InnoDB cluster and retrieve current topology information. Sessions between Router and metadata server are encrypted with SSL by default.

Where the generated keyring files are stored depends on how bootstrap is configured. For self-contained installations (when `--directory` is used), it is stored under `run/` in the self-contained directory. For system-wide installations, it is stored in the system-wide runtime state directory, and that path is platform specific. For additional information, see `master_key_path` and `keyring_path`

This user is assigned (and requires) the following privileges:

```
Privileges needed by the Router account:

 On Metadata Server:

  SELECT ON mysql_innodb_cluster_metadata.*

 On Target Replica Sets:

  SELECT ON performance_schema.replication_group_members
  SELECT ON performance_schema.replication_group_member_stats
```

The generated username follows this pattern: *mysql_router_`[0-9]{1,6}`_`[0-9a-z]{12}`*, where *[0-9]{1,6}* is the numeric router id and *[0-9a-z]{12}* is 12 random lowercase alphanumeric characters. The router id is reused if already present in `mysqlrouter.conf` and its length can not exceed 6 digits.

Note

This user is different from the `user` definition defined in the `[DEFAULT]` section, which is a system user.

- `metadata_cluster`

Property	Value
Type	String

Name of the InnoDB cluster.

Note

SQL query to list the MySQL InnoDB cluster names: *SELECT * FROM mysql_innodb_cluster_metadata.clusters;*

- `use_gr_notifications`

Property	Value
Deprecated	8.0.17
Type	Integer
Default Value	`0`

Valid Values	`0`
	
	`1`

Enables Group Replication notifications. When enabled, Router is asynchronously notified about most cluster changes. It can be enabled manually in `mysqlrouter.conf` or enabled there via the `--conf-use-gr-notifications` command-line bootstrap option.

When enabled, this GR notification feature allows a higher ttl value as ttl becomes an additional safeguard but not as a primary means of keeping the information about the cluster state. It refreshes metadata on each of the four notifications that Group Replication sends: group_replication/membership/quorum_loss, group_replication/membership/view, group_replication/status/role_change, and group_replication/status/state_change.

When disabled, a low `ttl` value (such as 0.5s, the default) is recommended to avoid the overhead of reconnecting to the metadata servers and querying them often.

- `ttl`

Property	Value
Type (>= 8.0.12)	Numeric
Type (<= 8.0.11)	Integer
Default Value (>= 8.0.12)	`0.5`
Default Value (>= 8.0.4, <= 8.0.11)	`5`
Default Value (<= 8.0.3)	`300`
Minimum Value	`0`
Maximum Value (>= 8.0.12)	`3600`
Maximum Value (<= 8.0.11)	`4294967295`

Time to live (in seconds) of information in the metadata cache.

Accepts either an integer or a floating point value. The granularity is limited to milliseconds, where 0.001 equates to one millisecond. Precision is truncated to the supported range; for example *ttl=0.0119* is treated as 11 milliseconds. The value *0* means that the metadata cache module queries the metadata continuously in a tight loop.

The only supported decimal separator is '.' (a period) regardless of locale, and scientific notation, such as *ttl=1.6E-2*, is supported.

Floating point support was added in MySQL Router 8.0.12.

- `level`

Property	Value
Type	String
Default Value	`INFO`
Valid Values	`INFO`

Property	Value

	`DEBUG`
	
	`WARNING`
	
	`ERROR`
	
	`FATAL`

Use the *logger* plugin to log notices, errors, and debugging information. The available log levels are *INFO* (default), *DEBUG*, *WARNING*, *ERROR*, and *FATAL*. These values are case-insensitive.

The *INFO* level displays all informational messages, warnings, and error messages. The *DEBUG* level displays additional diagnostic information from the Router code, including successful routes.

```
[logger]
level = DEBUG
```

Output behavior depends on the `logging_folder` option. Setting `logging_folder` to a folder saves a log file named `mysqlrouter.log` to that folder. Setting `logging_folder` to an empty value, or not setting it, outputs the log to the console. It is set in the *[DEFAULT]* section.

As of MySQL Router 8.0.4, bootstrapping accepts a configuration file using `--config` and utilizes the logger level definition.

- `timestamp_precision`

Property	Value
Introduced	8.0.18
Type	String

The logger timestamp precision; the available definitions with example values are:

- `second, sec, or s`: 2019-05-10 12:10:25
- `millisecond, msec, or ms`: 2019-05-10 12:10:25.428
- `microsecond, usec, or us`: 2019-05-10 12:10:25.428754
- `nanosecond, nsec, ns`: 2019-05-10 12:10:25.428754000

- `port`

Property	Value
Introduced	8.0.16
Type	Integer
Default Value	`8011`

The TCP port listening for HTTP requests; it defaults to 8011.

- `bind_address`

Property	Value
Introduced	8.0.16
Type	String

Property	Value
Default Value	`0.0.0.0`

IP address bound to the HTTP `port`; it defaults to 0.0.0.0.

- `static_folder`

Property	Value
Introduced	8.0.16
Type	String

Base directory for static file requests; it's empty by default. An empty value means no static files are served.

- `require_realm`

Property	Value
Introduced	8.0.16
Type	String

Name of the [http_auth_realm] instance.

- `ssl`

Property	Value
Introduced	8.0.16
Type	Integer
Default Value	`1`
Valid Values	`1` `0`

The value 1 enables SSL, and 0 disables it. TLS clients supporting TLSv1.2 or later are required. This is defined under the [http_server] section.

- `ssl_cert`

Property	Value
Introduced	8.0.16
Type	String

File name of the certificate and its chain certifications in PEM format; required if ssl=1. This is defined under the [http_server] section.

- `ssl_key`

Property	Value
Introduced	8.0.16
Type	String

File name of the key in PEM format; required if ssl=1. This is defined under the [http_server] section.

- `ssl_cipher`

Property	Value
Introduced	8.0.16
Type	String

The cipher-spec (see openssl's 'ciphers' list). Defaults to a comma-separated list of all approved ciphers. Unknown ciphers are silently ignored. Fails if list of ciphers is empty and ssl=1. This is defined under the [http_server] section.

- `ssl_dh_param`

Property	Value
Introduced	8.0.16
Type	String

Read the DH parameter from this file in PEM format. Uses the dh-param from RFC 5114 by default if ssl=1. This is defined under the [http_server] section.

- `backend`

Property	Value
Introduced	8.0.16
Type	String

Name of the [http_auth_backend] section.

- `method`

Property	Value
Introduced	8.0.16
Type	String
Default Value	`basic`

The HTTP authentication method; defaults to basic.

- `name`

Property	Value
Introduced	8.0.16
Type	String

Name of the realm presented to the authentication user.

- `require`

Property	Value
Introduced	8.0.16
Type	String
Default Value	`valid-user`

Requires that the user validates with the authentication backend; defaults to valid-user, which enables this check.

- `backend`

Property	Value
Introduced	8.0.16
Type	String
Default Value	`file`

Name of the backend implementation; accepted values are 'file' (default) or 'metadata_cache'.

Note

metadata_cache support was added in MySQL Router 8.0.20.

```
[http_auth_backend:name]
backend=metadata_cache

[metadata_cache]
auth_cache_refresh_interval=2
auth_cache_ttl=-1
```

- `filename`

Property	Value
Introduced	8.0.16
Type	String

Name of the backend storage file, is relative to the `data_folder` directory.

- `cluster_type`

Property	Value
Introduced	8.0.19
Type	String
Valid Values	`gr` `rs`

The type of AdminAPI object that the Router was bootstrapped against, which is either a InnoDB ReplicaSet (rs) or InnoDB cluster (gr).

Bootstrapping evaluates the target instance and sets this option accordingly in the generated configuration file.

This option was added in MySQL Router 8.0.19; the same version InnoDB ReplicaSet support was added.

4.3.3 Configuration File Example

Here is a basic connection routing example to a MySQL InnoDB cluster named `myCluster`. Both classic MySQL protocol and X Protocol are enabled, it uses TCP/IP connections instead of Unix domain sockets, and it was generated using `--bootstrap` as a standalone configuration with `--directory` set to `/opt/routers/myrouter`.

In this example, read-write (primary) traffic is sent to port 6446 (classic) or 64460 (X Protocol), and read-only (secondaries) are accessed using port 6447 (classic) or 64470 (X Protocol).

The routing section keys (such as *mycluster_default_rw*) are optional, but using these descriptive section keys is helpful for debugging, and also allows multiple configuration sections for the same plugin.

The `destinations` option references *metadata-cache* to utilize InnoDB cluster's metadata cache that dynamically configures host information. Alternatively, `destinations` could be a comma-separated list of hosts to accommodate basic connection routing without InnoDB cluster.

```
# File automatically generated during MySQL Router bootstrap
[DEFAULT]
logging_folder=/opt/routers/myrouter/log
runtime_folder=/opt/routers/myrouter/run
data_folder=/opt/routers/myrouter/data
keyring_path=/opt/routers/router/data/keyring
master_key_path=/opt/routers/myrouter/mysqlrouter.key
connect_timeout=30
read_timeout=30

[logger]
level = INFO

[metadata_cache:mycluster]
router_id=5
bootstrap_server_addresses=mysql://localhost:3310,mysql://localhost:3320,mysql://localhost:3330
user=mysql_router5_6owf3spq1c6n
metadata_cluster=mycluster
ttl=5

[routing:mycluster_default_rw]
bind_address=0.0.0.0
bind_port=6446
destinations=metadata-cache://mycluster/default?role=PRIMARY
routing_strategy=round-robin
protocol=classic

[routing:mycluster_default_ro]
bind_address=0.0.0.0
bind_port=6447
destinations=metadata-cache://mycluster/default?role=SECONDARY
routing_strategy=round-robin
protocol=classic

[routing:mycluster_default_x_rw]
bind_address=0.0.0.0
bind_port=64460
destinations=metadata-cache://mycluster/default?role=PRIMARY
routing_strategy=round-robin
protocol=x

[routing:mycluster_default_x_ro]
bind_address=0.0.0.0
bind_port=64470
destinations=metadata-cache://mycluster/default?role=SECONDARY
routing_strategy=round-robin
protocol=x
```

Chapter 5 MySQL Router Application

Table of Contents

The MySQL Router is an executable that typically runs on the same machine as the application that uses it. This chapter describes the application including available options, how to start the application, and how to use the logging feature.

There are a number of options available for controlling the application when executing `mysqlrouter`. See the `mysqlrouter` documentation for information about the command-line options.

5.1 Starting MySQL Router

MySQL Router requires a configuration file. Although Router searches a predetermined list of default paths for the configuration file, it is common to start Router by passing in a configuration file with the `--config` option.

The process of configuring MySQL Router to automatically start when the host reboots is similar to the steps needed for MySQL server, which is described at Starting and Stopping MySQL Automatically.

For example, when using **systemd**:

```
shell> sudo systemctl start mysqlrouter.service
shell> sudo systemctl enable mysqlrouter.service
```

Example Log Output

Starting MySQL Router generates several log entries, for example when connecting to a sandboxed InnoDB cluster:

```
shell> mysqlrouter --config=/path/to/file/my_router.conf
^C

shell> less /path/to/log/mysqlrouter.log
2019-04-07 16:30:49 INFO    [0x7000022fc000] [routing:devCluster_default_ro] started: listening on 0.0.
2019-04-07 16:30:49 INFO    [0x70000237f000] [routing:devCluster_default_rw] started: listening on 0.0.
2019-04-07 16:30:49 INFO    [0x700002402000] [routing:devCluster_default_x_ro] started: listening on 0.
2019-04-07 16:30:49 INFO    [0x700002485000] [routing:devCluster_default_x_rw] started: listening on 0.
2019-04-07 16:30:49 INFO    [0x700002279000] Starting Metadata Cache
2019-04-07 16:30:49 INFO    [0x700002279000] Connections using ssl_mode 'PREFERRED'
2019-04-07 16:30:49 INFO    [0x700002279000] Connected with metadata server running on 127.0.0.1:3310
2019-04-07 16:30:49 INFO    [0x700002279000] Changes detected in cluster 'devCluster' after metadata re
2019-04-07 16:30:49 INFO    [0x700002279000] Metadata for cluster 'devCluster' has 1 replicasets:
2019-04-07 16:30:49 INFO    [0x700002279000] 'default' (3 members, single-master)
2019-04-07 16:30:49 INFO    [0x700002279000]     localhost:3310 / 33100 - role=HA mode=RW
2019-04-07 16:30:49 INFO    [0x700002279000]     localhost:3320 / 33200 - role=HA mode=RO
2019-04-07 16:30:49 INFO    [0x700002279000]     localhost:3330 / 33300 - role=HA mode=RO
2019-04-07 16:30:49 INFO    [0x700002714000] Connected with metadata server running on 127.0.0.1:3310
```

The log shows that MySQL Router is listening on four ports, lists the active routing strategies by name, InnoDB cluster information, and more.

For example, the first line lists the active routing strategy named `routing:devCluster_default_ro`, is listening on port `6447`, and its mode is `read-only`. The corresponding section in the MySQL Router configuration file looks similar to:

```
[routing:devCluster_default_ro]
bind_address=0.0.0.0
bind_port=6447
```

```
destinations=metadata-cache://devCluster/default?role=SECONDARY
mode=read-only
protocol=classic
```

See how the name, port, and mode were taken directly from the configuration file. In this way, you can quickly determine which routing strategies are active. This could be particularly useful if running several instances of MySQL Router, or if multiple configuration files are loaded.

On Windows, MySQL Router can install, remove, or start the service. By default, the service name is *MySQLRouter*. For additional information, see the `--service` and related command line options for Windows services.

Example Start and Stop Scripts

Bootstrapping MySQL Router with the `--directory` option generates bash scripts to start and stop MySQL Router, which look similar to the following:

```
// *** start.sh ********************** //

#!/bin/bash
basedir=/opt/myrouter
ROUTER_PID=$basedir/mysqlrouter.pid /usr/bin/mysqlrouter -c $basedir/mysqlrouter.conf &
disown %-

// *** stop.sh ********************** //

if [ -f /opt/myrouter/mysqlrouter.pid ]; then
  kill -HUP `cat /opt/myrouter/mysqlrouter.pid`
  rm -f /opt/myrouter/mysqlrouter.pid
fi
```

5.2 Using the Logging Feature

The logging feature can be handy for developing and testing your application and deployment of the MySQL Router. To use logging, enable the logging `level` option in the configuration file under the section named `[logger]`. For example:

```
[logger]
level = INFO
```

Set the log file's location with the `logging_folder` option, defined as a directory path under the `[DEFAULT]` section in the configuration file. The logging file is named `mysqlrouter.log`. For example:

```
[DEFAULT]
# Logs are sent to /path/to/folder/mysqlrouter.log
logging_folder = /path/to/folder

[logger]
level = DEBUG
```

Setting `logging_folder` to an empty string sends logs to the console (stdout).

Two common logging levels are `INFO` (default) and `DEBUG`:

- `INFO`: includes informational messages like those shown above, and is the default mode
- `DEBUG`: includes messages generated inside Router's source code for use in diagnostics. The `DEBUG` mode presents verbose information concerning the inner workings of Router. While it may not be of interest to the application, use of the `DEBUG` mode may be helpful if you encounter a problem or when Router is not behaving as you expect.

The following example shows what the messages look like for the `DEBUG` logging level; compare the `INFO` and `DEBUG` messages:

```
2019-04-07 18:25:56 INFO    [0x700009673000] Connections using ssl_mode 'PREFERRED'
2019-04-07 18:25:56 INFO    [0x700009673000] Connected with metadata server running on 127.0.0.1:3310
2019-04-07 18:25:56 DEBUG   [0x700009673000] Updating metadata information for cluster 'devCluster'
2019-04-07 18:25:56 DEBUG   [0x700009673000] Updating replicaset status from GR for 'default'
2019-04-07 18:25:56 DEBUG   [0x700009673000] Replicaset 'default' has 3 members in metadata, 3 in statu
2019-04-07 18:25:56 DEBUG   [0x700009673000] End updating replicaset for 'default'
2019-04-07 18:25:56 INFO    [0x700009673000] Changes detected in cluster 'devCluster' after metadata re
2019-04-07 18:25:56 INFO    [0x700009673000] Metadata for cluster 'devCluster' has 1 replicasets:
```

Log Rotation

Router supports log rotation; listed here are scenerios with example implementations.

Rotation On Demand

Log rotation on demand can be accomplished in two steps: rename the log file, and then notify Router so it creates and switches to a new log file.

Execute log rotation either directly from the system's shell, or from a script that could be called automatically as a scheduled task. For example:

```
sudo mv /var/log/mysqlrouter/mysqlrouter.log /var/log/mysqlrouter/mysqlrouter.log.old
kill -HUP $(pidof mysqlrouter)
```

logrotate

The logrotate mechanism can also rotate Router's log file. After rotating, Router would be notified to reopen the log file and this is accomplished by sending HUP to the Router process. An example logrotate configuration file:

```
/var/log/mysqlrouter/mysqlrouter.log {
    rotate 9
    size 10M
    create 0755 mysqlrouter mysqlrouter
    postrotate
    kill -HUP $(pidof mysqlrouter)
    endscript
}
```

The example rotates the logs as mysqlrouter.log, mysqlrouter.log.1, ..., mysqlrouter.log.9. The rotation is triggered based on the size of the current mysqlrouter.log file, only if the size is greater than 10MB. Assuming this configuration is saved as `/etc/mysqlrouter/logrotate.conf`, it might be executed periodically (added to cron) as follows:

```
[sudo] logrotate  /etc/mysqlrouter/logrotate.conf
```

Chapter 6 MySQL Router REST API

Table of Contents

MySQL Router REST API interface.

6.1 A Simple MySQL Router REST API

This guide sets up a basic Router REST API, adds basic authentication, and exposes a route to check Router's status. The REST API is configured using configuration sections and options are required to enable and use the REST API. For example, here's a minimal MySQL Router configuration file that enables the most basic REST API functionality:

```
[DEFAULT]
logging_folder=

# Exposes http://127.0.0.1:8081
[http_server]

# Exposes /api/20190715/swagger.json
[rest_api]
```

A typical Router configuration file contains other options but this guide focuses on the REST API. Save this file (our guide assumes (`/foo/mysqlrouter.conf`), start Router loading this file (such as `mysqlrouter -c /foo/mysqlrouter.conf`, and confirm that `http://127.0.0.1:8081/api/20190715/swagger.json` exists. Example `swagger.json` content:

```
{
  "swagger": "2.0",
  "info": {
    "title": "MySQL Router",
    "description": "API of MySQL Router",
    "version": "20190715"
  },
  "basePath": "/api/20190715",
  "tags": [],
  "paths": {},
  "definitions": {}
}
```

This demonstrates that the Router REST API plugin is loaded, and that additional plugins exposing routes and paths are not enabled. Authentication is not required to retrieve `swagger.json`.

Note

The API version number may change in a future release; and future releases may include functionality to retrieve this API integer.

Next, let's enable the simple `rest_router` plugin to expose the *router/status* path. Authentication is required, and enabling authentication requires additional configuration options. For example:

```
[DEFAULT]
logging_folder=

# Exposes http://127.0.0.1:8081
[http_server]

# Exposes /api/20190715/swagger.json
[rest_api]

# Exposes /api/20190715/router/status
```

```
[rest_router]
require_realm=somerealm

# Define our realm
[http_auth_realm:somerealm]
backend=somebackend
method=basic
name=Some Realm

# Define our backend; this file must exist and validate
[http_auth_backend:somebackend]
backend=file
filename=/etc/mysqlrouter/mysqlrouter.pwd
```

Router uses realms for authentication, and the `mysqlrouter_passwd` command-line utility generates and manages these users. For example, this creates a user named *someuser* and saves it as a new file named `/etc/mysqlrouter/mysqlrouter.pwd`:

```
# Generate and save the user/pass
shell> mysqlrouter_passwd set /etc/mysqlrouter/mysqlrouter.pwd someuser
Please enter password:

# Optionally list usernames and salted passwords in the file:
shell> mysqlrouter_passwd list /etc/mysqlrouter/mysqlrouter.pwd

someuser:$5$43tfYEwobPBLkYDB$XnHyC0uXY1F4f6ryd8Vj5CUnEqcH3tqf4pud9kqIji3
```

Restarting Router with our new configuration file generates a different `swagger.json` that now contains [rest_router] plugin information for its */router/status* route:

```
{
  "swagger": "2.0",
  "info": {
    "title": "MySQL Router",
    "description": "API of MySQL Router",
    "version": "20190715"
  },
  "basePath": "/api/20190715",
  "tags": [
    {
      "name": "app",
      "description": "Application"
    }
  ],
  "paths": {
    "/router/status": {
      "get": {
        "tags": [
          "app"
        ],
        "description": "Get status of the application",
        "responses": {
          "200": {
            "description": "status of application",
            "schema": {
              "$ref": "#/definitions/RouterStatus"
            }
          }
        }
      }
    }
  },
  "definitions": {
    "RouterStatus": {
      "type": "object",
      "properties": {
        "timeStarted": {
          "type": "string",
          "format": "data-time"
        },
        "processId": {
```

```
          "type": "integer"
        },
        "version": {
          "type": "string"
        },
        "hostname": {
          "type": "string"
        },
        "productEdition": {
          "type": "string"
        }
      }
    }
  }
}
```

Loading *http://127.0.0.1/api/20190715/routes/status* prompts for a username and password (that we created in our example) and on success returns Router's current status. For example:

```
{
  "processId": 1883,
  "productEdition": "MySQL Community - GPL",
  "timeStarted": "2019-12-24T22:08:30.978640Z",
  "version": "8.0.21",
  "hostname": "boat"
}
```

We set up a basic Router REST API with an authenticated backend; an upcoming guide will explain how to use more useful Router REST API functionality.

6.2 MySQL Router REST API Reference

health

GET `/health`

Check if service is ready to serve client connections. It includes known backends and the cluster's health status.

Available Responses

200	Description: Service is healthy		
	Response Content-Type: *application/json*		
	Response Schema		
	state	string	
		Current state of the metadata cache	
	reason	string	
		Explaination for current state	
500	Description: Service is not healthy		
	Response Content-Type: *application/json*		
	Response Schema		
	state	string	
		Current state of the metadata cache	

	reason	string
		Explaination for current state

metadata

GET `/metadata/{replicasetName}/cache`

Returns status of InnoDB Cluster metadata cache

Available Responses

200 — Description: Status of InnoDB Cluster metadata cache

Response Content-Type: *application/json*

Response Schema

last_refreshed	string
	Last time metadata has been attempted to be refreshed
refresh_failed	integer
	Number of times attempt to refresh metadata failed
refresh_succeeded	integer
	Number of times attempt to refresh metadata succeeded

Path Parameters

replicasetName (required)	string
	Name of a replicaset

GET `/metadata/{replicasetName}/nodes`

Returns nodes configuration and status of InnoDB Cluster nodes

Available Responses

200 — Description: Status and Configuration of InnoDB Cluster metadata nodes

Response Content-Type: *application/json*

Response Schema

replicasets	object

Path Parameters

replicasetName (required)	string
	Name of a replicaset

router

GET `/router/status`

Get status of router

Available Responses

200 — Description: Status of Router

Response Content-Type: *application/json*

Response Schema

running_since	string
	Time at which application was started
version	string
	Version of the application

routing

GET `/routing`

Returns names of routes

Available Responses

200 — Description: Names of routes

Response Content-Type: *application/json*

GET `/routing/{routeName}/config`

Get configuration of route

Available Responses

200 — Description: Configuration of route

Response Content-Type: *application/json*

Response Schema

bind_address	string
	Address the socket is bound to
client_connect_timeout	number
	Connect timeout for clients in seconds
destination_connect_timeout	number
	Connect timeout to destinations in seconds
max_active_connections	integer
	Max allowed parallel connections
protocol	string
	Protocol

404 Description: Route not found

Path Parameters

routeName (required) string

Name of a route

GET `/routing/{routeName}/status`

Get status of a route

Available Responses

200 Description: Status of a route

Response Content-Type: *application/json*

Response Schema

active_connections integer

Currently active connections

total_connections integer

Finished connections since start

404 Description: Route not found

Path Parameters

routeName (required) string

Name of a route

GET `/routing/{routeName}/blockedhosts`

Get blocked hosts of a route

Available Responses

200 Description: Blocked hosts of a route

Response Content-Type: *application/json*

404 Description: Route not found

Path Parameters

routeName (required) string

Name of a route

Appendix A MySQL Router Frequently Asked Questions

A.1. Where do I install MySQL Router?

For best performance, MySQL Router is typically installed on the same host as the application that uses it. Doing so can decrease network latency, allow a local unix domain socket connection to the application instead of TCP/IP, and typically application server's are easiest to scale. But, this is not a requirement as Router can be installed on any host, even its own.

Note

Unix domain sockets can function with applications connecting to MySQL Router, but not for MySQL Router connecting to a MySQL Server.

A.2. Can I run more than one instance of the router application?

Yes, see also the `--directory` bootstrap option.

A.3. How do I make the router application highly available?

Use MySQL Router as part of InnoDB cluster. For additional details, see InnoDB Cluster.

A.4. Does the router inspect packets?

No.

A.5. Does the router impact performance?

Whenever you introduce a component in a communication stream there will be a certain amount of overhead incurred and is affected heavily by workload. Fortunately, performance testing on the current release has shown approximately 1% within the same speed as a direct connection for simple redirect connection routing.

A.6. Please explain the different MySQL Router versions, especially why Router went from 2.1.4 to 8.0.3.

MySQL Router 2.0 was the initial version and is meant for MySQL Fabric users. It has since been deprecated and is no longer supported.

MySQL Router 2.1 was introduced to support MySQL InnoDB cluster, and it also added new features such as bootstrapping.

MySQL Router 8.0 expands on MySQL Router 2.1 but with a version number that aligns with MySQL Server. In other words, Router 2.1.5 was released as Router 8.0.3 (along with MySQL Server 8.0.3), and the 2.1.x branch was replaced by 8.0.x. The two branches are fully compatible.

A.7. Can I bind the router to multiple IP addresses?

No, the `bind_address` option in the configuration file accepts only one address. However, it is possible to use `bind_addres = 0.0.0.0` to bind to all ports on the localhost.

A.8. What is the difference between the different scheduling modes and strategies?

Before version 8.0, the `mode` option determined the scheduling strategy. Setting `mode=read-write` means Router uses the first destination host until it fails and then moves to the next until all hosts were attempted and failed. Setting `mode=read-only` cycles through the list of host destinations in a circular (round-robin) manner retrying servers that may have failed previously.

Router 8.0 introduced the `routing_strategy` option as a replacement to the now deprecated `mode` option. It offers the *first-available*, *next-available*, *round-robin* and *round-robin-with-fallback* strategies. See the `routing_strategy` documentation for additional details.

The *next-available* routing strategy is identical to the *read-write* mode's schedule, and the *round-robin* routing strategy is identical to the *read-only* mode's schedule.

A.9. How many concurrent connections does each MySQL Router instance support?

Over 5000 as of MySQL Router 2.1.5 and 8.0.4, depending on the operating system's poll() limits, and just over 500 in earlier versions due to their internal use of select() instead of poll().

A.10. How can I configure MySQL Router to use a non-default directory on a system using AppArmor?

If you use the `--directory` option on a system using AppArmor, for example Ubuntu, you could encounter a permissions error related to MySQL Router accessing the non-default directory. In this case, add the path you pass to `--directory` to the AppArmor file as suggested, and restart AppArmor.