MySQL Router 2.1

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.

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 either the MySQL Forums or MySQL Mailing Lists, where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Document generated on: 2017-09-20 (revision: 54051)

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 this document 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 this document for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Legal Notices

Documentation Accessibility

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

Access to Oracle Support

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

Chapter 1 General Information

Table of Contents

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

Router uses a configuration file to define how routing is performed, and can be configured to enable several applications to use a single router.

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 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 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 instances in the cluster. As long as there are enough online replicas, and communication between the components is intact, applications will be able to contact one of them. Router also makes it possible for this to happen by simply repointing applications to connect to 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 Router with an InnoDB cluster after the cluster is configured are:

1. Install MySQL Router.

   For details, see the Installation section.

2. Bootstrap for an InnoDB cluster, and test.

   Router can be automatically configured by calling it with `--bootstrap`. During bootstrap, Router connects to the cluster, fetches its metadata, and configures itself for use. For details, see Chapter 3, *Deploying MySQL Router*.

3. Set up Router for automatic startup.

   To make Router automatically start when the host reboots, you need to configure your system to start Router. This process is 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 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 the PRIMARY with a metadata server
- Creates a self-contained installation with all generated directories and files at `/opt/myrouter/`
- Only the host's system user named `snoopy` will have access to `/opt/myrouter/*`
- Files and directories are generated under `/opt/myrouter/` including `start.sh`, `stop.sh`, `log/`, and a fully functional MySQL Router configuration file named `mysqlrouter.conf`.

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

Bootstrapping and group_replication_single_primary_mode

When bootstrapping, the available ports and sockets are affected by the `group_replication_single_primary_mode` MySQL server configuration option.

Note

This document refers to default bootstrapping behavior. Other MySQL Router configuration options may affect this behavior, and generated configuration values can be modified after bootstrapping.

- With *group_replication_single_primary_mode*=*ON* (the default): Both Read-Write (primary) and Read-Only (secondary) ports are configured.
- With *group_replication_single_primary_mode*=*OFF*: Only Read-Write (primary) ports are configured.

For example:

With *group_replication_single_primary_mode*=*ON*, all connections to ports 6446 and 64460 go to the single primary, and all connections to ports 6447 and 64470 go to the secondaries using the round-robin mode schedule.

```
shell> mysqlrouter --bootstrap localhost:3310

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
```

With *group_replication_single_primary_mode*=*OFF*, all connections to ports 6446 and 64460 go to the primaries using the round-robin mode schedule.

```
shell> mysqlrouter --bootstrap localhost:3310
```

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

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

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.

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 enables redirection of 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”.

This means you can set up your application to connect to MySQL Router, and retry the connection if the current MySQL server fails as Router then selects a new MySQL server to redirect the connection to. 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 the router to listen for connections on port 7002 of the localhost, and then redirect those connections to any of the servers in the list named by the `destinations` option, including servers running on the localhost listening on ports 3306, 3307, and 3308. Finally, we use the `mode` option to tell the router to allow both readers and writers. For more information about the available modes, see the section entitled, Configuration File Setup below.

```
[routing:simple_redirect]
bind_port = 7002
mode = read-write
destinations = localhost:3306,localhost:3307,localhost:3308
```

Notice that the section is entitled, `routing:simple_redirect`. The first part, `routing` is called the section name and is used internally to determine which plugin to load. The last part is an option's section key (name) you can optionally provide should you want to set up more than one routing strategy.

When a server is no longer reachable, MySQL Router moves to the next server destination in the list, and halts redirection if the list is exhausted because this is the default mode schedule when the `mode` option is set to *read-write*.

1.4 Application Considerations

Using Router does not require specific libraries or interfaces. Aside from managing the MySQL Router instance, your application is written as if 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 connection made at startup that do 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. Thus, if a 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 you a better idea of why you may want to use the router, and a look into how it is used from an application.

Scenerios

There are several possible scenarios for MySQL Router, such as the following:

- 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 2.1

This section summarizes many of the new features added to MySQL Router 2.1, in relation to MySQL Router 2.0.

MySQL Router is part of InnoDB cluster, which is also new with this MySQL Router release.

Features

- Bootstrapping support was added. For details about bootstrapping, see Deploying with Bootstrapping.
- A metadata cache plugin was added. It is the information repository of the managed MySQL topology information that MySQL Router uses to route the MySQL server clients to the appropriate location. For additional information, see Section 1.2, “Cluster Metadata and State”.
- Keyring key management was added to securely manage passwords, and is used to secure the MySQL users that fetch metadata. For additional information, see documentation for the `master_key_path` and `keyring_path` configuration options.

Command Line Options

- *Bootstrapping*: `--bootstrap`, `--bootstrap-socket` (2.1.4+), `--conf-base-port`, `--conf-bind-address`, `--conf-use-sockets`, `--conf-skip-tcp`, `--directory`, `--force-password-validation` (2.1.4+), `--password-retries` (2.1.4+), `--force`, and `--name`
- *SSL*: `--ssl-mode`, `--ssl-ca`, `--ssl-capath`, `--ssl-cipher`, `--ssl-crl`, `--ssl-crlpath`, and `--tls-version`.

Configuration File Options

- `protocol`: the protocol configuration option was added to support the X Protocol. Setting `protocol` to `x` enables the X Protocol for connections, otherwise the default `classic` protocol is used.
- `master_key_path` and `keyring_path`: paths to files that store passwords using the new keyring management feature.

Package and Build Related Changes

- Windows: downloads now require Visual C++ Redistributable for Visual Studio 2015, when before the 2013 version was required.

Additional Changes

- Help output (mysqlrouter `--help`) now includes the current default folder locations for the system, and usage examples.
- MySQL Fabric support was removed.
- The default configuration file was renamed from `mysqlrouter.ini` to `mysqlrouter.conf`. For backward compatability, Router still looks for the *.ini* variant in each directory.

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, Red Hat, Oracle Linux, 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 http://dev.mysql.com/downloads/router.
- **Download the source code and compile yourself**: The source code is available at http://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 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

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

  ```
  shell> sudo dpkg -i mysql-apt-config_0.6.0-1_all.deb
  ```

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

- Update your APT repository:

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

- Next, 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*`-1ubu1404-amd64.deb`, where *version* is the MySQL Router version number.

Installing RPM packages

On Red Hat-based systems, and other systems that use the RPM package format, 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:

```
shell> sudo rpm -Uvh mysql-community-release-el7-7.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*`-1fc10.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. If you wish to also remove the configuration files and data directory, use this command:

```
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 http://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 downloading the ZIP Archive.

Note

The Windows binaries were added in MySQL Router 2.0.4.

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.

Installation Using MySQL Installer

The general MySQL Installer download is available at http://dev.mysql.com/downloads/windows/installer/. The MySQL Installer application can install, upgrade, and manage most MySQL products, including MySQL Router.

Recommended Approach

Managing all of your MySQL products, including MySQL Router, with MySQL Installer is the recommended approach. It handles all requirements and prerequisites, configurations, 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 2.1`, 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”.

Installation Using the ZIP Archive

The ZIP Archive download is available at http://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/router2.1
```

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-2.1.4-src.tar.gz
shell> cd mysql-router-2.1.4-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) that were installed on the system. This file is useful for uninstalling MySQL Router.

However, there are a couple of things that you need to do in order to prepare your system for compiling the source code.

2.4.1 Prerequisites

The following components and libraries are required to compile MySQL Router.

- 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.

  Oracle Linux 6 works as well, but you have to install the Software Collection Library 1.2. For RedHat and CentOS, see Docs and Downloads. For Oracle Linux, 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
- 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

MySQL Router v2.1.4 on Linux (64-bit) (GPL community edition)
Copyright (c) 2015, 2017, Oracle and/or its affiliates. All rights reserved.

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

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.

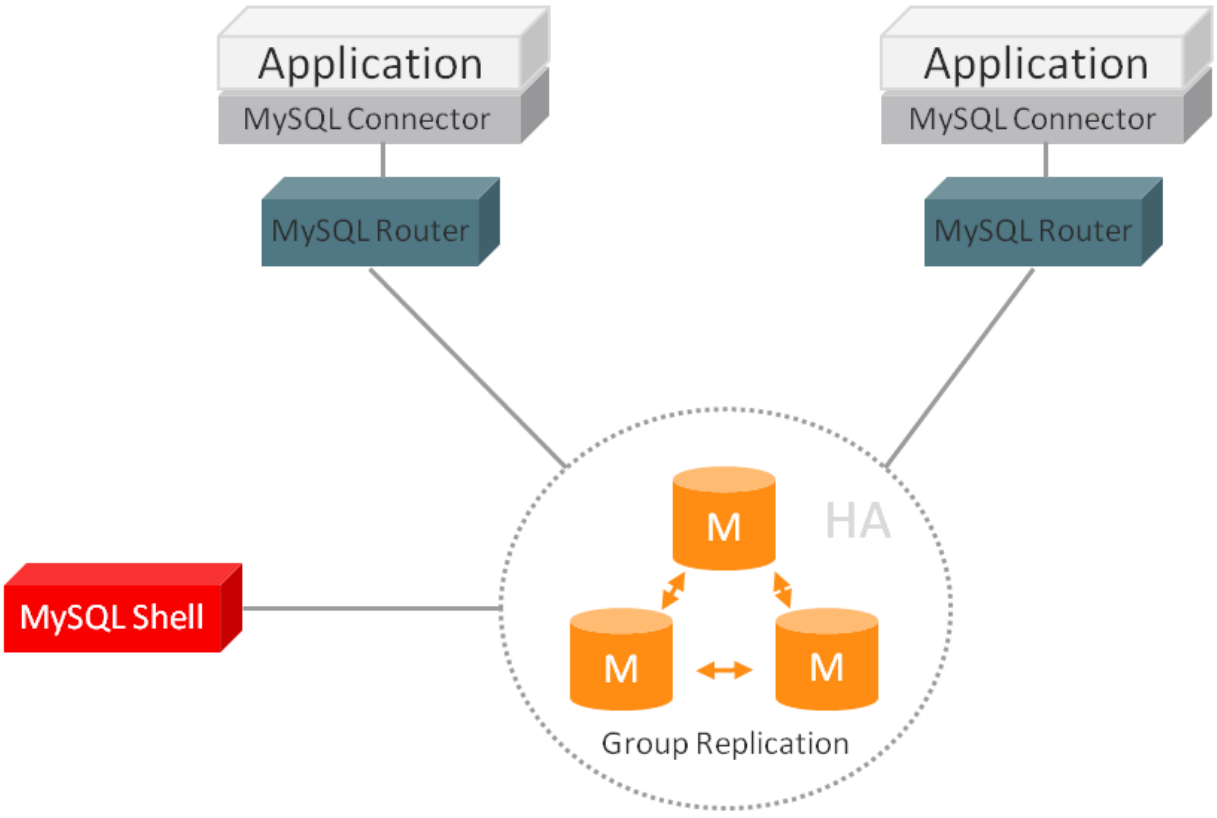
Note

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@192.168.23.45* instead of a value like *myapp@%*.
- Typically application servers are easiest to scale.

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

Figure 3.1 Example MySQL Router Deployment



3.1 Bootstrapping

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

Note

Bootstrapping was added in MySQL Router 2.1.

This example creates a standalone MySQL Router instance using the `--directory` option, enables sockets, and assumes that an InnoDB cluster named `clusterFriend` 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...
MySQL Router  has now been configured for the InnoDB cluster 'clusterFriend'.

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

Classic MySQL protocol connections to cluster 'clusterFriend':
- Read/Write Connections: localhost:6446
- Read/Write Connections: /tmp/myrouter/mysql.sock
- Read/Only Connections: localhost:6447
- Read/Only Connections: /tmp/myrouter/mysqlro.sock

X protocol connections to cluster 'clusterFriend':
- Read/Write Connections: localhost:64460
- Read/Write Connections: /tmp/myrouter/mysqlx.sock
- Read/Only Connections: localhost:64470
- Read/Only Connections: /tmp/myrouter/mysqlxro.sock

shell> cd /tmp/myrouter
shell> ./start.sh

PID 29294 written to /tmp/myrouter/mysqlrouter.pid
```

A generated MySQL Router directory looks similar to:

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

data
log
mysql.sock
mysqlro.sock
mysqlrouter.conf
mysqlrouter.key
mysqlrouter.pid
mysqlx.sock
mysqlxro.sock
run
start.sh
stop.sh
```

A generated MySQL Router configuration file will look 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

[logger]
```

```
level = INFO

[metadata_cache:clusterFriend]
router_id=1
bootstrap_server_addresses=mysql://localhost:3310,mysql://localhost:3320,mysql://localhost:3330
user=mysql_router1_jy95yozko3k2
metadata_cluster=clusterFriend
ttl=300

[routing:clusterFriend_default_rw]
bind_address=0.0.0.0
bind_port=6446
socket=/tmp/myrouter/mysql.sock
destinations=metadata-cache://clusterFriend/default?role=PRIMARY
mode=read-write
protocol=classic

[routing:clusterFriend_default_ro]
bind_address=0.0.0.0
bind_port=6447
socket=/tmp/myrouter/mysqlro.sock
destinations=metadata-cache://clusterFriend/default?role=SECONDARY
mode=read-only
protocol=classic

[routing:clusterFriend_default_x_rw]
bind_address=0.0.0.0
bind_port=64460
socket=/tmp/myrouter/mysqlx.sock
destinations=metadata-cache://clusterFriend/default?role=PRIMARY
mode=read-write
protocol=x

[routing:clusterFriend_default_x_ro]
bind_address=0.0.0.0
bind_port=64470
socket=/tmp/myrouter/mysqlxro.sock
destinations=metadata-cache://clusterFriend/default?role=SECONDARY
mode=read-only
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 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.

```
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 Getting Started with 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.
- 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 Getting Started with 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",
        "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"
            }
        }
    }
}
```

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 will also use the metadata cache plugin to securely store the credentials.

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

Please enter MySQL password for root:

Bootstrapping MySQL Router instance at /opt/myrouter...
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 /opt/myrouter

shell> ./start.sh

PID 28817 written to /opt/myrouter/mysqlrouter.pid
```

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

Testing the Router

Now connect to router as you would any other MySQL Server, for example using the standard `mysql` client:

Note

Using the option `--conf-use-sockets` during MySQL Router bootstrap also configures Unix domain socket connections.

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

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

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

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

As demonstrated, we connected to MySQL Router on port 6446 but see we are connected to port 3310 (our PRIMARY). It also shows how connecting to one of the secondaries on port 6447 shows a connection to one of the secondary MySQL instances, in this case on port 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)

mysql-js> dba.killSandboxInstance(3310)
The MySQL sandbox instance on this host in
/Users/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:read_only]
bind_address = localhost
bind_port = 7001
destinations = foo.example.org:3306,bar.example.org:3306,baz.example.org:3306
mode = read-only

[routing:read_write]
bind_address = localhost
bind_port = 7002
destinations = foo.example.org:3306,bar.example.org:3306
mode = read-write
```

Here we use connection routing to round-robin MySQL connections to three MySQL servers on port 7001, as the *read-only* `mode` causes round-robin behavior. This example also configures the *read-write* mode for two of the servers using port 7002. The read-write mode defaults to the first-available strategy, as described in the `mode` option's documentation. 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 mode, so it is up the the application to determine where to send read and write requests, so either port 7001 or 7002 in our example.

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 will send requests to the appropriate MySQL instance. 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 will first connect 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 used. For additional information about this behavior, see documentation for the `mode` option.

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. For additional information, see `--bootstrap`. This also means that editing the configuration file becomes optional with bootstrapping because the generated `mysqlrouter.conf` is fully functional.

4.1 Configuration File Syntax

The format of the configuration file 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 a 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
mode = read-only # Read only mode
```

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`

Name	Description
`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
`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, depending 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

MySQL Router v2.1.4 on macOS v10.12 (64-bit) (GPL community edition)
Copyright (c) 2015, 2017, Oracle and/or its affiliates. All rights reserved.

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

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, by default MySQL Router scans the following locations, 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.

  Common 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 -c /custom/path/to/router.conf -a /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. In addition, 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*.

4.3 Configuration Options

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

When `bootstrapping`, the generated configuration and files depend on which bootstrap options are 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 Command Line Options

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

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.

The following tables summarize the MySQL Router options read by the `mysqlrouter` command. Detailed information about each of these options, such as descriptions and allowed values, is documented below these tables.

General Options

Table 4.2 General

Format	Description
--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.
--user	Run mysqlrouter as the user having the defined user name or numeric user id.
--version	Display version information and exit.

Bootstrapping Options

Table 4.3 Bootstrapping

Format	Description	Introduced
--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 conjuction with --bootstrap.	2.1.4
--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-sockets	Whether to use Unix domain sockets.	

Format	Description	Introduced
--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.	2.1.4
--name	Gives a symbolic name for the router instance.	
--password-retries	The number of retries to use for generating the Router's user password.	2.1.4

SSL Options

These options match behaviour and functionality to their counterparts in the `mysql` client, as described at Command Options for Encrypted Connections. These options are used during both bootstrapping and during normal router operations. The values can be lower, mixed, or upper case, and is preserved when bootstrap writes to the configuration file.

Table 4.4 SSL

Format	Description
--ssl-ca	Path to SSL CA file to verify server's certificate against.
--ssl-capath	Path to directory containing SSL CA files to verify server's certificate against.
--ssl-cipher	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-mode	SSL connection mode for use during bootstrap and normal operation, when connecting to the metadata server. Analogous to --ssl-mode in the mysql client.
--tls-version	Comma-separated list of TLS versions to request, if SSL is enabled.

Windows Services Options

Specific to Microsoft Windows, these control the configuration of MySQL Router as a service.

Table 4.5 Windows Services

Format	Description
--clear-all-credentials	Clear all stored credentials
--install-service	On Windows, install MySQL Router as a service named MySQLRouter, and set it to automatically start when Windows restarts.
--install-service-manual	On Windows, install MySQL Router as a service named MySQLRouter, that can be manually started.
--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

MySQL Router Command Line Option Descriptions

The following list contains descriptions of each startup option available for use with `mysqlrouter`, including allowed and default values. Options noted as boolean need only be specified in order to take effect; you should not try to set a value for them.

- `--version`, `-v`

Command-Line Format	`--version`

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

```
shell> mysqlrouter --version

MySQL Router v2.1.4 on Linux (64-bit) (GPL community edition)
```

- `--help`, `-h`

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

MySQL Router v2.1.4 on Linux (64-bit) (GPL community edition)
Copyright (c) 2015, 2017, Oracle and/or its affiliates. All rights reserved.

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

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] [-h|--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`*

Command-Line Format	`--bootstrap URI`	
Permitted Values	Type	string

The main option to perform a bootstrap of the router by connecting to the MySQL metadata server at the URI. A password is prompted if needed. If a username is not passed to the URI then the default user name "root" is used.

Note

While `--bootstrap` accepts a URI for TCP/IP connections, also passing in `--bootstrap-socket` with a local Unix domain socket name will replace the "host:port" part in the `--bootstrap` definition with the socket on the same machine.

By default, bootstrap will perform a system-wide configuration of the router. Only one instance of the router may be configured for system-wide operation. The system instance of the router has a router_name of "default". If additional instances are desired, then use the `--directory` to create self-contained router installations.

If a configuration file already exists on bootstrap, the existing router_id in that file will be reused, and a reconfiguration process will occur. The configuration file will be regenerated from scratch and the router's metadata server account will be 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 Router instance. This account is used by 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

Bootstrapping was added in MySQL Router 2.1.1.

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

Introduced	2.1.4
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 directory`, `-d`

Command-Line Format	`--directory directory`	
Permitted Values	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.

- `--conf-use-sockets`

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-skip-tcp`

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`

Command-Line Format	`--conf-base-port port`	
Permitted Values	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`

Command-Line Format	`--conf-bind-address address`	
Permitted Values	Type	string
	Default	`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.

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

Command-Line Format	`--user name`	
Platform Specific	Linux	
Permitted Values	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` *`name`*

Command-Line Format	`--name name`	
Permitted Values	**Type**	string

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`

Introduced	2.1.4
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` *`password-retries`*

Introduced	2.1.4	
Command-Line Format	`--password-retries password-retries`	
Permitted Values	**Type**	integer
	Default	`20`
	Min Value	`1`
	Max 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`

Command-Line Format	`--force`

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

- `--ssl-mode mode`

Command-Line Format	`--ssl-mode mode`	
Permitted Values	Type	string
	Default	`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-cipher ciphers`

Command-Line Format	`--ssl-cipher ciphers`	
Permitted Values	Type	string
	Default	

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

- `--tls-version versions`

Command-Line Format	`--tls-version versions`	
Permitted Values	Type	string
	Default	

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

- `--ssl-ca path`

Command-Line Format	`--ssl-ca path`	
Permitted Values	Type	string
	Default	

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

- `--ssl-capath directory`

Command-Line Format	`--ssl-capath directory`	
Permitted Values	Type	string
	Default	

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

- `--ssl-crl path`

Command-Line Format	`--ssl-crl path`	
Permitted Values	Type	string
	Default	

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

- `--ssl-crlpath directory`

Command-Line Format	`--ssl-crlpath directory`	
Permitted Values	Type	string
	Default	

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

- `--config path`, `-c`

Command-Line Format	`--config 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 path`, `-a`

Command-Line Format	`--extra-config 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`

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`

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`

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

Remove the Router Windows service.

This option is only available on Windows.

- `--service`

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

Start Router as a Windows service.

This option is only available on Windows.

- `--update-credentials-section`

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`

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`

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.

4.3.2 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 4.6 [DEFAULT]

Option Name	Description	Type
`config_folder`	Path to configuration files	string
`keyring_path`	Path to keyring file	string
`logging_folder`	Path to router logs	string
`master_key_path`	Path to master keyring file	string
`plugin_folder`	Path to router plugins	string
`runtime_folder`	Path to runtime files	string
`user`	System user that router is run as	string

Routing Options

Table 4.7 [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 a MySQL Server connection is considered timed out	integer
`destinations`	Routing destinations as a comma-separated list of MySQL servers	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
`socket`	Path to unix domain socket file	string

Metadata Cache Options

Table 4.8 [metadata_cache]

Option Name	Description	Type
`bootstrap_server_addresses`	MySQL servers with metadata, as a comma-separated list	string

Option Name	Description	Type
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
ttl	Time To Live, in seconds	integer
user	MySQL user that accesses the MySQL Server's metadata schema	string

Logging Options

Table 4.9 [logger]

Option Name	Description	Type
level	Logging level	string

MySQL Router Configuration File Option Descriptions

- logging_folder

Permitted Values	Type	string
	Default	$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

Permitted Values (Windows)	Type	string
	Default	
Permitted Values (Other)	Type	string
	Default	/usr/local/lib/mysqlrouter

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

Permitted Values (Windows)	**Type**	string
	Default	
Permitted Values (Other)	**Type**	string
	Default	`/run/mysqlrouter`

Path to the MySQL Router runtime files.

Default value: `/run/mysqlrouter`

- `config_folder`

Permitted Values (Windows)	**Type**	string
	Default	
Permitted Values (Other)	**Type**	string
	Default	`/usr/local/etc/mysqlrouter`

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`

- `keyring_path`

Permitted Values (Windows)	**Type**	string
	Default	`%PROGRAMDATA%\MySQL\MySQL Router\keyring-data`
Permitted Values (Other)	**Type**	string
	Default	`/run/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
```

Note

This option was added in MySQL Router 2.1.

- `master_key_path`

Permitted Values (Windows)	**Type**	string

	Default	`%PROGRAMDATA%\MySQL\MySQL Router \mysqlrouter.key`
Permitted Values (Other)	**Type**	string
	Default	`/run/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
```

Note

This option was added in MySQL Router 2.1.

- `user (system)`

Permitted Values	**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.

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

An account created by the official MySQL Router packages does not have shell access and its home directory points to the directory where the default configuration file is stored.

Note

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

- `bind_address`

Permitted Values	**Type**	string
	Default	`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`

Permitted Values	**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`

Platform Specific	Unix	
Permitted Values	**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`

Permitted Values	Type	string
	Default	`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).

Note

The protocol option, and general X protocol support, was added in Router 2.1.

- `connect_timeout`

Permitted Values	Type	integer
	Default	`1`

	Min Value	`1`
	Max 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:

```
connect_timeout = 1
```

- `destinations`

Permitted Values	**Type**	string

Provides a comma-separated list of destination addresses that should be used when establishing connections. Exact behavior depends on the `mode` option, and its associated strategy.

Example usage:

```
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".

- `mode`

Permitted Values	**Type**	string
	Valid Values	`read-write`
		`read-only`

Setting this parameter is **required**, and each mode has different scheduling. Two modes are supported:

- **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, and will continue trying each MySQL server on the list. If no more MySQL servers are available on the list, then routing is aborted. This method is also known as "first-available".

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

- **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.

  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 checked, and when available they are put back onto the available `destinations` list. The destinations order is maintained.

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

- `max_connections`

Permitted Values	Type	integer
	Default	`512`
	Min Value	`1`
	Max 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
```

Known limitation: A MySQL Router instance can handle just over 500 concurrent connections. See also this related FAQ .

- `max_connect_errors`

Permitted Values	Type	integer
	Default	`100`
	Min Value	`1`
	Max Value	`4294967296`

The default value is 100, and the valid range is between 1 and 2^32 (4294967296, 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.

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`

Permitted Values		
	Type	integer
	Default	`9`
	Min Value	`2`
	Max 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
```

- `router_id`

Permitted Values		
	Type	integer

The MySQL Router ID.

- `ssl_mode`

Permitted Values		
	Type	string
	Default	`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 boostrapping; see `--ssl-mode`.

- `bootstrap_server_addresses`

Permitted Values		
	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.

- `user (MySQL)`

Permitted Values	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`

Permitted Values	Type	string

Name of the InnoDB cluster.

Note

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

- `ttl`

Permitted Values	Type	integer
	Default	`300`

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

- `level`

Permitted Values	Type	string
	Default	`INFO`
	Valid Values	`INFO`
		`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.

4.3.3 Configuration File Example

Here is a basic connection routing example to a MySQL InnoDB cluster named `mycluster`. Both classic and X protocols 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), and read-only (secondaries) are accessed using port 6447 (classic) or 64470 (x).

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.

```
# 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

[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=300

[routing:mycluster_default_rw]
bind_address=0.0.0.0
```

```
bind_port=6446
destinations=metadata-cache://mycluster/default?role=PRIMARY
mode=read-write
protocol=classic

[routing:mycluster_default_ro]
bind_address=0.0.0.0
bind_port=6447
destinations=metadata-cache://mycluster/default?role=SECONDARY
mode=read-only
protocol=classic

[routing:mycluster_default_x_rw]
bind_address=0.0.0.0
bind_port=64460
destinations=metadata-cache://mycluster/default?role=PRIMARY
mode=read-write
protocol=x

[routing:mycluster_default_x_ro]
bind_address=0.0.0.0
bind_port=64470
destinations=metadata-cache://mycluster/default?role=SECONDARY
mode=read-only
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`. Those options are documented at Section 4.3.1, “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
2017-04-07 16:30:49 INFO    [0x7000022fc000] [routing:devCluster_default_ro] started: listening on 0.0.
2017-04-07 16:30:49 INFO    [0x70000237f000] [routing:devCluster_default_rw] started: listening on 0.0.
2017-04-07 16:30:49 INFO    [0x700002402000] [routing:devCluster_default_x_ro] started: listening on 0.
2017-04-07 16:30:49 INFO    [0x700002485000] [routing:devCluster_default_x_rw] started: listening on 0.
2017-04-07 16:30:49 INFO    [0x700002279000] Starting Metadata Cache
2017-04-07 16:30:49 INFO    [0x700002279000] Connections using ssl_mode 'PREFERRED'
2017-04-07 16:30:49 INFO    [0x700002279000] Connected with metadata server running on 127.0.0.1:3310
2017-04-07 16:30:49 INFO    [0x700002279000] Changes detected in cluster 'devCluster' after metadata re
2017-04-07 16:30:49 INFO    [0x700002279000] Metadata for cluster 'devCluster' has 1 replicasets:
2017-04-07 16:30:49 INFO    [0x700002279000] 'default' (3 members, single-master)
2017-04-07 16:30:49 INFO    [0x700002279000]     localhost:3310 / 33100 - role=HA mode=RW
2017-04-07 16:30:49 INFO    [0x700002279000]     localhost:3320 / 33200 - role=HA mode=RO
2017-04-07 16:30:49 INFO    [0x700002279000]     localhost:3330 / 33300 - role=HA mode=RO
2017-04-07 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:

```
2017-04-07 18:25:56 INFO    [0x700009673000] Connections using ssl_mode 'PREFERRED'
2017-04-07 18:25:56 INFO    [0x700009673000] Connected with metadata server running on 127.0.0.1:3310
2017-04-07 18:25:56 DEBUG   [0x700009673000] Updating metadata information for cluster 'devCluster'
2017-04-07 18:25:56 DEBUG   [0x700009673000] Updating replicaset status from GR for 'default'
2017-04-07 18:25:56 DEBUG   [0x700009673000] Replicaset 'default' has 3 members in metadata, 3 in statu
2017-04-07 18:25:56 DEBUG   [0x700009673000] End updating replicaset for 'default'
2017-04-07 18:25:56 INFO    [0x700009673000] Changes detected in cluster 'devCluster' after metadata re
2017-04-07 18:25:56 INFO    [0x700009673000] Metadata for cluster 'devCluster' has 1 replicasets:
```

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 round-robin and first-available scheduling modes?

Round-robin differs from first-available in that it will cycle through the list of servers specified in the `destinations` option in a circular manner retrying servers that may have failed previously while first-available will stop once it reaches the end of the list.

Round-robin scheduling is enabled by using the `read-only` mode and first-available scheduling is enabled by using the `read-write` mode.

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

Currently just over 500 due to our internal use of select().