MySQL Router 8.4

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 MySQL AdminAPI.

MySQL Router 8.4 is highly recommended for use with MySQL Server 8.4.

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

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

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

Document generated on: 2024-07-26 (revision: 79150)

Table of Contents

Preface and Legal Notices

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

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

Legal Notices

License Restrictions

Warranty Disclaimer

Restricted Rights Notice

Hazardous Applications Notice

measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Documentation Accessibility

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

Access to Oracle Support for Accessibility

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

Chapter 1 General Information

Table of Contents

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

MySQL Router officially supports active MySQL Server versions equal to or below the MySQL Router version. For example, MySQL Router 8.4 officially supports MySQL 8.0 and 8.4, but also makes an effort to support MySQL 9.x.

1.1 Routing for MySQL InnoDB Cluster

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

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

Introduction

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

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

Deploying Router with MySQL InnoDB Cluster

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

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

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

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

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

3. Set up MySQL Router for automatic startup.

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

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

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

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

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

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

1.2 Cluster Metadata and State

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

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

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

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

Note

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

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

1.2.1 MySQL Router Read Replica Support

MySQL Router reads the values defined in the metadata field, `v2_router_options.router_options.read_only_targets`, to retrieve routing information for read-only traffic.

`v2_router_options.router_options.read_only_targets` is populated by the AdminAPI method *`cluster`*`.setRoutingOption()` which sets the routing policy to one of the following values using the `read_only_targets` option:

- `all`: all Read Replicas and Secondary cluster members are used for read-only traffic.
- `read_replicas`: only Read Replicas are used for read-only traffic.
- `secondaries`: only Secondary cluster members are used for read-only traffic.

Note

If `read_only_targets` is not present, or set to a value other than `all`, `read_replicas`, or `secondaries`, MySQL Router defaults to `secondaries` and logs a warning message.

MySQL Router does not use Read Replicas as a source for Cluster metadata. Also, it is not possible to use a Read Replica in a MySQL Router bootstrap command. An error is returned for any attempt to bootstrap with a Read Replica.

Failure Handling

MySQL Router does not route connections to Read Replicas in the following situations:

- If there is no quorum in the Cluster.
- If all Cluster members are in OFFLINE state.
- If no Cluster members can be reached when checking their Group Replication state.

MySQL Router routing policy is affected by configuration in the following ways:

- If the Cluster state is INVALID and the `invalidated_cluster_policy` is set to `drop_all`, Read Replicas are not used for new read-only connections and all existing connections to the Read Replicas are dropped.
- If the Cluster state is INVALID and the `invalidated_cluster_policy` is set to `allow_ro`, Read Replicas are used for new read-only connections and existing connections to the Read Replicas are unaffected.

MySQL Router uses the standard quarantine mechanism for Read Replicas, as defined by the `desination_status` configuration parameters. See Destination Status Options.

1.3 Connection Routing

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

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

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

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

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

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

1.4 Application Considerations

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

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

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

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

Scenarios

There are several possible scenarios for MySQL Router, including:

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

- As an administrator, I want to configure a routing strategy 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:

```
$> 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`).

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.

System Requirements

- MySQL Router supports the same platforms as MySQL Server, as listed here: https://www.mysql.com/support/supportedplatforms/database.html
- `Hardware`: Minimum requirement is 1 CPU Core and 256 MB of RAM. 4+ CPU Cores and 4+ GB of RAM is recommended.
- `Disk Space`: Minimum requirement is 100 MB.
- `External libraries`: Most external dependencies, such as protobuf and rapidjson, are bundled within the MySQL Router packages. One exception is OpenSSL, which is only bundled for Windows builds. Package managers should resolve the OpenSSL dependency and install the proper OpenSSL version as required.

2.1 Installing MySQL Router on Linux

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

Installation options include:

- **Official MySQL Yum or APT repository packages**: These binaries are built by the MySQL Release team. For additional information about installing these, see the quick guides for installing them using Yum or APT.
- **Download official MySQL packages**: Downloads are available at https://dev.mysql.com/downloads/router. Download and install using your preferred package manager.
- **Download the source code and compile yourself**: The source code is available as part of MySQL Server at https://dev.mysql.com/downloads/mysql. Alternatively, the source code is also available on GitHub (specifically in the `router` directory).

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

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

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

Installing DEB packages

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

Using the APT Package Manager

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

   Note

   Download the APT configuration package from here.

   ```
   $> sudo dpkg -i mysql-apt-config_0.8.30-1_all.deb
   ```

   Choose the desired MySQL Server series to install, such as MySQL Server 8.4. This choice also determines the MySQL Router version that is installed from the MySQL repository.

2. Update your APT repository:

   ```
   $> sudo apt-get update
   ```

3. Install MySQL Router. For example:

   ```
   $> sudo apt-get install mysql-router-community
   ```

Manually Installing a Package

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

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

package`.deb` is the MySQL Router package name; for example, `mysql-router-community-`*version*`-1ubuntu24.04_amd64.deb`, where *version* is the MySQL Router version number.

Installing RPM packages

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

Using the Yum Package Manager

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

  Note

  Download the Yum configuration package from here.

  ```
  $> sudo rpm -Uvh mysql84-community-release-el7-1.noarch.rpm
  ```

- Next, install MySQL Router. For example:

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

Manually Installing an RPM Package

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

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

Uninstalling

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

Uninstalling DEB packages

To uninstall a Debian package, use this command:

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

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

```
$> 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:

```
$> sudo rpm -e mysql-router-community
```

Note

Similarly, use `yum remove mysql-router-community`.

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 with Docker

The Docker deployment framework supports easy installation and configuration of MySQL Router. This section explains how to use a MySQL Router Docker image.

You need to have Docker installed on your system before you can use a MySQL Router Docker image. See Install Docker for instructions.

Important

You need to either run `docker` commands with `sudo`, or create a `docker` user group, and then add to it any users who want to run `docker` commands. See

details here. Because Docker containers are always run with root privileges, you should understand the Docker daemon attack surface and properly mitigate the related risks.

Basic Steps for MySQL Router Deployment with Docker

Warning

The MySQL Docker images maintained by the MySQL team are built specifically for Linux platforms. Other platforms are not supported, and users using these MySQL Docker images on them are doing so at their own risk.

Downloading a MySQL Router Docker Image

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

```
$> docker pull container-registry.oracle.com/mysql/community-router:tag
```

The *tag* is the label for the image version you want to pull (for example, `8.0`). If **:tag** is omitted, the `latest` label is used, and the image for the latest GA version of MySQL Community Router is downloaded. Refer to Oracle Container Registry and navigate to the MySQL Router image in the MySQL repository for a complete list of tags for available versions.

Table 2.1 Variables

Variable	Description
`MYSQL_HOST`	Required. MySQL host to connect to.
`MYSQL_PORT`	Required. MySQL server listening port.
`MYSQL_USER`	Required. MySQL user to connect with.
`MYSQL_PASSWORD`	Required. String. MySQL user's password.
`MYSQL_INNODB_CLUSTER_MEMBERS`	Optional. Integer. Wait for this number of cluster instances to be online.
`MYSQL_CREATE_ROUTER_USER`	Optional. Boolean. Whether to create a new account for MySQL Router to use when running. Default value is enabled (1). Set to 0 (zero) to disable.
`MYSQL_ROUTER_BOOTSTRAP_EXTRA_OPTIONS`	Optional. Comma-separated list of additional command line options to apply during bootstrapping.

Running in a container requires a working InnoDB cluster. If supplied, the run script waits for the given mysql host to start, the InnoDB cluster to have the `MYSQL_INNODB_CLUSTER_MEMBERS`-defined number of members, and then uses the supplied host for bootstrapping. See Section 3.1, “Bootstrapping MySQL Router”.

For example:

```
$> docker run \
  -e MYSQL_HOST=localhost \
  -e MYSQL_PORT=3306 \
  -e MYSQL_USER=mysql \
  -e MYSQL_PASSWORD=mysql \
  -e MYSQL_INNODB_CLUSTER_MEMBERS=3 \
  -e MYSQL_ROUTER_BOOTSTRAP_EXTRA_OPTIONS="--conf-use-socket --conf-use-gr-notification" \
```

```
  -ti container-registry.oracle.com/mysql/community-router
```

To use a specific version of MySQL Router, add a tag to the `-ti` value. For example: `-ti container-registry.oracle.com/mysql/community-router:8.4.1` for MySQL Router 8.4.1. To use the latest version, do not add a tag.

Checking the status:

```
$> docker ps
```

For additional details, see Oracle Container Registry and navigate to the MySQL Router image in the MySQL repository.

2.3 Installing MySQL Router on macOS

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

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

2.4 Installing MySQL Router on Windows

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

Windows Prerequisites

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

Installing Using MSI

To install MySQL Router on Microsoft Windows using the MSI Installer, do the following:

1. Download the **Windows (x86, 64-bit), MSI Installer** package from http://dev.mysql.com/downloads/router/.
2. When prompted, click **Run**.
3. Follow the steps in the Setup Wizard.

Installing the ZIP Archive

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

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

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

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

2.5 Installing MySQL Router from Source Code

MySQL Router is part of the MySQL Server source code tree; compiling MySQL Server also compiles MySQL Router. This assumes `-DWITH_ROUTER=ON`, which is enabled by default. The instructions here are brief, see Installing MySQL from Source for specific prerequisites and additional details.

Note

MySQL Router source code can be found in the `router` directory inside the MySQL Server source code repository.

Get Source Code

To compile MySQL Router, download the MySQL Server source code from https://dev.mysql.com/downloads/mysql. Alternatively, git clone mysql-server on GitHub.

Download and unpack the MySQL Server source files, for example:

```
$> tar xzf mysql-8.4.0.tar.gz
$> cd mysql-8.4.0
```

Once this is complete, you need to configure using cmake as you would for MySQL Server.

Configure

The CMake program provides control over how you configure a source distribution. Typically, you do this using options on the CMake command line. The CMake options are not documented here, see MySQL Source-Configuration Options.

To compile the source code, create a folder to contain the compiled binaries and executables, run cmake to create the make file, and then compile the code. See Installing MySQL Server from Source for additional details, including platform specific prerequisites and concerns.

Note

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

Begin by executing the `cmake` command to create the make file. The following commands are run from the root of the MySQL Server source code tree:

```
$> mkdir build && cd build
$> cmake ..
```

Executing `cmake` may yield errors related to missing libraries or tools. For example, macOS builds may need to reference a newer `bison` executable:

```
$> cmake .. -DBISON_EXECUTABLE=/usr/local/opt/bison/bin/bison
```

Compile

You can compile MySQL Server as you normally would (simply `make`) as it also compiles MySQL Router, or build MySQL Router specific targets. For example, to only build MySQL Router with its libraries, plugins, and tests:

```
$> make mysqlrouter_all
```

Optionally execute the MySQL Router specific tests with `ctest`:

```
$> ctest -R routertest_
```

Installation

There is not a make option to only install MySQL Router from source because executing `make install` initiates a full MySQL Server build.

Developer Related Notes

Notes related to using and testing a locally compiled MySQL Router version for development purposes:

- To run a local build without `make install`, configure Router to find the newly built `plugin_folder` as compiling generates a non-standard installation directory structure. Either manually edit the generated `mysqlrouter.conf` or set it during bootstrap, for example with: `--conf-set-option=DEFAULT.plugin_folder=../plugin_output_directory`

  Similarly, also set `runtime_folder` accordingly, for example: `--conf-set-option=DEFAULT.runtime_folder=../runtime_output_directory`

- While individual targets do produce binaries, such as `make mysqlrouter_password`, building all Router targets is recommended

- To avoid building unit tests, also configure with `-DWITH_UNIT_TESTS=0`

2.6 Upgrading MySQL Router

MySQL Router as a part of InnoDB Cluster

MySQL Router is most commonly used as an InnoDB Cluster component; with Router bootstrapped against the cluster. For related information, see Section 3.1, “Bootstrapping MySQL Router”.

If No Metadata Upgrade Needed

MySQL Router can be upgraded independently of the InnoDB Cluster components if a metadata upgrade is not needed.

Since the assumption is that the Router configuration file and state file remain backward compatible, the simplest upgrade scenario is to install a new version using an installer/upgrade package for the system. In most cases, the installer handles stopping and restarting the running instance after the upgrade. If that is not the case (such as installing from the source or a tar.gz archive) then the running Router instance must be manually stopped and restarted after the installation/upgrade process.

If a Metadata Upgrade Needed

When the InnoDB Cluster requires a cluster metadata schema upgrade, MySQL Router must be upgraded as a part of the metadata upgrade procedure described in the MySQL Shell guide at Upgrade Metadata Schema.

MySQL Router logs indicate if existing metadata is incompatible with the new version with an error, such as:

```
This version of MySQL Router is not compatible with the provided MySQL InnoDB cluster metadata
```

Bootstrapping Router after installing a new version

Usually bootstrapping is not needed after the upgrade. The exceptions to this are:

- If the new Router version introduces new capabilities, another bootstrap operation is required to use them. For example, if an active cluster is part of a ClusterSet, bootstrapping sets up the appropriate configuration options to work with a ClusterSet.
- If the new MySQL Router is installed at a different location than the previous version; in that case the configuration file will contain paths (such as `plugin_folder`) to the previous installation. Manually changing the existing configuration file is an alternative.

Standalone MySQL Router (not a part of InnoDB Cluster)

Since the assumption is that the Router configuration file and state file remain backward compatible, the simplest upgrade scenario is to install a new version using an installer/upgrade package for the system. In most cases, the installer handles stopping and restarting the running instance after the upgrade. If that is not the case (such as installing from the source or a tar.gz archive) then the running Router instance must be manually stopped and restarted after the installation/upgrade process.

The existing configuration file is likely compatible with the new version but would require adjusting to set newly added options.

Chapter 3 Deploying MySQL Router

Table of Contents

Performance Recommendations

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

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

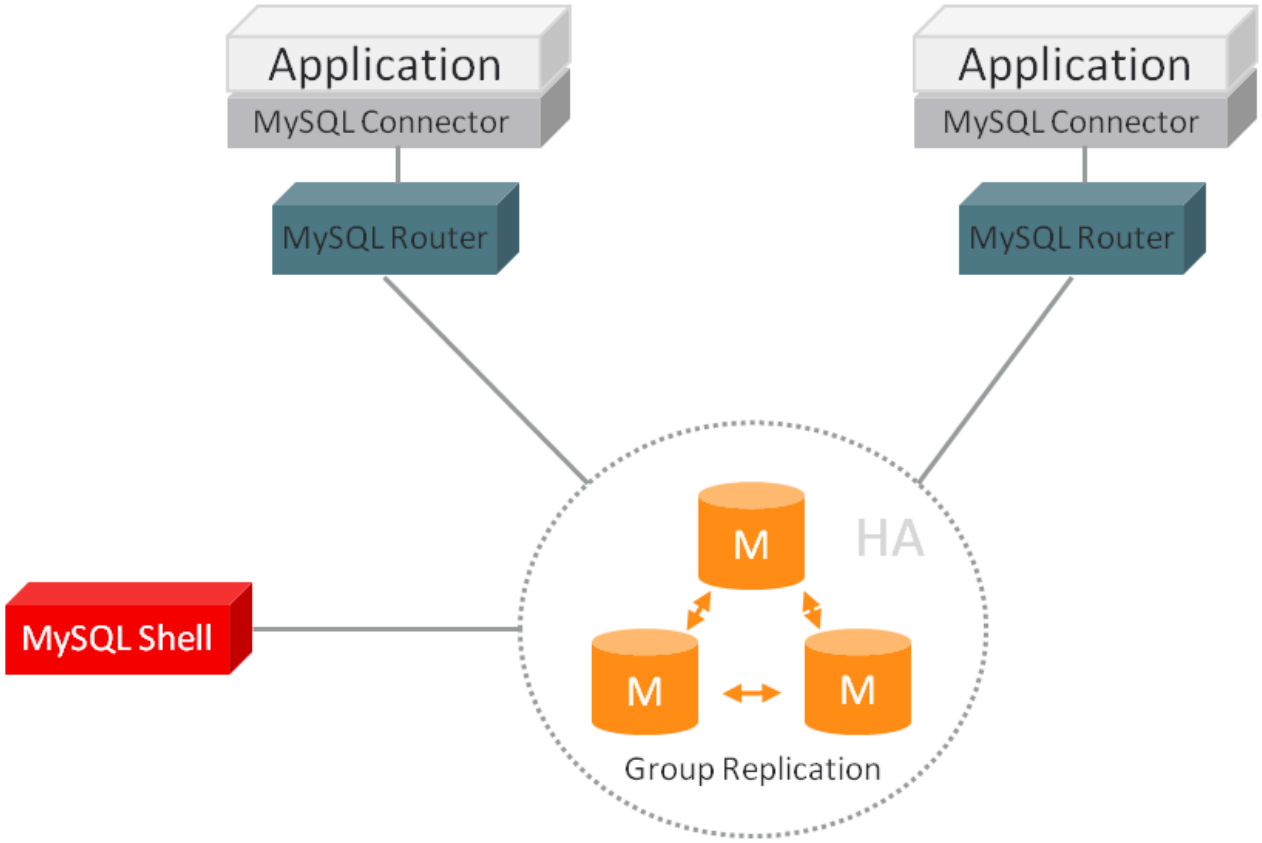
Note

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

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

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

Figure 3.1 Example MySQL Router Deployment



3.1 Bootstrapping MySQL Router

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

This example creates a standalone MySQL Router instance using the `--directory` option, enables sockets, uses `--account` to customize Router's MySQL username, and sets `--account-create` to `always` to only bootstrap if the account does not already exist. This example assumes that an InnoDB Cluster named `myCluster` already exists.

```
$> mysqlrouter --bootstrap root@localhost:3310 --directory /tmp/myrouter
  --conf-use-sockets --account routerfriend --account-create always

Please enter MySQL password for root:

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

Please enter MySQL password for routerfriend:

- Creating account(s)
- Verifying account (using it to run SQL queries that would be run by Router)
- Storing account in keyring
- Adjusting permissions of generated files
- Creating configuration /tmp/myrouter/mysqlrouter.conf

# MySQL Router configured for the InnoDB Cluster 'myCluster'

After this MySQL Router has been started with the generated configuration
```

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

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

## MySQL Classic protocol

- Read/Write Connections: localhost:6446, /tmp/myrouter/mysql.sock
- Read/Only Connections:  localhost:6447, /tmp/myrouter/mysqlro.sock

## MySQL X protocol

- Read/Write Connections: localhost:6448, /tmp/myrouter/mysqlx.sock
- Read/Only Connections:  localhost:6449, /tmp/myrouter/mysqlxro.sock
```

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

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

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

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

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

[logger]
level = INFO

[metadata_cache:myCluster]
cluster_type=gr
router_id=1
user=routerfriend
metadata_cluster=myCluster
ttl=0.5
auth_cache_ttl=-1
auth_cache_refresh_interval=2
use_gr_notifications=0

[routing:myCluster_rw]
bind_address=0.0.0.0
bind_port=6446
socket=/tmp/myrouter/mysql.sock
destinations=metadata-cache://myCluster/?role=PRIMARY
routing_strategy=first-available
protocol=classic

[routing:myCluster_ro]
bind_address=0.0.0.0
bind_port=6447
socket=/tmp/myrouter/mysqlro.sock
destinations=metadata-cache://myCluster/?role=SECONDARY
```

```
routing_strategy=round-robin-with-fallback
protocol=classic

[routing:myCluster_x_rw]
bind_address=0.0.0.0
bind_port=6448
socket=/tmp/myrouter/mysqlx.sock
destinations=metadata-cache://myCluster/?role=PRIMARY
routing_strategy=first-available
protocol=x

[routing:myCluster_x_ro]
bind_address=0.0.0.0
bind_port=6449
socket=/tmp/myrouter/mysqlx.sock
destinations=metadata-cache://myCluster/?role=SECONDARY
routing_strategy=round-robin-with-fallback
protocol=x
```

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

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

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

```
$> 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 Deploying a Production 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 MySQL AdminAPI.

  For a scripted approach, see Scripting AdminAPI.

- 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.

The following example uses the AdminAPI method to set up our cluster sandbox. This is a brief overview, so see MySQL 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 "...".

```
$> 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": "127.0.0.1:3310",
      "ssl": "REQUIRED",
      "status": "OK",
      "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
      "topology": {
          "127.0.0.1:3310": {
              "address": "127.0.0.1:3310",
              "memberRole": "PRIMARY",
              "mode": "R/W",
              "readReplicas": {},
              "replicationLag": null,
              "role": "HA",
              "status": "ONLINE",
              "version": "8.0.27"
          },
          "127.0.0.1:3320": {
              "address": "127.0.0.1:3320",
              "memberRole": "SECONDARY",
              "mode": "R/O",
```

```
            "readReplicas": {},
            "replicationLag": null,
            "role": "HA",
            "status": "ONLINE",
            "version": "8.0.27"
        },
        "127.0.0.1:3330": {
            "address": "127.0.0.1:3330",
            "memberRole": "SECONDARY",
            "mode": "R/O",
            "readReplicas": {},
            "replicationLag": null,
            "role": "HA",
            "status": "ONLINE",
            "version": "8.0.27"
        }
    },
    "topologyMode": "Single-Primary"
  },
  "groupInformationSourceMember": "127.0.0.1:3310"
}

mysql-js> \q

Bye!
```

Set Up the Router

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

```
$> mysqlrouter --bootstrap root@localhost:3310 --directory /tmp/router

Please enter MySQL password for root:
# Bootstrapping MySQL Router instance at '/tmp/router'...

- Creating account(s) (only those that are needed, if any)
- Verifying account (using it to run SQL queries that would be run by Router)
- Storing account in keyring
- Adjusting permissions of generated files
- Creating configuration /tmp/router/mysqlrouter.conf

# MySQL Router configured for the InnoDB Cluster 'myCluster'

After this MySQL Router has been started with the generated configuration

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

InnoDB Cluster 'myCluster' can be reached by connecting to:

## MySQL Classic protocol

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

## MySQL X protocol

- Read/Write Connections: localhost:6448
- Read/Only Connections:  localhost:6449

$> cd /tmp/router

$> ./start.sh
```

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

Testing the Router

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

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

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

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

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

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

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

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

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

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

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

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

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

mysql-js> dba.killSandboxInstance(3310)

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

Killing MySQL instance...

Instance localhost:3310 successfully killed.
```

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

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

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

$> 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.

Router also enables a REST API by default in the generated `mysqlrouter.conf` at bootstrap, and by default the following URL displays a `swagger.json` for your local setup: `https://127.0.0.1:8443/api/20190715/swagger.json`. See also Chapter 6, *MySQL Router REST API*.

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 4.3.3, “Configuration File Options”.

```
[logger]
level = INFO

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

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

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

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

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

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

```
$> ./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:

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

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

3.4 Connection Sharing and Reuse

MySQL Router enables server connections to be pooled and shared. If a client disconnects, the server connection is moved to the connection pool, where it is available for reuse. If the client connection is idle for more than a specified time, the server connection idles until a new client connection is established. This lowers the number of connections the server has to maintain and frees up resources normally bound to idling connections.

MySQL Router tracks the statements executed by the client and the SQL state of the session to ensure client connections do not lose their session state. If a connection is shared, the reconnected session is in the state the client left it. If that is not possible, the connection is not shared.

Warnings and errors generated by statements are captured and returned when requested by the client. As are session variables.

Note

The default number of I/O threads is the same as the number of CPU threads supported by the host and can be configured with the `threads` configuration option.

Limitations

- Connection sharing is not supported in PASSTHROUGH mode or if `server-ssl-mode=AS_CLIENT` and `client-ssl-mode=PREFERRED`.
- Connection sharing is only supported for classic connections.
- SQL statements that depend on previous session state will not work when connection sharing is active, unless inside a transaction.
- Certain features will leave the connection in a state that blocks it from being shared when idle. Closing or resetting the connection (COM_RESET_CONNECTION) will allow the connection to be reused again.

Unsupported SQL Features

The following statements and functions are not supported when connection sharing is active, except inside a transaction.

- `GET DIAGNOSTICS`
- `LAST_INSERT_ID()`

SQL Features which Prevent Sharing

The following SQL features prevent the connection from being pooled until the connection is closed or reset by the client.

- `SQL_CALC_FOUND_ROWS`,
- `GET_LOCK()`, `service_get_write_locks()`. and `SQL_CALC_FOUND_ROWS`
- User variables
- Temporary tables
- Prepared statements

Note

Transactions and `LOCK TABLES` also block connection sharing until the transaction is closed, or the lock released.

Configuration

Connection sharing is configured using the following options:

- `connection_sharing`
- `connection_sharing_delay`
- `max_idle_server_connections`
- `idle_timeout`

The following is an example of configuring connection sharing during bootstrap:

```
      --conf-set-option=routing:bootstrap_rw.connection_sharing=1
      --conf-set-option=routing:bootstrap_ro.connection_sharing=1
      --conf-set-option=connection_pool.max_idle_server_connections=32
```

3.5 Read/Write Splitting

MySQL Router supports Read-Write splitting. This configuration enables you to direct all read traffic to read-only instances, and all write traffic to read-write instances.

Read-write instances are primaries or sources. Read-only instances are secondaries in an InnoDB Cluster or the primary or secondary instances in a Replica Cluster.

MySQL Router classifies each query as read or write and directs it to the appropriate backend. It is also possible to manually, or programmatically, specify the type of query using `ROUTER SET` or `query_attributes`.

Note

Each client session can communicate with one read_write and one read_only destination.

3.5.1 Configuration

To enable read-write splitting, the following `router` options must be enabled:

- `access_mode`: must be set to `auto`.

Note

It is possible to define `read_write` and `read_only` `access_mode` values per session, only. See Per-Session Configuration.

- `connection_sharing`: must be set to `1`.
- `protocol`: must be set to `classic`.
- `destinations`: must be set to a `metadata-cache` URL with the role set to `PRIMARY_AND_SECONDARY`.

Per-Session Configuration

Read-write splitting configuration can be defined per session, using one of the following:

- `ROUTER SET optionName='value'`
- `query_attributes router.optionName value`

The following are the possible *`optionNames`* and values:

- `access_mode` set to one of the following values:
  - `read_write`: all session traffic is sent to a read_write server.
  - `read_only`: all session traffic is sent to a read_only server.
  - `auto`: the server is selected based on the type of transaction, reads are targetted to read_only servers, writes to read_write servers.
- `wait_for_my_writes [ 0 | 1 ]`: If enabled, `1`, read-only queries wait for the last written transaction of the session.
- `wait_for_my_writes_timeout [ 0 | 4294967295 ]`: Maximum time in seconds to wait for a read_only destination to apply the written transaction, before falling back to a read_write destination. Default is 1.

Note

Session variables are reset to their initial values if the client sends a `change_user` or `reset_connection`.

For example:

```
SQL> ROUTER SET access_mode='read_write'
```

```
SQL> query_attribute router.access_mode read_write
```

Bootstrapping

When MySQL Router is bootstrapped, the default configuration is created with the following values:

```
[DEFAULT]
max_idle_server_connections=64

[router:read_write_split]
bind_port=6450
destinations=metadata-cache://mycluster/?role=PRIMARY_AND_SECONDARY
routing_strategy=round-robin
access_mode=auto
protocol=classic
connection_sharing=1
```

To disable this configuration, you must bootstrap with `--disable-rw-split`.

3.5.2 Statements

- The following describes read-only statements:
  - Statements are read-only if they start with:
    - `SELECT`
    - `DO`
    - `VALUES`
    - `TABLE`
    - `WITH` that is not followed by `UPDATE` or `DELETE`.
    - `EXPLAIN`, `DESCRIBE`, or `DESC` which are not followed by `UPDATE` or `DELETE`.
    - `HELP`
    - `USE`
    - `CHECKSUM`
    - Parenthesized Query Expressions.
  - If they start with any of the above and do not contain functions or keywords which must be executed on a read-write server. Such as:
    - `GET_LOCK()`
    - `FOR UPDATE|SHARE`
    - `LOCK IN SHARE MODE`
  - Some functions can produce a write from within a read-only statement. Such statements fail with an error similar to the following

    ```
    The MySQL server is running with the --super-read-only option so it cannot execute this statement.
    ```

    Such statements can be explicitly routed to a read-write server, with `ROUTER SET` or `query_attributes`. See Per-Session Configuration. You can also wrap the statement in a `START TRANSACTION ... COMMIT`.
- The following describes read-write statements:

- Data Definition Statements.
- The following DML: `CALL`, `INSERT`, `UPDATE`, `DELETE`, `REPLACE`, `IMPORT TABLE`, `LOAD DATA`, `LOAD XML`, `WITH...UPDATE/DELETE`.
- The following Account Management statements: `GRANT`, `REVOKE`, `RENAME USER`, `CREATE ROLE`, `CREATE USER`, `DROP ROLE`, `DROP USER`, `SET PASSWORD`, `SET ROLE`, `SET DEFAULT ROLE`.
- The following Transaction and Locking statements: `BEGIN`, `START TRANSACTION` , `XA`, `SAVEPOINT`, `ROLLBACK`, `COMMIT`.
- The following database administration statements: `SHOW CREATE...`, `SHOW VARIABLES`, `SHOW STATUS`, `SET TRANSACTION` (including `SET SESSION ...`), `SET NAMES`, `SET CHARACTER SET`, `FLUSH PRIVILEGES`.
- The following utility statements: `EXPLAIN`, `DESCRIBE`, `DESC`.
- The following table maintenance statements: `ANALYZE TABLE`, `OPTIMIZE TABLE`, `CHECK TABLE`, `REPAIR TABLE`.

- The following statements are not supported if `access_mode=auto`:
  - Any statement which is not read-only or read-write. This includes ALTER RESOURCE GROUP Statement, Replication Statements, and Other Administrative Statements.

3.6 MySQL Router TLS Session Cache

TLS sessions from client to router and router to server can be cached and resumed when needed. This shortens the connection handshake, saving time and resources.

MySQL Router uses the following caches:

- Client TLS session cache: Caches TLS session from the client to MySQL Router.
- Server TLS session cache: Caches TLS sessions from the MySQL Router to the server.

The following configuration options control the session caching:

- Client TLS session cache:
  - `client_ssl_session_cache_mode`: Enables or disables the cache for client-router TLS sessions.

    Note

    Enabled by default. If this parameter is not set, the cache is enabled. To disable the cache, you must explicitly define it.
  - `client_ssl_session_cache_size`: Defines the maximum number of sessions cached.
  - `client_ssl_session_cache_timeout`: Defines the maximum amount of time, in seconds, a session remains in the cache. If the timeout is reached, and this session is not reused, the session is removed from the cache and the connection is closed.
- Server TLS Cache:
  - `server_ssl_session_cache_mode`: Enables or disables the cache for router-server sessions.

Note

Enabled by default. If this parameter is not set, the cache is enabled. To disable the cache, you must explicitly define it.

- `server_ssl_session_cache_size`: Defines the maximum number of sessions cached.
- `server_ssl_session_cache_timeout`: Defines the maximum amount of time, in seconds, a session remains in the cache. If the timeout is reached, and this session is not reused, the session is removed from the cache and the connection is closed.

3.7 MySQL Router Set Trace

MySQL Router supports tracing of statements as they are processed by MySQL Router from client to server and the response to the client. The trace is returned as JSON.

This enables debugging, testing, application connection comparisons, and so on.

Configuration

To configure `ROUTER SET trace` you must add the following to your MySQL Router configuration file:

- `max_idle_server_connections`: add to the `DEFAULT` section. This must be set to at least 1.
- The following values can be added to the `DEFAULT` section and apply to all connections, or you can add them to the individual `ROUTING: ...` sections of connections you want to examine in detail.
  - `client_ssl_mode`: Set to `PREFERRED` or `REQUIRED`.
  - `server_ssl_mode`: Set to `PREFERRED`,`REQUIRED`, or `DISABLED`.
  - `connection_sharing`: Set to 1 to enable connection sharing.

For example:

```
[DEFAULT]
max_idle_server_connections=64

[routing:{...}]
client_ssl_mode=PREFERRED
server_ssl_mode=PREFERRED
connection_sharing=1
```

Enable ROUTER TRACE

ROUTER TRACE can be enabled per session or per statement on the command line of your MySQL client.

- Enable per session:

```
ROUTER SET TRACE = 1;
```

- Disable per session:

```
ROUTER SET TRACE = 0;
```

- Enable per statement:

```
query_attributes router.trace 1;
```

- Disable per statement:

```
query_attributes router.trace 0;
```

Trace Format

The trace is returned in a JSON object with the following properties:

- `start_time`: Date and time string denoting the start of the span.
- `end_time`: Date and time string denoting the end of the span.
- `elapsed_in_span_us`: Microseconds spent in the current span. This value is `end_time` minus `start_time`.
- `status_code`: Represents the canonical status code of a finished Span. Default value is empty.
- `name`: Name of the event.
- `attributes`: Attributes of the event.
- `events`: An array of events. These contain the following:
  - `timestamp`: Date and time string.
  - `name`: Name of the event.
  - `attributes`: Attributes of the event.

For example:

```
> mysql --host=127.0.0.1 --port=6446 --show-warnings
> ROUTER SET trace = 1;
> SELECT @@port;
+--------+
| @@port |
+--------+
|   3306 |
+--------+
1 row in set, 1 warning (0,02 sec)
Note (code 4600): {
  "start_time": "2023-03-23T15:31:08.052442Z",
  "end_time": "2023-03-23T15:31:08.052653Z",
  "elapsed_in_span_us": 211,
  "name": "mysql/query",
  "attributes": {
    "mysql.sharing_blocked": false
  },
  "events": [
    {
      "timestamp": "2023-03-23T15:31:08.052444Z",
      "name": "mysql/query_classify",
      "attributes": {
        "mysql.query.classification": "change_on_tracker"
      }
    },
    {
      "start_time": "2023-03-23T15:31:08.052455Z",
      "end_time": "2023-03-23T15:31:08.052495Z",
      "elapsed_in_span_us": 39,
      "name": "mysql/connect_and_forward",
      "attributes": {
```

```
        "mysql.remote.is_connected": true,
        "mysql.remote.endpoint": "localhost:3306",
        "mysql.remote.connection_id": 17,
        "db.name": ""
      },
      "events": [
        {
          "start_time": "2023-03-23T15:31:08.052458Z",
          "end_time": "2023-03-23T15:31:08.052495Z",
          "elapsed_in_span_us": 36,
          "name": "mysql/forward"
        }
      ]
    },
    {
      "start_time": "2023-03-23T15:31:08.052623Z",
      "end_time": "2023-03-23T15:31:08.052627Z",
      "elapsed_in_span_us": 3,
      "name": "mysql/response",
      "attributes": {
        "mysql.session.@@SESSION.statement_id": "84"
      }
    }
  ]
}
```

Trace Events

The following trace events and attributes are supported:

`mysql/query`

MySQL Router receives a query.

Attributes:

- `mysql.sharing_blocked`: Boolean. If connection sharing is blocked, `mysql.sharing_blocked_by` is displayed along with a reason why sharing is blocked.
- `mysql.sharing_blocked_by`: String. Displays the reason why connection sharing is blocked. This can be one of the following values:
  - `trx-state`: A transaction is active.
  - `trx-characteristics`: Transaction state is set. For example, `SET TRANSACTION READ ONLY`.
  - `some-state-changed`: The session is in an unrecoverable state.
  - `session-track-gtids`: `session_track_gtids` does not contain the expected value.
  - `session-track-state-change`: `session_track_state_change` does not contain the expected value.
  - `session-track-transaction-info`: `session_track_state_change` does not contain the expected value.

`mysql/query_classify`

Describes how MySQL Router analyzed the statement in the context of connection-sharing.

Attributes:

- `mysql.query.classification`: comma-separated list of none or more of the following:
  - `accept_session_state_from_session_tracker`: The statement resulted in a notification from the session tracker which was accepted as is.
  - `ignore_session_tracker_some_state_changed`: The statement resulted in a notification from the session tracker which was ignored.
  - `session_not_sharable_on_error`: Statements such as `SET known_variable = 1, unknown_variable = 2` can cause a session state change, although the statement failed. The server responds with an error, but no session tracker, even though the session state changed.
  - `session_not_sharable_on_success`: Set if a statement modifies the session state, but the session tracker does not report it.
  - `forbidden_function_with_connection_sharing`: The statement contains functions or keywords which are not possible with connection sharing. Such as `GET DIAGNOSTICS` or `LAST_INSERT_ID()`.
  - `forbidden_set_with_connection_sharing`: The statement attempted to set the session tracker information required for connection sharing.

`mysql/connect_and_forward`

Attributes:

- `mysql.remote.is_connected`: Boolean. If `fale`, there is no connection. If `true`, the following values are returned:
  - `mysql.remote.endpoint`: Name of the server connection endpoint.
  - `mysql.remote.connection_id`: Connection ID of the server connection.
  - `db.name`: Name of the schema.

`mysql/from_pool_or_connect`

Attributes:

- `mysql.remote.candidates`: Comma-separated list of endpoints.
- `net.peer.name`: Hostname of the endpoint this connection connected to in its previous session.
- `net.peer.port`: Port of the endpoint this connection connected to in its previous session.

`mysql/from_pool`	Attributes: • `mysql.error_message`: Displayed if `status_code` is `ERROR`. • `mysql.remote.connection_id`: Connection ID of the server connection.
`mysql/connect`	Attributes: • `net.peer.name`: Hostname of the endpoint. • `net.peer.port`: Port of the endpoint.
`mysql/authenticate`	Attributes: • `mysql.remote.needs_full_authentication`: Boolean. If a full handshake is required (`true`) or if a fast reset-connection is possible (`false`). If `true`, followed by `mysql/change_user`. If `false`, followed by `mysql/reset_connected`.
`mysql/server_greeting`	Attributes: • `mysql.remote.connection_id`: Connection ID of the server connection.
`mysql/client_greeting`	Attributes: • `db.name`: Name of the schema.
`mysql/tls_connect`	Attributes: • `tls.version`: TLS version in use. • `tls.cipher`: TLS cipher used for the connection. • `tls.session_resused`: Boolean. `True` if the TLS session was reused.
`mysql/response`	Attributes: • `mysql.session.@@SESSION.*`: Session variables changed according to the server session tracker. • `mysql.session.transaction_state`: Comma-separated list of transaction states. • `mysql.session.transaction_characteristics`: Statement required to restore the transaction state.
`mysql/set_var`	Attributes: • `mysql.session.@@SESSION.*`: Session variables restored after a reconnect.

The following events have the same attributes as `mysql/query`:

- `mysql/ping`

- `mysql/stmt_prepare`
- `mysql/stmt_execute`
- `mysql/kill`
- `mysql/statistics`
- `mysql/set_option`
- `mysql/reload`
- `mysql/list_fields`

The following events have no attributes:

- `mysql/prepare_server_connection`:
- `mysql/reset_connection`:
- `mysql/greeting`:
- `mysql/forward`:

Trace Examples

Simple Query Forwarding

The following example shows a trace of a simple forwarding of a query:

1. MySQL Router receives a query.
2. MySQL Router forwards the query to the server.
3. MySQL Router waits for the result.
4. MySQL Router forwards the result to the client.

```
$ mysql --host=127.0.0.1 --port=6446 --show-warnings
> ROUTER SET trace = 1;
> SELECT @@port;
+--------+
| @@port |
+--------+
|   3306 |
+--------+
1 row in set, 1 warning (0,02 sec)
Note (code 4600): {
  "start_time": "2023-03-23T15:31:08.052442Z",
  "end_time": "2023-03-23T15:31:08.052653Z",
  "elapsed_in_span_us": 211,
  "name": "mysql/query",
  "attributes": {
    "mysql.sharing_blocked": false
  },
  "events": [
    {
      "timestamp": "2023-03-23T15:31:08.052444Z",
      "name": "mysql/query_classify",
      "attributes": {
```

```
        "mysql.query.classification": "accept_session_state_from_session_tracker"
      }
    },
    {
      "start_time": "2023-03-23T15:31:08.052455Z",
      "end_time": "2023-03-23T15:31:08.052495Z",
      "elapsed_in_span_us": 39,
      "name": "mysql/connect_and_forward",
      "attributes": {
        "mysql.remote.is_connected": true,
        "mysql.remote.endpoint": "localhost:3306",
        "mysql.remote.connection_id": 17,
        "db.name": ""
      },
      "events": [
        {
          "start_time": "2023-03-23T15:31:08.052458Z",
          "end_time": "2023-03-23T15:31:08.052495Z",
          "elapsed_in_span_us": 36,
          "name": "mysql/forward"
        }
      ]
    },
    {
      "start_time": "2023-03-23T15:31:08.052623Z",
      "end_time": "2023-03-23T15:31:08.052627Z",
      "elapsed_in_span_us": 3,
      "name": "mysql/response",
      "attributes": {
        "mysql.session.@@SESSION.statement_id": "84"
      }
    }
  ]
}
```

Chapter 4 Configuration

Table of Contents

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

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

For command-line syntax related information and options, see Section 4.3.1, “Defining Options Using the Command Line”.

4.1 Configuration File Syntax

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

Note

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

Comments

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

Sections

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

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

[section name:optional section key]

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

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

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

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

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

Default Section

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

Options

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

```
name = value
```

Any leading or trailing blank characters on the option name or option value are removed before being handled. Option names are case-insensitive. Trailing comments are not supported, so in this example the option `routing_strategy` is given the value `round-robin # Circles back to first server` and generates an error when starting the router.

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

Variable Interpolation

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

Consider this sample configuration file:

```
[DEFAULT]
prefix = /usr/

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

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

Predefined variables

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

Table 4.1 Predefined variables

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

Command Line Related Details

For command-line syntax related information and options, see Section 4.3.1, “Defining Options Using the Command Line”.

4.2 Configuration 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
- User-Defined and Extra Configuration Files
- Default Configuration File Locations (Linux)
- Default Configuration File Locations (Windows)
- MySQL Router Configuration in Cluster Metadata

Default Configuration File Locations

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

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

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

```
$> mysqlrouter --help

...

Start MySQL Router.

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

```
  /usr/local/mysql-router/data
Default Runtime State Directory:
  /usr/local/mysql-router/run

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

...
```

Important

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

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

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

Note

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

2. `$HOME/.mysqlrouter.conf`

Note

For backward compatibility, MySQL Router also looks for the *.ini* variant in each directory. In doing so, Router looks in the initial directory for the *.conf* version, then checks for a *.ini* version, and then repeats the process in the next directory which is typically the user's home directory on the system.

User-Defined and Extra Configuration Files

Two command line options help control these configuration file locations:

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

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

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

For example:

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

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

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

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

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

Default Configuration File Locations (Linux)

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

Note

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

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

Default Configuration File Locations (Windows)

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

Note

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

- Default system-wide installation under `C:\ProgramData\MySQL\MySQL Router` : `C:\ProgramData\MySQL\MySQL Router\mysqlrouter.conf`
- In addition: `C:\Users\`*`username`*`\AppData\Roaming\mysqlrouter.conf` where *`username`* is replaced with your system's user.
- In addition to *mysqlrouter.conf*, for backwards compatibility the system also looks for *mysqlrouter.ini*
- With `--directory`: a bootstrapped standalone installation using `--directory` adds `mysqlrouter.conf` into the directory defined by *--directory*.

MySQL Router Configuration in Cluster Metadata

As of MySQL Router 8.4, the full configuration of routers bootstrapped against a InnoDB Cluster is stored in the InnoDB Cluster Metadata Schema and can be read by the MySQL Shell operation, *`object`*`.routerOptions`, for Cluster, ClusterSet, and ReplicaSets.

The configuration is stored per router as JSON in the `mysql_innodb_cluster_metadata.routers` table with one row per router. Each router's row is updated by the router on startup or restart.

For example:

```
        mysql>  select JSON_PRETTY(attributes->>'$.Configuration') as Configuration from mysql_innodb_clust

        Configuration: {
"io": {
    "backend": "poll",
    "threads": 0
},
"common": {
    "name": "system",
    "user": "",
    "read_timeout": 30,
    "client_ssl_key": "/Users/areliga/dev/server/mysql-trunk/build/pt/data/router-key.pem",
    "client_ssl_cert": "/Users/areliga/dev/server/mysql-trunk/build/pt/data/router-cert.pem",
    "client_ssl_mode": "PREFERRED",
```

```
    "connect_timeout": 5,
    "server_ssl_mode": "PREFERRED",
    "server_ssl_verify": "DISABLED",
    "max_total_connections": 512,
    "unknown_config_option": "error",
    "router_require_enforce": true,
    "max_idle_server_connections": 64
},
"loggers": {
    "filelog": {
    "level": "info",
    "filename": "mysqlrouter.log",
    "destination": "",
    "timestamp_precision": "second"
    }
},
"endpoints": {
    "bootstrap_ro": {

    ......

    }
   }
```

See Viewing Router Configurations with MySQL Shell for more information.

For backward compatibility, MySQL Router continues to store some configuration parameters in the `attributes` JSON in `mysql_innodb_cluster_metadata.routers`.

For example:

```
select JSON_PRETTY(attributes) from mysql_innodb_cluster_metadata.routers;
| {
  "ROEndpoint": "6447",
  "RWEndpoint": "6446",
  "ROXEndpoint": "6449",
  "RWXEndpoint": "6448",
  "RWSplitEndpoint": "6450",
  "MetadataUser": "mysql_router1_plje99d",
  "Configuration": { /*...*/ },
  "bootstrapTargetType": "cluster"
} |
```

4.3 Configuration Options

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

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

4.3.1 Defining Options Using the Command Line

Options can be configured and overridden at runtime using these different methods:

- Using standard runtime options as shown by `mysqlrouter --help`; how it affects the generated configuration file depends on the option. For example:

```
$> mysqlrouter --bootstrap foo@bar.com --connect-timeout=20
```

- Using the form `--section[:section_key].option_name=option_value` at runtime; this does not affect the generated configuration file. This is typically used for testing as using a configuration file is preferred. For example:

```
$> mysqlrouter -c mysqrouter.conf --logger.level=debug
```

- Using the `--conf-set-option=section[:section_key].option_name=option_value` option that does alter the generated configuration file. This is used while bootstrapping to add or override a configuration option. It has precedence over other forms.

```
$> mysqlrouter --bootstrap foo@bar.com \
  --conf-set-option=logger.level=debug \
  --conf-set-option=DEFAULT.unknown_config_option=warning \
  --conf-set-option=DEFAULT.connect_timeout=20 \
  --connect-timeout=10
```

  This sets `connect_timeout` to 20 in the generated `mysqlrouter.conf` because `--conf-set-option` always takes precedence.

4.3.2 MySQL Router Command Line Programs

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

4.3.2.1 mysqlrouter — Command Line Options

- mysqlrouter Option Summaries
- mysqlrouter Option Descriptions

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

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

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

mysqlrouter Option Summaries

Table 4.2 General Options

Option Name	Description
--conf-set-option	Sets a value for a generated configuration option during bootstrap
--config	Read configuration options from the provided file
--extra-config	Read this file after configuration files are read from either default locations or from files specified by the --config option
--help	Display help text and exit
--pid-file	Location to store the PID file

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

Table 4.3 Bootstrapping Options

Option Name	Description
--account	The MySQL user account used by Router after bootstrapping
--account-create	Bootstrapped account creation behavior
--account-host	The host pattern used for bootstrapped accounts
--bootstrap	Bootstrap and configure Router for operation with a MySQL InnoDB cluster
--bootstrap-socket	Connect to the MySQL metadata server through a Unix domain socket, used in conjunction with --bootstrap
--conf-base-port	Base port to use for listening Router ports
--conf-bind-address	IP address of the interface to which router's listening sockets should bind
--conf-skip-tcp	Whether to disable binding of a TCP port for incoming connections
--conf-target-cluster	Sets the target_cluster metadata option to a cluster type
--conf-target-cluster-by-name	Sets the target_cluster metadata option to a specific cluster name
--conf-use-gr-notifications	Enables Group Replication notifications
--conf-use-sockets	Whether to use Unix domain sockets
--connect-timeout	Number of seconds before connection attempts to a metadata server are considered timed out
--directory	Creates a self-contained directory for a new instance of the Router
--disable-rest	Disables generation of REST API configuration details into the generated mysqlrouter.conf file
--disable-rw-split	Disables generation of read-write splitting configuration details into the generated mysqlrouter.conf file
--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
--https-port	MySQL Router REST API HTTP server port
--master-key-reader	Script that returns the master key to STDOUT
--master-key-writer	Script that reads the master key from STDIN
--name	Gives a symbolic name for the router instance

Option Name	Description
--password-retries	The number of retries to use for generating the Router's user password
--read-timeout	Number of seconds before read operations to a metadata server are considered timed out
--report-host	Router's hostname; overrides auto-detection
--strict	Enables bootstrap strict mode

Table 4.4 SSL Options

Option Name	Description
--client-ssl-ca	The path to the Certificate Authority (CA) certificate file in PEM format
--client-ssl-capath	The path to the directory that contains the trusted SSL Certificate Authority (CA) certificate files in PEM format.
--client-ssl-cert	The path to the SSL public key certificate file, in PEM format, used to encrypt client-to-router connections
--client-ssl-cipher	Which ciphers are allowed between client and MySQL Router, defaults to a secure list of SSL ciphers
--client-ssl-crl	The path to the file containing the certificate revocation lists in PEM format
--client-ssl-crlpath	The path to the directory that contains the certificate revocation list files in PEM format
--client-ssl-curves	Which curves are allowed between the client and MySQL Router, defaults to a secure list of SSL curves
--client-ssl-dh-params	Filename of the DH parameter file. Not set by default
--client-ssl-key	The path name of the SSL private key file, in PEM format, used to encrypt client-to-router connections
--client-ssl-mode	Controls if connections from the client to MySQL Router must be encrypted, defaults to PREFERRED if not set
--server-ssl-ca	The path to the Certificate Authority (CA) certificate file in PEM format
--server-ssl-capath	The path to the directory that contains the trusted SSL Certificate Authority (CA) certificate files in PEM format.
--server-ssl-cipher	SSL Cipher for Server
--server-ssl-crl	The path to the file containing the certificate revocation lists in PEM format
--server-ssl-crlpath	The path to the directory that contains the certificate revocation list files in PEM format
--server-ssl-curves	SSL Curves for Server

Option Name	Description
--server-ssl-mode	Controls if connections from router to server must be encrypted.
--server-ssl-verify	Verification of the SSL certificates presented to the router by the server
--ssl-ca	Path to SSL Certificate Authority file to verify server's certificate against
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files
--ssl-cert	The client-side SSL certificate to facilitate client-side authentication during bootstrap
--ssl-cipher	A colon-separated list of SSL ciphers to allow, if SSL is enabled
--ssl-crl	Path to SSL CRL file to use when verifying server certificate
--ssl-crlpath	Path to directory containing SSL CRL files to use when verifying server certificate
--ssl-key	The private SSL key to facilitate client-side authentication during bootstrap
--ssl-mode	Desired security state when connecting to the metadata server during bootstrap and normal operation. Analogous to --ssl-mode in mysql client
--tls-version	Comma-separated list of TLS versions to request, if SSL is enabled

Table 4.5 Windows Services Options

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

mysqlrouter Option Descriptions

- `--version`, `-V`

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

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

```
$> mysqlrouter --version

MySQL Router v8.4.0 on Linux (64-bit) (GPL community edition)
```

- `--help`, `-?`

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:

```
$> mysqlrouter --help

...

Start MySQL Router.

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

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

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

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

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

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

Note

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

By default, the bootstrap process performs a system-wide configuration of MySQL Router. Only one instance of MySQL Router can be configured for system-wide operation. The system instance of MySQL

Router has a `router_name` of "system". If additional instances are desired, use the `--directory` option to create self-contained MySQL Router installations.

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

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

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

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

The generated configuration file is named `mysqlrouter.conf`, and its location depends on the type of instance being configured, the system, and the package. For system-wide installations, the generated configuration file is added to the system's configuration directory such as `/etc` or `%PROGRAMDATA%\MySQL\MySQL Router\`. Executing `mysqlrouter --help` will display this location.

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

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

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

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

Table 4.6 Conditions that affect default --bootstrap values

Condition	Description
`--conf-base-port`	Modifies generated `bind_port` values for each connection type. By default, generated `bind_port` values are as follows: For the classic protocol, Read-Write uses 6446 and Read-Only uses 6447, and for the X protocol Read-Write uses 6448 and Read-Only uses 6449.

Condition	Description
	Setting `--conf-base-port` to 0 changes the default `bind_port` values to the following defaults: For the classic protocol, Read-Write uses 6446 and Read-Only uses 6447, and for the X protocol Read-Write uses 64460 and Read-Only uses 64470.
`--conf-use-sockets`	Inserts `socket` definitions for each connection type.
`--conf-skip-tcp`	TCP/IP connection definitions are not defined.
`--directory`	Affects all file paths, and also generates additional files.
Other	This list is not exhaustive, other options and conditions also affect the generated values.

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

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

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

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

This option is not available on Windows.

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

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

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

The self-contained directory structure for Router is:

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

```
$path/log/mysqlrouter.log
```

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

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

- `--master-key-writer`

Command-Line Format	`--master-key-writer file_path`
Type	String

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

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

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

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

```
#!/bin/bash

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

Example usage:

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

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

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

- `--master-key-reader`

Command-Line Format	`--master-key-reader file_path`
Type	String

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

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

is not written to the `mysqlrouter.key` master key file, and instead uses the value provided by this option's script.

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

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

```
#!/bin/bash

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

Example usage:

```
$> mysqlrouter --bootstrap=127.0.0.1:3310 --master-key-reader=./reader.sh
  # Or, multiple hosts--master-key-writer=./writer.sh
```

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

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

- `--strict`

Command-Line Format	`--strict`
Type	String

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

- `--account`

Command-Line Format	`--account username`
Type	String

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

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

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

Bootstrapping without passing `--account` does not recreate an existing MySQL server account.

Using this option assumes the user has sufficient access rights for Router because the bootstrap process does not attempt to add missing grants to existing accounts. The bootstrap process does verify the permissions and outputs information to the console of the failed check. The bootstrap process

continues despite these failed checks unless the optional `--strict` option is also used. Example required permissions:

```
GRANT USAGE ON *.* TO `theuser`@`%`
GRANT SELECT, EXECUTE ON `mysql_innodb_cluster_metadata`.* TO `theuser`@`%`
GRANT INSERT, UPDATE, DELETE ON `mysql_innodb_cluster_metadata`.`routers` TO `theuser`@`%`
GRANT INSERT, UPDATE, DELETE ON `mysql_innodb_cluster_metadata`.`v2_routers` TO `theuser`@`%`
GRANT SELECT ON `performance_schema`.`global_variables` TO `theuser`@`%`
GRANT SELECT ON `performance_schema`.`replication_group_member_stats` TO `theuser`@`%`
GRANT SELECT ON `performance_schema`.`replication_group_members` TO `theuser`@`%`
```

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

- `--account-create`

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

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

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

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

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

- `--account-host`

Command-Line Format	`--account-host host_pattern`
Type	String

Default Value	%

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

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

Note

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

Note

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

Examples:

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

# Or, multiple hosts
$> mysqlrouter --bootstrap localhost:3310 --account-host host1 --account-host host2
  --account-host host3
```

- `--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 `protocol` option. With the classic protocol enabled, the file is named `mysql.sock` for read-write connections, and `mysqlro.sock` for read-only connections. With the X Protocol enabled, the file is named `mysqlx.sock` for read-write connections, and `mysqlxro.sock` for read-only connections.

This option is not available on Windows.

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

Command-Line Format	`--conf-use-gr-notifications`

Enables the `use_gr_notifications` [metadata_cache] option during bootstrap. When enabled, Router is asynchronously notified about most cluster changes. See `use_gr_notifications` for more information. In addition, using this option sets `ttl`=60 and `auth_cache_refresh_interval`=60.

- `--pid-file` *`path`*

Command-Line Format	`--pid-file path`

Type	String

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

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

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

- `--report-host`

Command-Line Format	`--report-host hostname`
Type	String

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

Router does not check or confirm that the supplied hostname is reachable, but does use RFC 1123 to validate host names, and RFC 2181 to validate addresses.

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

- `--conf-skip-tcp`

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

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

This option is not available on Windows.

- `--conf-base-port` *`port_num`*

Command-Line Format	`--conf-base-port port_num`
Type	Integer
Default Value	`0`

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.

Setting `--conf-base-port` to 0 changes the default `bind_port` values to the following defaults, which were as follows: For the classic protocol, Read-Write uses 6446 and Read-Only uses 6447, and for the X protocol Read-Write uses 64460 and Read-Only uses 64470.

Example usage:

```
# Example without --conf-base-port
$> 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:6448
- Read/Only Connections: localhost:6449

# Example demonstrating --conf-base-port set to 0
$> mysqlrouter --bootstrap root@localhost:3310 --conf-base-port 0
...
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
```

- `--conf-bind-address` *`address`*

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

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

Note

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

- `--read-timeout` *`num_seconds`*

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

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

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

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

- `--connect-timeout` *`num_seconds`*

Command-Line Format	`--connect-timeout num_seconds`
Type	Integer

Default Value	`30`

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

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

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

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

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

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

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

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

This option is not available on Windows.

- `--name router_name`

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

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

- `--force-password-validation`

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

By default, MySQL Router skips the MySQL Server's validate_password mechanism and instead Router generates and uses a STRONG password based on known validate_password default settings. This is because validate_password can be configured by the user and Router can not take into account unusual custom settings.

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

- `--password-retries num_retries`

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

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

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

- `--force`

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`
Type	String
Default Value	`PREFERRED`

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

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

During bootstrap, all connections to metadata servers made by the Router will use the SSL options specified. If `ssl_mode` is not specified in the configuration, it will default to PREFERRED. During normal operation, after Router is launched, its Metadata Cache plugin will read and honor all configured SSL settings.

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

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

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

- `--ssl-cert file_path`

Command-Line Format	`--ssl-cert file_path`
Type	String
Default Value	

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

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

- `--ssl-key file_path`

Command-Line Format	`--ssl-key file_path`
Type	String

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

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

- `--ssl-cipher ciphers`

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

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

- `--tls-version versions`

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

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

- `--ssl-ca file_path`

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

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

- `--ssl-capath dir_path`

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

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

- `--ssl-crl file_path`

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

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

- `--ssl-crlpath dir_path`

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

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

- `--client-ssl-mode mode`

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

Valid Values	`PREFERRED`
	`DISABLED`
	`PASSTHROUGH`
	`REQUIRED`

SSL connection mode for use during bootstrap and normal operation when connecting between MySQL Router and client.

During bootstrap, all connections to clients made by the Router will use the SSL options specified. If `client_ssl_mode` is not specified in the configuration, it defaults to `PREFERRED`.

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

- `--client-ssl-cert file_path`

Command-Line Format	`--client-ssl-cert file_path`
Type	String

Default Value	

The path name of the SSL public key certificate file in PEM format. This is used to facilitate client-side authentication during the bootstrap process.

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

- `client-ssl-curves`

Command-Line Format	`--client-ssl-curves`
Type	String

Defaults to a secure list of SSL curves. Format this string as a colon separated list of curve names.

- `--client-ssl-key` *`file_path`*

Command-Line Format	`--client-ssl-key file_path`
Type	String

The path name of the SSL private key file in PEM format. This is used to facilitate client-side authentication during the bootstrap process.

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

- `--client-ssl-cipher` *`ciphers`*

Command-Line Format	`--client-ssl-cipher`
Type	String

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

- `client-ssl-dh-params`

Command-Line Format	`--client-ssl-dh-params=filepath`
Type	String

Filename of the DH parameter file. If specified and not empty, the DH parameters from this file are used instead of internal default DH parameters. Format the DH param file in PEM format.

- `--client-ssl-ca` *`file_path`*

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

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

- `--client-ssl-capath dir_path`

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

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

- `--client-ssl-crl file_path`

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

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

- `--client-ssl-crlpath dir_path`

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

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

- `--server-ssl-mode mode`

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

Valid Values	`AS_CLIENT`
	`DISABLED`
	`PREFERRED`
	`REQUIRED`

SSL connection mode for use during bootstrap and normal operation when connecting between MySQL Router and server.

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

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

- `--server-ssl-cipher ciphers`

Command-Line Format	`--server-ssl-cipher`
Type	String

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

- `--server-ssl-ca file_path`

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

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

- `--server-ssl-capath dir_path`

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

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

- `--server-ssl-crl file_path`

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

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

- `--server-ssl-crlpath dir_path`

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

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

- `server-ssl-curves`

Command-Line Format	`--server-ssl-curves`
Type	String

Defaults to a secure list of SSL curves. Format this string as a colon separated list of curve names.

- `--server-ssl-verify string`

Command-Line Format	`--server-ssl-verify`
Type	String
Default Value	`DISABLED`

Valid Values	`DISABLED`
	`VERIFY_CA`
	`VERIFY_IDENTITY`

Verification of the SSL certificates presented to the router by the server

- `DISABLED`: the connection fails if the server does not provide a certificate in the handshake.
- `VERIFY_CA`: the connection fails if the server's certificate does not match a CA trusted by MySQL Router.
- `VERIFY_IDENTITY`: the connection fails if the server's certificate does not match a CA trusted by MySQL Router, or the server certificate's subject does not match the hostname or IP address MySQL Router connected to.

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

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

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

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

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

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

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

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

- `--install-service`

Command-Line Format	`--install-service [service_name]`
Platform Specific	Windows

Install Router as a Windows service that automatically starts when Windows starts. The service name defaults to *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 [service_name]`
Platform Specific	Windows

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

This option is only available on Windows.

- `--remove-service`

Command-Line Format	`--remove-service [service_name]`
Platform Specific	Windows

Remove the Router Windows service; service name defaults to MySQLRouter.

This option is only available on Windows.

- `--service`

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

Start Router as a Windows service. This is a private option, meaning it is only meant to be used by the Windows Service when launching Router as a 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.

- `--conf-target-cluster`

Command-Line Format	`--conf-target-cluster value`
Type	String

Valid Values	`current`
	
	`primary`

Sets the `target_cluster` metadata MySQL Router option. Accepts one of the following strings:

- `current`: sets `target_cluster` to the cluster containing the node being bootstrapped against. It defines it as the cluster's UUID value.

  If this is also the Primary, it does not dynamically follow role changes like the `primary` does; instead it remains static.

- `primary`: sets `target_cluster` to the primary cluster, including when it changes at runtime.

See also `--config-target-cluster-by-name`, which sets the `target_cluster` to a specific static cluster name.

Note

Bootstrapping against a ClusterSet requires the `cluster_type` Router configuration option set to *gr*.

- `--conf-set-option`

Command-Line Format	`--conf-set-option section_name[:optional_section_key].option=va`
Type	String

Sets a value for a generated configuration option during bootstrap; this can set a value for any bootstrapped option, for example:

```
$> mysqlrouter -B 127.0.0.1:5000 \
               --directory=dir1  \
               --conf-set-option=logger.level=debug \
               --conf-set-option=routing:test_rw.max_connect_errors=0 \
               --conf-set-option=routing:test_ro.max_connect_errors=0
```

Those commands alter the default values for those specific options by defining them as such:

```
[logger]
level=debug

[routing:test_rw]
...
max_connect_errors=0
...

[routing:test_ro]
...
max_connect_errors=0
...
```

`--conf-set-option` definitions take precedence over option specific parameters to set specific value. For example, if both `--connect-timeout=X` and `--conf-set-option=DEFAULT.connect_timeout=Y` are specified when bootstrapping, the `connect_timeout` is set to *Y* in the generated configuration file.

- `--conf-target-cluster-by-name`

Command-Line Format	`--conf-target-cluster-by-name clusterName`
Type	String

Sets the `target_cluster` metadata MySQL Router option to a specific cluster name.

Or, instead use `--conf-target-cluster` to assign a dynamic cluster type, such as primary.

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

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.

- `--disable-rw-split`

Command-Line Format	`--disable-rw-split`

By default, a `[router:read_write_split]` section is added to the generated `mysqlrouter.conf` at bootstrap; and this parameter means those details are not added. For additional information, see Bootstrapping.

- `--disable-rest`

Command-Line Format	`--disable-rest`

By default, configuration details for the MySQL Router REST API web service functionality are added to the generated `mysqlrouter.conf` file at bootstrap; and this parameter means those details are not added. This does not disable REST API functionality, as the REST API functionality can be manually configured (to enable it) later on.

- `--https-port`

Command-Line Format	`--https-port value`
Type	Integer
Default Value	`8443`
Minimum Value	`1`
Maximum Value	`65535`

Optionally define the HTTP server's `port` for the MySQL Router REST API under the [http_server] section in generated `mysqlrouter.conf` at bootstrap. It defaults to 8443. Availability of the port is not checked.

4.3.2.2 mysqlrouter_plugin_info — Command Line Options

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

Usage information:

```
$> ./mysqlrouter_plugin_info --help

Usage:
  ./mysqlrouter_plugin_info <mysqlrouter_plugin_file> <mysql_plugin_name>

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

To print help information:
  ./mysqlrouter_plugin_info --help
```

```
To print application version:
  ./mysqlrouter_plugin_info --version

$> ./bin/mysqlrouter_plugin_info --version

MySQLRouter Plugin Info App 8.0.3
```

Example usage:

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

4.3.2.3 mysqlrouter_passwd — Command Line Options

The `mysqlrouter_passwd` utility is a command line application to manage the accounts in the passwd file. For example usage, see Section 6.1, “A Simple MySQL Router REST API Guide”.

Usage information:

```
Usage
  bin/mysqlrouter_passwd [opts] <cmd> <filename> [<username>]
  bin/mysqlrouter_passwd --help
  bin/mysqlrouter_passwd --version

Commands

  delete
      Delete username (if it exists) from <filename>.
  list
      list one or all accounts of <filename>.
  set
      add or overwrite account of <username> in <filename>.
  verify
      verify if password matches <username>'s credentials in <filename>.

Options

  -?, --help
      Display this help and exit.
  --kdf <name>
      Key Derivation Function for 'set'. One of pbkdf2-sha256, pbkdf2-sha512,
      sha256-crypt, sha512-crypt. default: sha256-crypt
  -V, --version
      Display version information and exit.
  --work-factor <num>
      Work-factor hint for KDF if account is updated.
```

4.3.2.4 mysqlrouter_keyring — Command Line Options

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

Usage information:

Generic commands

- `--help`: usage information.

- `--version`: the tool's version.

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

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

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

- `list`: List usernames stored in the keyring; or list properties of a user stored in the keyring.
- `get`: Get property of user from the keyring.
- `export`: Export all entries of the keyring as JSON.
- `set`: Add or overwrite account of the user in the keyring file
- `delete`: Delete user from the keyring.

Master-key commands

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

Examples:

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

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

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

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

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

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

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

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

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

4.3.3 Configuration File Options

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

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

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

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

General Options

Table 4.7 [DEFAULT]

Option Name	Description	Type
`config_folder`	Path to configuration files	String
`connect_timeout`	Number of seconds before connection attempts to a metadata server are considered timed out	Integer
`core-file`	Write core file on Router crashes	Boolean
`event_source_name`	Microsoft Windows platforms only. Defines the service name used by MySQL Router when it is run as a service on Microsoft Windows.	String
`keyring_path`	Path to keyring file	String
`logging_folder`	Path to router logs	String
`master_key_path`	Path to master keyring file	String
`master-key-reader`	Script that returns the master key to STDOUT	String
`master-key-writer`	Script that reads the master key from STDIN	String
`max_total_connections`	Total maximum number of allowed client connections from the router	Integer
`pid_file`	Location to store the PID file	String
`plugin_folder`	Path to router plugins	String
`runtime_folder`	Path to runtime files	String
`sinks`	Logging method(s) to receive configured log data	String
`thread_stack_size`	Size in KB of memory allocated to each thread stack	Integer
`unknown_config_option`	Error type sent if an unknown configuration option is encountered	String

Option Name	Description	Type
user	System user MySQL Router is run as	String

Routing Options

Table 4.8 [routing]

Option Name	Description	Type
access_mode	Splits reads and writes according to the category of transaction.	String
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
client_ssl_ca	The path to the Certificate Authority (CA) certificate file in PEM format	String
client_ssl_capath	The path to the directory that contains the trusted SSL Certificate Authority (CA) certificate files in PEM format.	String
client_ssl_cert	The path to the SSL certificate (PEM) used to encrypt client-to-router communications	String
client_ssl_cipher	Which ciphers are allowed between client and MySQL Router, defaults to a secure list of SSL ciphers	String
client_ssl_crl	The path to the file containing the certificate revocation lists in PEM format	String
client_ssl_crlpath	The path to the directory that contains the certificate revocation list files in PEM format	String
client_ssl_curves	Which curves are allowed between the client and MySQL Router, defaults to a secure list of SSL curves	String
client_ssl_dh_params	Filename of the DH parameter file. Not set by default	String
client_ssl_key	The path to the SSL private key certificate file (PEM) used to encrypt client-to-router communications	String

Option Name	Description	Type
client_ssl_mode	Controls if connections from the client to MySQL Router must be encrypted, defaults to PREFERRED if not set	String
client_ssl_session_cache_mode	Enables or disables the TLS session cache for client connections	Boolean
client_ssl_session_cache_size	Number of entries in the TLS session cache for client connections	Integer
client_ssl_session_cache_timeout	Time in seconds until TLS sessions are removed from the client TLS session cache	Integer
connect_retry_timeout	Number of seconds MySQL Router waits before retrying a connection to a backend	Integer
connect_timeout	Number of seconds before connection attempts to a MySQL server are considered timed out	Integer
connection_sharing	Whether to enable connection sharing.	Integer
connection_sharing_delay	Seconds to wait before moving an idle connection to the connection pool.	Numeric
destinations	Routing destinations as either a comma-separated list of MySQL servers, or a metadata-cache definition	String
dynamic_state	Path to generated JSON file used to track and store active MySQL InnoDB Cluster Metadata server addresses	String
max_connect_errors	Maximum number of failed MySQL server connections before giving up	Integer
max_connections	Maximum number of connections assigned to a routed destination MySQL server	Integer
net_buffer_length	Set net_buffer_length	Integer
protocol	Protocol for connecting to MySQL Server	String
read_timeout	Number of seconds before read operations to a metadata server are considered timed out	Integer

Option Name	Description	Type
router_require_enforce	If enabled, retrieves the attributes for the current user and enforces them	Boolean
routing_strategy	Routing strategy (optional), how router chooses destination MySQL servers	String
server_ssl_ca	The path to the Certificate Authority (CA) certificate file in PEM format	String
server_ssl_capath	The path to the directory that contains the trusted SSL Certificate Authority (CA) certificate files in PEM format.	String
server_ssl_cert	The path to the SSL certificate (PEM) used to encrypt router-to-server communications	String
server_ssl_cipher	SSL Cipher for Server	String
server_ssl_crl	The path to the file containing the certificate revocation lists in PEM format	String
server_ssl_crlpath	The path to the directory that contains the certificate revocation list files in PEM format	String
server_ssl_curves	SSL Curves for Server	String
server_ssl_key	The path to the SSL private key certificate file (PEM) used to encrypt router-to-server communications	String
server_ssl_mode	Controls if connections from router to server must be encrypted	String
server_ssl_session_cache_mode	Enables or disables the TLS session cache for server connections	Boolean
server_ssl_session_cache_size	Number of entries in the TLS session cache for server connections	Integer
server_ssl_session_cache_timeout	Time in seconds until TLS sessions are removed from the server TLS session cache	Integer
server_ssl_verify	Verification of the SSL certificates presented to the router by the server	String
socket	Path to Unix domain socket file	String
wait_for_my_writes	Read-only queries wait for the last written transaction.	Integer

Option Name	Description	Type
wait_for_my_writes_timeout	Maximum time in seconds to wait for a read_only destination to apply the written transaction, before falling back to a read_write destination.	Integer

Destination Status Options

Table 4.9 [destination_status]

Option Name	Description	Type
error_quarantine_interval	Defines the interval, in seconds, between checks on quarantined destination connectivity. If a connection is possible, the destination is moved out of quarantine and made available for connections.	Integer
error_quarantine_threshold	Defines the threshold of consecutive, failed attempts to connect to a routing destination before MySQL Router adds the destination to quarantine and stops using it as a destination until it is cleared by the quarantine mechanism. For example, if set to 5, the destination is quarantined after 5 consecutive, failed attempts to connect to it.	Integer

Table 4.10 [connection_pool]

Option Name	Description	Type
idle_timeout	Seconds to keep the idling connection in the collection pool before closing it	Numeric
max_idle_server_connections	Connections to keep open after the client disconnects	Numeric

Metadata Cache Options

Table 4.11 [metadata_cache]

Option Name	Description	Type
auth_cache_refresh_interval	Time between auth-cache refresh attempts	Numeric
auth_cache_ttl	Time until the cache becomes invalid if not refreshed	Numeric
cluster_type	Object Router was bootstrapped against	String

Option Name	Description	Type
metadata_cluster	InnoDB Cluster name	String
router_id	Router ID	Integer
ssl_ca	SSL CA file to verify server's certificate against	String
ssl_capath	Directory containing SSL CA files to verify server's certificate against	String
ssl_crl	SSL CRL file to verify server's certificate against	String
ssl_crlpath	Directory containing SSL CRL files to verify server's certificate against	String
ssl_mode	SSL connection mode for connecting to the metadata server, defaults to PREFERRED if not set	String
tls_version	Comma-separated list of TLS versions to request, if SSL is enabled	String
ttl	Time To Live, in seconds	Integer
use_gr_notifications	Group Replication notifications behavior	Integer
user	MySQL user that accesses the MySQL Server's metadata schema	String

Logging Options

Table 4.12 [logger]

Option Name	Description	Type
destination	Name of device to log to; optionally used with [consolelog]	String
filename	Log file name; optionally used with [logger] and [filelog]	String
level	Logging level	String
timestamp_precision	Logger timestamp precision	String

HTTP Server Options

Table 4.13 [http_server]

Option Name	Description	Type
bind_address	IP address bound to the HTTP port	String
port	HTTP server TCP port	Integer
require_realm	[http_auth_realm] name	String
ssl_cert	SSL certification file name	String

Option Name	Description	Type
ssl_cipher	Approved SSL ciphers	String
ssl_dh_param	DH parameter file name	String
ssl	Enables TLSv1.2 or later support	Integer
ssl_key	SSL key filename	String
static_folder	Directory for HTTP server static file requests	String

Table 4.14 [http_auth_realm]

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

Table 4.15 [http_auth_backend]

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

Table 4.16 [io]

Option Name	Description	Type
backend	The IO backend	String
threads	The IO thread count	Numeric

MySQL Router Configuration File Option Descriptions

- access_mode

Type	String
Default Value	
Valid Values	auto

Defines how MySQL Router treats read-only and read-write queries. If enabled, read-only queries are directed to read-only servers, and read-write queries are directed to read-write servers. See Section 3.5, “Read/Write Splitting”.

- wait_for_my_writes

Type	Integer
Default Value	1
Minimum Value	0
Maximum Value	1

Read-only queries wait for the last written transaction.

See Section 3.5, “Read/Write Splitting”.

- `wait_for_my_writes_timeout`

Type	Integer
Default Value	`1`
Minimum Value	`0`
Maximum Value	`4294967295`

Maximum time in seconds to wait for a read_only destination to apply the written transaction, before falling back to a read_write destination.

See Section 3.5, “Read/Write Splitting”.

- `router_require_enforce`

Type	Boolean
Default Value	`0`

If enabled, MySQL Router retrieves the values defined in the user's `router_requires` attribute in the `USER_ATTRIBUTES` table.

The attribute must take the following format:

```
        {router_require: {value}}
```

The following are the possible values:

- `{}`: no requirements.
- `{ssl: true}`: MySQL Router requires SSL from the client.
- `{x509: true}`: MySQL Router requires SSL and an x509 certificate from the client.
- `{issuer: ""}`: MySQL Router requires SSL, an x509 certificate, and the certificate issuer from the client.
- `{ssl: true}`: MySQL Router requires SSL, an x509 certificate, and the certificate subject from the client.

- `--core-file`

Command-Line Format	`--core-file[={0|1}]`
Type	Boolean
Default Value	`0`

Write a core file if `mysqlrouter` dies. The name and location of the core file is system dependent. On Linux, a core file named `core.pid` is written to the current working directory of the process. *`pid`* represents the process ID of the server process. On macOS, a core file named `core.pid` is written to the `/cores` directory, if the process has the `com.apple.security.get-task-allow` entitlement.

On Solaris, use the `coreadm` command to specify where to write the core file and how to name it. On Windows, a minidump file named `mysqlrouter.{pid}.dmp` is written to the current working directory of the process.

- `event_source_name`

Type	String
Default Value	

Microsoft Windows platforms only. Defines the service name used by MySQL Router when it is run as a service on Microsoft Windows. This enables you to differentiate between services when running multiple instances of MySQL Router and between their messages in the Event Log.

For example:

```
[DEFAULT]
event_source_name = MySQLRouterService
```

- `logging_folder`

Type	String
Default Value	`$router_basepath`

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

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

Note

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

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

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

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

- `plugin_folder`

Type	String
Default Value (Windows)	
Default Value (Other)	`/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`

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

Path to the MySQL Router runtime files.

Default value: `/run/mysqlrouter`

- `master-key-writer`

Command-Line Format	`--master-key-writer file_path`
Type	String

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

- `master-key-reader`

Command-Line Format	`--master-key-reader file_path`
Type	String

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

- `config_folder`

Type	String
Default Value (Windows)	
Default Value (Other)	`/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`

- `sinks`

Type	String

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

Valid Values (Other)	`consolelog` `filelog`

	`syslog`

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

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

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

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

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

- `keyring_path`

Type	String
Default Value (Windows)	`%PROGRAMDATA%\MySQL\MySQL Router \keyring-data`
Default Value (Other)	`/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
```

- `master_key_path`

Type	String
Default Value (Windows)	`%PROGRAMDATA%\MySQL\MySQL Router \mysqlrouter.key`
Default Value (Other)	`/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
```

- `unknown_config_option`

Type	String
Default Value	`warning`

Valid Values	`warning`
	
	`error`

Determines MySQL Router behavior for handling unknown configuration options, such as typos.

A *warning* is default behavior, and bootstrapping defines it as *error* in the generated configuration file. Warning logs a warning message but does not halt, whereas an error means MySQL Router fails to initialize and exits.

```
[DEFAULT]
unknown_config_option=warning
```

- `user (system)`

Type	String

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

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

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

MySQL Router can be bootstrapped and executed under any Operating System user and does not require special privileges other than read and write access to its own files. The files it accesses include plugins (read/execute), configuration file, logs, UNIX domain socket files (if enabled), and more.

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

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

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

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

Note

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

- `ssl_ca`

Type	String

Path to the SSL CA file to verify server's certificate against when connecting to the metadata servers.

Can optionally be set with the `--ssl-ca` bootstrap option.

- `ssl_capath`

Type	String

Path to directory containing SSL CA files to verify server's certificate against when connecting to the metadata servers.

Can optionally be set with the `--ssl-capath` bootstrap option.

- `ssl_crl`

Type	String

Path to SSL CRL file to use when connecting to metadata servers and verifying their SSL certificate.

Can optionally be set with the `--ssl-crl` bootstrap option.

- `ssl_crlpath`

Type	String

Path to directory containing SSL CRL files to use when connecting to metadata servers and verifying their SSL certificate.

Can optionally be set with the `--ssl-crlpath` bootstrap option.

- `tls_version`

Type	String

Comma-separated list of TLS versions to request, such as 'TLSv1.2,TLSv1.3', if SSL is enabled.

Can optionally be set with the `--tls-version` bootstrap option.

- `bind_address`

Type	String

Default Value	`127.0.0.1`

Information related to the optional `bind_address` option:

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

Example usage:

```
bind_address = 127.0.0.1:7001
```

Note

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

- `bind_port`

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.

Optionally set these values by using the `--conf-base-port` bootstrap option.

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	Linux
Type	String

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

```
[routing]
socket = /tmp/mysqlrouter.sock
```

```
destinations = a.example.com:3306,b.example.com:3307
```

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

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

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

```
[routing:my_redirect]
bind_address = localhost:1234
socket = /tmp/mysqlrouter.sock
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`

Type	String
Default Value	`classic`

Valid Values	`classic`
	
	`x`

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

Example usage:

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

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

- `pid_file`

Type	String

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

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

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

- `connect_timeout`

Type	Integer
Default Value	`5`
Minimum Value	`1`
Maximum Value	`65536`

Timeout value used by the MySQL Router when connecting to the destination MySQL server. 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.

Example usage:

```
[routing]
connect_timeout = 5
```

Can be set at bootstrap using `--conf-set-option=routing.connect_timeout`.

- `connect_timeout`

Type	Integer
Default Value	`5`

Timeout value used by the MySQL Router when connecting to the MySQL metadata server.

Example usage:

```
[DEFAULT]
connect_timeout = 5
```

Can be set at bootstrap using either `--connect-timeout` or `--conf-set-option=DEFAULT.connect_timeout`.

- `read_timeout`

Type	Integer
Default Value	`30`

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

Example usage:

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

- `destinations`

Type	String

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

Example usage with specific hosts (static routing):

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

Note

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

Example usage with InnoDB cluster metadata cache:

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

The `metadata-cache` URI options are:

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

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

- `disconnect_on_promoted_to_primary`: Controls whether existing client connections to a secondary are closed when the secondary is promoted as a primary. The default value is "no", meaning existing client connections to the promoted secondary are not closed after promotion. Set *disconnect_on_promoted_to_primary=yes* in the URI to close these existing connections.

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

Note

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

- `dynamic_state`

Type	String

This option tracks and stores active MySQL InnoDB Cluster Metadata server addresses and loads them if Router is restarted. This functionality is activated by `--bootstrap`.

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

Example `mysqlrouter.conf` entry:

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

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

```
{
    "metadata-cache": {
        "group-replication-id": "4b9e817a-0254-11e9-9cc0-080027bb5030",
```

```
        "cluster-metadata-servers": [
            "mysql://localhost:3310",
            "mysql://localhost:3320",
            "mysql://localhost:3330"
        ]
    },
    "version": "1.0.0"
}
```

- `routing_strategy`

Type	String

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

	`round-robin-with-fallback`

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

Available strategies:

Note

The role documentation following this section describes the available `role` and `routing_strategy` combinations and conflicts.

Unreachable destinations are quarantined and skipped, and are probed for availability every `error_quarantine_interval` seconds. All routing strategies except for `next-available` utilize this behavior.

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

  Limitations include:

  - After all nodes of the selection are discarded, there is no way to add servers back to the list.

    Unlike other strategies, unreachable destinations are not probed for availability every `error_quarantine_interval` seconds.
  - After restarting MySQL Router, all knowledge of what servers are discarded is lost and all servers are available again.
  - Metadata cache does not support the next-available routing policy, as next-available only functions with static routing.

The `role` defaults and available combinations:

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

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

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

- `max_connections`

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

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

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

```
[routing:mycluster_default_rw]
max_connections = 512
```

Alternatively, use the newer `max_total_connections` configuration option that sets one value for all Router sections combined.

The maximum depends both on the system's poll (or linux_epoll) limitations and the number of available CPU cores/threads. See also the [IO] `backend` and `threads` configuration options.

Optionally setting `max_connections` in the `[DEFAULT]` section sets the default value for each routing destination.

- `max_total_connections`

Type	Integer
Default Value	`512`
Minimum Value	`1`
Maximum Value	`9223372036854775807`

The maximum number of client connections handled by Router, to help prevent running out of the file descriptors.

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

```
[DEFAULT]
max_total_connections = 512
```

Note

The legacy `max_connections` option sets a value per routing instance, such as one value for read-only, and another for write-only. The `max_total_connections` option sets one value for all routing instances combined.

The default value is 512, and it's set under the `[DEFAULT]` section.

- `thread_stack_size`

Type	Integer
Default Value	`64`
Minimum Value	`1`
Maximum Value	`65535`

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

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

- `net_buffer_length`

Type	Integer

Sets the `net_buffer_length` MySQL server option.

- `max_connect_errors`

Type	Integer
Default Value	`100`
Minimum Value	`1`
Maximum Value	`4294967295`

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

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

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

A successful connection resets the error counter.

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`

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

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

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

```
client_connect_timeout = 9
```

- `auth_cache_refresh_interval`

Type	Numeric
Default Value	`2`
Minimum Value	`0.001`
Maximum Value	`3600`

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

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

- `auth_cache_ttl`

Type	Numeric
Default Value	`-1`
Minimum Value	`0.001`
Maximum Value	`3600`

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

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

- `router_id`

Type	Integer
Maximum Value	`4294967295`

The MySQL Router ID.

- `server_ssl_cert`

Type	String
Default Value	

The path name of the SSL public key certificate file in PEM format. This is used to facilitate server-side authentication during the bootstrap process.

- `server_ssl_key`

Type	String
Default Value	

The path name of the SSL private key file in PEM format used to encrypt router-to-server connections. See also Section 4.4, “TLS Configuration” .

- `server_ssl_curves`

Type	String

Defaults to a secure list of SSL curves. Format this string as a colon separated list of curve names.

- `server_ssl_cipher`

Type	String

Defaults to a secure list of SSL ciphers. Format this string as a colon separated list of cipher names.

- `server_ssl_verify`

Type	String
Default Value	`DISABLED`

Valid Values	`DISABLED`
	`VERIFY_CA`
	`VERIFY_IDENTITY`

Verification of the SSL certificates presented to the router by the server.

- `DISABLED`: the connection fails if the server does not provide a certificate in the handshake.
- `VERIFY_CA`: the connection fails if the server's certificate does not match a CA trusted by MySQL Router.
- `VERIFY_IDENTITY`: the connection fails if the server's certificate does not match a CA trusted by MySQL Router, or the server certificate's subject does not match the hostname or IP address MySQL Router connected to.

- `server_ssl_mode`

Type	String
Default Value	`AS_CLIENT`

Valid Values	`AS_CLIENT`
	`DISABLED`
	`PREFERRED`
	`REQUIRED`

SSL connection mode to use when connecting between MySQL Router and server. See also Section 4.4, “TLS Configuration” .

- `server_ssl_ca`

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

The path name of the Certificate Authority (CA) certificate file in PEM format. The file contains a list of trusted SSL Certificate Authorities. See also Section 4.4, “TLS Configuration” .

- `server_ssl_capath`

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

The path name of the directory that contains trusted SSL Certificate Authority (CA) certificate files in PEM format. See also Section 4.4, “TLS Configuration” .

- `client_ssl_cert`

Command-Line Format	`--client-ssl-cert file_path`
Type	String
Default Value	

The path name of the SSL public key certificate file in PEM format. This is used to facilitate client-side authentication during the bootstrap process.

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

- `server_ssl_crlpath`

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

The path of the directory that contains certificate revocation-list files in PEM format. See also Section 4.4, “TLS Configuration” .

- `server_ssl_crl`

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

The path name of the file containing certificate revocation lists in PEM format. See also Section 4.4, “TLS Configuration” .

- `client_ssl_key`

Command-Line Format	`--client-ssl-key file_path`
Type	String
Default Value	

The path name of the SSL private key file in PEM format used to encrypt client-to-router connections. See also Section 4.4, “TLS Configuration” .

- `client_ssl_dh_params`

Type	String

Filename of the DH parameter file. If specified and not empty, the DH parameters from this file are used instead of internal default DH parameters. Format the DH param file in PEM format.

- `client_ssl_curves`

Type	String

Which curves are allowed between the client and MySQL Router, defaults to a secure list of SSL curves. Format this string as a colon separated list of curve names.

- `client_ssl_cipher`

Type	String

Which ciphers are allowed between client and MySQL Router, defaults to a secure list of SSL ciphers. Format this string as a colon separated list of cipher names.

- `client_ssl_mode`

Type	String
Default Value	`PREFERRED`

Valid Values	`PREFERRED`
	`DISABLED`
	`PASSTHROUGH`
	`REQUIRED`

Controls if connections from the client to MySQL Router must be encrypted. See also Section 4.4, “TLS Configuration” .

- `ssl_mode`

Type	String
Default Value	`PREFERRED`

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

	VERIFY_IDENTITY

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

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

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

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

- user (MySQL)

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_{router_id}_[0-9a-z]{7}*, where *{router_id}* is the numeric router id and *[0-9a-z]{7}* is 7 random lowercase alphanumeric characters. The

router id is reused if already present in `mysqlrouter.conf` and its value can not exceed 4294967295 (2^32-1).

Note

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

This structure changed in MySQL Router 8.4.1, previously it was *mysql_router_`[0-9]{1,6}`_`[0-9a-z]{12}`*.

- `metadata_cluster`

Type	String

Name of the InnoDB Cluster.

Note

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

- `use_gr_notifications`

Type	Integer
Default Value	`0`
Valid Values	`0` `1`

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

When Router receives any of the following notifications from Group Replication, it refreshes the cluster metadata:

- group_replication/membership/quorum_loss
- group_replication/membership/view
- group_replication/status/role_change
- group_replication/status/state_change

Note

The Group Replication notifications feature requires an X Protocol connection from Router to each instance, which must be running X Plugin. If an X Protocol

> connection is not available, the metadata refresh is carried out at `ttl` intervals as though the notifications feature was not enabled.
>
> Although the Group Replication notifications rely on an X Protocol connection, received notifications trigger a metadata refresh which uses a classic MySQL protocol connection to the instance.

When enabled, the Group Replication notification feature allows a higher `ttl` value because the metadata refreshes carried out at `ttl` intervals become an additional safeguard, rather than the primary means of keeping the information about the cluster state up to date. When disabled, a low `ttl` value (such as 0.5s, the default) is recommended to avoid the overhead of reconnecting to the instances and querying them for metadata changes often.

- `ttl`

Type	Numeric
Default Value	`0.5`
Minimum Value	`0`
Maximum Value	`3600`

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

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

The value must be smaller than `auth_cache_refresh_interval` and `auth_cache_ttl` else Router won't start.

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

- `destination`

Type	String
Default Value (Windows)	`CON`
Default Value (Other)	`/dev/stderr`
Valid Values (Windows)	`CON` `NUL`

Valid Values (Other)	`/dev/null`
	`/dev/stderr`
	`/dev/stdout`

Direct console log output to this device destination; set under the [consolelog] section. Defaults to */dev/stderr* and an empty value uses the default.

Available values are: `/dev/stdout`, `/dev/stderr`, and `/dev/null`; or `CON` and `NUL` on Windows.

```
[DEFAULT]
logging_folder=
```

```
[consolelog]
destination=/dev/null
```

- `filename`

Type	String

Redirect log output to a specific file named `filename` that resides in the `logging_folder` directory. It must be defined as a file name and not a file path, and works with both the [logger] and [filelog] sections.

Using `filename` with [logger] to define the default value for the [filelog] section, and it also changes Router's log file from `mysqlrouter.log` to this new value.

```
[DEFAULT]
logging_folder=/path/to/logs/

[logger]
filename = router_error.log
```

Router does not report an error if filename is set under [logger] but no file-based logger is used.

Using `filename` with [filelog]:

```
[DEFAULT]
logging_folder=/path/to/logs/

[filelog:a]
filename = a_router_error.log

[filelog:b]
filename = b_router_error.log
```

If filename is empty or not set under [filelog] then the filename definition under [logger] is used; and the default log file is used (`mysqlrouter.log`) if filename is not set under [logger] either.

Related, directing console output to `/dev/null`:

```
[DEFAULT]
logging_folder=

[consolelog]
destination=/dev/null
```

- `level`

Type	String
Default Value	`INFO`

Valid Values	`DEBUG`
	`NOTE`
	`INFO`
	`WARNING`
	`ERROR`
	`SYSTEM`

	`FATAL`

Use the *logger* plugin to log notices, errors, and debugging information. The available log levels are *DEBUG*, *NOTE*, *INFO* (default), *WARNING*, *ERROR*, *SYSTEM*, 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. *SYSTEM* includes messages such as startup messages.

```
[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.

Bootstrapping accepts a configuration file using `--config` and utilizes the logger level definition.

- `timestamp_precision`

Type	String

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

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

- `port`

Type	Integer
Default Value	`8081`

The TCP port listening for HTTP requests; it defaults to 8081.

- `bind_address`

Type	String
Default Value	`0.0.0.0`

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

- `static_folder`

Type	String

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

- `require_realm`

Type	String

Name of the [http_auth_realm] instance.

- `ssl`

Type	Integer
Default Value	`1`
Valid Values	`1` `0`

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

- `ssl_cert`

Type	String

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

- `ssl_key`

Type	String

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

- `ssl_cipher`

Type	String

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

- `ssl_dh_param`

Type	String

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

- `backend`

Type	String
Default Value (Windows)	`poll`
Default Value (Other)	`linux_epoll`
Valid Values (Windows)	`poll`
Valid Values (Other)	`linux_epoll` `poll`

The IO backend that handles async operations. The generic poll backend is available on all platforms, while each platform may provide alternative backends.

Options are `poll` (all platforms) and `linux_epoll` (Linux). Defaults to `linux_epoll` on Linux.

```
[io]
backend=linux_epoll
threads=32
```

Note

This is one of several `backend` options, each in a different `[section]` with a different purpose:

- `[io] backend` for async operations.
- `[http_auth_realm] backend` defines a custom name for a backend associated with a particular realm
- `[http_auth_backend] backend` type of auth backend

- `threads`

Type	Numeric
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1024`

The number of IO threads that handles connections.

Defaults to 0 (uses all available CPU cores/threads) but also accepts a number between 1 and 1024. At runtime the system may restrict the upper limit beyond this value.

```
[io]
backend=linux_epoll
threads=32
```

- `connection_sharing_delay`

Type	Numeric
Default Value	`1`
Minimum Value	`0`
Maximum Value	`2^63-1`

Seconds to wait before an idle server connection is available for reuse by another client connection.

See Section 3.4, “Connection Sharing and Reuse”.

- `connection_sharing`

Type	Integer
Default Value	`0`
Minimum Value	`0`

Maximum Value	`1`

Whether to enable connection sharing.

See Section 3.4, “Connection Sharing and Reuse”.

- `idle_timeout`

Type	Numeric
Default Value	`5`
Minimum Value	`1`
Maximum Value	`4294967296`

Seconds to keep the idling connection in the connection pool before closing it. This is set in the [connection_pool] section, and affects all routes in the connection pool. Defaults to 5, accepts a value between 1 and 4294967296.

- `max_idle_server_connections`

Type	Numeric
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967296`

Connections to keep open in the connection pool after the client disconnects; and is set in the `[connection_pool]` section. The default is 0, which disables connection pooling.

- `client_ssl_session_cache_mode`

Type	Boolean
Default Value	`1`

Enables or disables the cache for client-router TLS sessions.

Note

Enabled by default. If this parameter is not set, the cache is enabled. To disable the cache, you must explicitly define it.

- `client_ssl_session_cache_size`

Type	Integer
Default Value	`1024`
Minimum Value	`1`
Maximum Value	`2^31-1`

Defines the maximum number of sessions cached. If adding a new session to the cache causes the number of cached sessions to exceed the defined maximum, the oldest cached session is dropped to allow the newest to be cached.

- `client_ssl_session_cache_timeout`

Type	Integer
Default Value	`300`
Minimum Value	`1`
Maximum Value	`84600`

Defines the maximum amount of time, in seconds, a session remains in the cache. If the timeout is reached, and this season is not reused, the session is removed from the cache and the connection is closed.

- `server_ssl_session_cache_mode`

Type	Boolean
Default Value	`1`

Enables or disables the cache for router-server TLS sessions.

Note

Enabled by default. If this parameter is not set, the cache is enabled. To disable the cache, you must explicitly define it.

- `server_ssl_session_cache_size`

Type	Integer
Default Value	`1024`
Minimum Value	`1`
Maximum Value	`2^31-1`

Defines the maximum number of sessions cached. If adding a new session to the cache causes the number of cached sessions to exceed the defined maximum, the oldest cached session is dropped to allow the newest to be cached.

- `server_ssl_session_cache_timeout`

Type	Integer
Default Value	`300`
Minimum Value	`1`
Maximum Value	`84600`

Time in seconds until TLS sessions are removed from the server TLS session cache.

- `connect_retry_timeout`

Type	Integer
Default Value	`7`
Minimum Value	`1`

Maximum Value	`3600`

If a classic connection fails with a transient error, such as `max-connections reached`, MySQL Router waits the defined number of seconds before retrying the connection. The connection is retried according to the defined routing strategy.

If `connect_retry_timeout` is not defined, it defaults to 7 seconds. If the value of `connect_retry_timeout` is defined outside of the valid range of values, MySQL Router will fail to start.

Note

If connection sharing is enabled, the retried connection is to the same server as the initial connection attempt.

If a connection fails with a transient error after authentication has occurred, the connection can only be retried if the client-router connection is TLS encrypted or has a public key.

Table 4.17 SSL Modes and Retry

client_ssl_mode	server_ssl_mode	Supports Retry
PASSTHROUGH	Any	No
DISABLED	Any	No
PREFERRED	AS_CLIENT	No
PREFERRED	Any other mode	Yes
REQUIRED	Any	Yes

- `backend`

Type	String

Name of the `[http_auth_backend]` section.

Note

This is one of several `backend` options, each in a different `[section]` with a different purpose:

- `[io] backend` for async operations.
- `[http_auth_realm] backend` defines a custom name for a backend associated with a particular realm
- `[http_auth_backend] backend` type of auth backend

- `method`

Type	String
Default Value	`basic`

The HTTP authentication method; defaults to basic.

- `name`

Type	String

Name of the realm presented to the authentication user.

- `require`

Type	String
Default Value	`valid-user`

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

- `backend`

Type	String
Default Value	`file`

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

```
[http_auth_backend:name]
backend=metadata_cache

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

Note

This is one of several `backend` options, each in a different `[section]` with a different purpose:

- `[io] backend` for async operations.
- `[http_auth_realm] backend` defines a custom name for a backend associated with a particular realm
- `[http_auth_backend] backend` type of auth backend

- `filename`

Type	String

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

- `cluster_type`

Type	String
Valid Values	`gr`

	`rs`

The type of AdminAPI object that the Router was bootstrapped against, which is either an InnoDB ReplicaSet (rs) or InnoDB Cluster (gr). Use 'gr' for cluster sets.

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

- `error_quarantine_interval`

Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`65535`

Defines the interval, in seconds, between checks on quarantined destination connectivity. If a connection is possible, the destination is moved out of quarantine and made available for connections.

If an invalid value is defined, MySQL Router fails to start and an error is logged.

For example:

```
[destination_status]
 error_quarantine_threshold=5
 error_quarantine_interval=20
```

Note

If undefined in the configuration file, the default value, 1, is used.

- `error_quarantine_threshold`

Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`3600`

Defines the threshold of consecutive, failed attempts to connect to a routing destination before MySQL Router adds the destination to quarantine and stops using it as a destination until it is cleared by the quarantine mechanism. For example, if set to 5, the destination is quarantined after 5 consecutive, failed attempts to connect to it.

If an invalid value is defined, MySQL Router fails to start and an error is logged.

For example:

```
[destination_status]
 error_quarantine_threshold=5
 error_quarantine_interval=20
```

Note

If undefined in the configuration file, the default value, 1, is used.

4.3.4 Configuration File Example

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

In this example, read-write (primary) traffic is sent to port 6446 (classic) or 6448 (X Protocol), and read-only (secondaries) are accessed using port 6447 (classic) or 6449 (X Protocol).

The routing section keys (such as *myCluster_rw*) are optional but descriptive section keys help while debugging and also allows multiple configuration sections for the same plugin.

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

The options starting with `[http_server]` reference the REST API that is enabled by default. For additional details, see Chapter 6, *MySQL Router REST API*

```
# File automatically generated during MySQL Router bootstrap
[DEFAULT]
logging_folder=/tmp/router/log
runtime_folder=/tmp/router/run
data_folder=/tmp/router/data
keyring_path=/tmp/router/data/keyring
master_key_path=/tmp/router/mysqlrouter.key
connect_timeout=15
read_timeout=30
dynamic_state=/tmp/router/data/state.json
client_ssl_cert=/tmp/router/data/router-cert.pem
client_ssl_key=/tmp/router/data/router-key.pem
client_ssl_mode=PREFERRED
server_ssl_mode=AS_CLIENT
server_ssl_verify=DISABLED

[logger]
level = INFO

[metadata_cache:myCluster]
cluster_type=gr
router_id=1
user=mysql_router1_x9v4uk10nbcd
metadata_cluster=myCluster
ttl=0.5
auth_cache_ttl=-1
auth_cache_refresh_interval=2
use_gr_notifications=0

[routing:myCluster_rw]
bind_address=0.0.0.0
bind_port=6446
destinations=metadata-cache://myCluster/?role=PRIMARY
routing_strategy=first-available
protocol=classic

[routing:myCluster_ro]
```

```
bind_address=0.0.0.0
bind_port=6447
destinations=metadata-cache://myCluster/?role=SECONDARY
routing_strategy=round-robin-with-fallback
protocol=classic

[routing:myCluster_x_rw]
bind_address=0.0.0.0
bind_port=6448
destinations=metadata-cache://myCluster/?role=PRIMARY
routing_strategy=first-available
protocol=x

[routing:myCluster_x_ro]
bind_address=0.0.0.0
bind_port=6449
destinations=metadata-cache://myCluster/?role=SECONDARY
routing_strategy=round-robin-with-fallback
protocol=x

[http_server]
port=8443
ssl=1
ssl_cert=/tmp/router/data/router-cert.pem
ssl_key=/tmp/router/data/router-key.pem

[http_auth_realm:default_auth_realm]
backend=default_auth_backend
method=basic
name=default_realm

[rest_router]
require_realm=default_auth_realm

[rest_api]

[http_auth_backend:default_auth_backend]
backend=metadata_cache

[rest_routing]
require_realm=default_auth_realm

[rest_metadata_cache]
require_realm=default_auth_realm
```

4.4 TLS Configuration

Important

This section is a draft and subject to change.

The default behavior is:

```
client_ssl_mode = PREFERRED
server_ssl_mode = AS_CLIENT
```

This establishes TLS connections between the client and Router if the client desires switching to TLS and the server supports TLS. This also matches the existing behavior for client and server without the Router in-between.

TLS Endpoint Configuration

MySQL Router accepts the TLS session and opens a new TLS session to the server. For example:

```
client <-> router              // TCP
          router <-> server    // TCP
client <-> router              // TLS
          router <-> server    // TLS
```

To accept a TLS session from a client, Router has to present a TLS client with the certificate using `client_ssl_cert` and `client_ssl_key`.

To connect a TLS session to a server, Router verifies the server's certificates using `server_ssl_verify` `server_ssl_verify` `server_ssl_ca` `server_ssl_capath` `server_ssl_crl`, and `server_ssl_crlpath`.

Note

The TLSv1 and TLSv1.1 connection protocols are deprecated as of MySQL Router 8.0.26 and support for them is subject to removal in a future version.

SSL Modes

Because there are two TLS sessions (between client and Router; Router and server) there can also be two independent states of the connection.

Both `client_ssl_mode` and `server_ssl_mode` accept DISABLED, PREFERRED, or REQUIRED. In addition, `server_ssl_mode` accepts AS_CLIENT, and `client_ssl_mode` accepts PASSTHROUGH.

- `DISABLED`: Router does not offer encryption to the client, and the client can't switch the client-router connection to TLS. The client may abort the connection if it must switch to TLS.
- `PREFERRED` (default): Router accepts a TLS connection from the client, but is also okay if the client does not switch to encryption.
- `REQUIRED`: Router accepts a TLS connection from the client, and will fail if the connection is not switched to TLS before authentication finishes.
- `PASSTHROUGH`: Means 'forward everything to the server' and lets the client and server decide if they want to switch to TLS or not. This was default behavior before Router 8.0.23, and is only accepted by `client_ssl_mode`.
- `AS_CLIENT` (default): if the client-router connection is encrypted then also encrypt the router-server connection, otherwise do not. This option is only accepted by `server_ssl_mode`.

Additional Related Options

The `server_ssl_verify` option splits out 'VERIFY_CA' and 'VERIFY_IDENTITY' from the 'ssl_mode' that is known from the MySQL client and MySQL server. In the MySQL client's case, VERIFY_CA means ssl_mode=REQUIRED and to verify the CA|IDENTITY. In Router's case, Router verifies certificates independent of `server_ssl_mode`; instead it's purely based on whether the connection is encrypted and if `server_ssl_verify` is not DISABLED, in which case it is verified.

Additional options include `server_ssl_dh_params`, `client_ssl_dh_params`, `server_ssl_dh_params`, `client_ssl_dh_params`, `server_ssl_curves`, and `client_ssl_curves`.

All routing options and additional information is available at Routing Options.

Chapter 5 MySQL Router Application

Table of Contents

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

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

5.1 Starting MySQL Router

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

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

For example, when using **systemd**:

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

Example Log Output

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

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

$> less /path/to/log/mysqlrouter.log
2019-04-07 16:30:49 INFO    [0x7000022fc000] [routing:devCluster_default_ro] started: listening on 0.0.0.0
2019-04-07 16:30:49 INFO    [0x70000237f000] [routing:devCluster_default_rw] started: listening on 0.0.0.0
2019-04-07 16:30:49 INFO    [0x700002402000] [routing:devCluster_default_x_ro] started: listening on 0.0.0
2019-04-07 16:30:49 INFO    [0x700002485000] [routing:devCluster_default_x_rw] started: listening on 0.0.0
2019-04-07 16:30:49 INFO    [0x700002279000] Starting Metadata Cache
2019-04-07 16:30:49 INFO    [0x700002279000] Connections using ssl_mode 'PREFERRED'
2019-04-07 16:30:49 INFO    [0x700002279000] Connected with metadata server running on 127.0.0.1:3310
2019-04-07 16:30:49 INFO    [0x700002279000] Changes detected in cluster 'devCluster' after metadata refres
2019-04-07 16:30:49 INFO    [0x700002279000] Metadata for cluster 'devCluster' has 1 replicasets:
2019-04-07 16:30:49 INFO    [0x700002279000] 'default' (3 members, single-master)
2019-04-07 16:30:49 INFO    [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
protocol=classic
```

See how the name and port 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.
- `DEBUG`: includes messages generated inside Router's source code for use in diagnostics. `DEBUG` presents verbose information concerning the inner workings of Router. While it may not be of interest to

the application, use of `DEBUG` may be helpful if you encounter a problem or when Router is not behaving as you expect.

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

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

Log Rotation

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

Note

This functionality is not supported on Windows.

Rotation On Demand

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

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

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

logrotate

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

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

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

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

Chapter 6 MySQL Router REST API

Table of Contents

MySQL Router REST API interface.

6.1 A Simple MySQL Router REST API Guide

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

```
[DEFAULT]
logging_folder=

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

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

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

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

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

Note

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

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

```
[DEFAULT]
logging_folder=

# Exposes http://127.0.0.1:8081
```

```
[http_server]

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

# Exposes /api/20190715/router/status
[rest_router]
require_realm=somerealm

# Exposes /api/20190715/routes/*
#[rest_routing]
#require_realm=somerealm

# Exposes /api/20190715/metadata/*
#[rest_metadata_cache]
#require_realm=somerealm

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

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

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

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

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

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

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

```
{
  "swagger": "2.0",
  "info": {
    "title": "MySQL Router",
    "description": "API of MySQL Router",
    "version": "20190715"
  },
  "basePath": "/api/20190715",
  "tags": [
    {
      "name": "app",
      "description": "Application"
    }
  ],
  "paths": {
    "/router/status": {
      "get": {
        "tags": [
          "app"
        ],
        "description": "Get status of the application",
        "responses": {
```

```
          "200": {
            "description": "status of application",
            "schema": {
              "$ref": "#/definitions/RouterStatus"
            }
          }
        }
      }
    }
  },
  "definitions": {
    "RouterStatus": {
      "type": "object",
      "properties": {
        "timeStarted": {
          "type": "string",
          "format": "data-time"
        },
        "processId": {
          "type": "integer"
        },
        "version": {
          "type": "string"
        },
        "hostname": {
          "type": "string"
        },
        "productEdition": {
          "type": "string"
        }
      }
    }
  }
}
```

Loading *http://127.0.0.1:8081/api/20190715/router/status* prompts for a username and password (that we created in our example) and on success returns Router's current status. For example:

```
{
  "processId": 1883,
  "productEdition": "MySQL Community - GPL",
  "timeStarted": "2022-01-25T21:23:50.442399Z",
  "version": "8.4.0",
  "hostname": "boat"
}
```

We set up a basic Router REST API with an authenticated backend; a REST API with two of the REST API plugins enabled.

6.2 MySQL Router REST API Reference

Knowing the basePath prefix is assumed. The basePath contains the API version, such as "/api/20190715". For example, if the endpoint is "/metadata" then the URL is similar to "https://localhost:8443/api/20190715/metadata". See Section 6.1, “A Simple MySQL Router REST API Guide” for related information.

Table 6.1 MySQL Router REST API Endpoints

Endpoint	Description	Plugin	Method
/metadata	Get metadata instance names	rest_metadata_cache	GET
/metadata/{metadataName}/config	Get metadata configuration details	rest_metadata_cache	GET

Endpoint	Description	Plugin	Method
/metadata/{metadataName}/status	Check metadata status	rest_metadata_cache	GET
/router/status	Check Router status	rest_router	GET
/routes	Get list of routes	rest_routing	GET
/routes/{routeName}/blockedHosts	Get list of blocked IPs	rest_routing	GET
/routes/{routeName}/config	Get route configuration details	rest_routing	GET
/routes/{routeName}/connections	Get route connections	rest_routing	GET
/routes/{routeName}/destinations	Get route destinations	rest_routing	GET
/routes/{routeName}/health	Check route health	rest_routing	GET
/routes/{routeName}/status	Check route status	rest_routing	GET
/connection_pool/{name}/config	Check connection_pool config	rest_connection_pool	GET
/connection_pool/{name}/status	Check connection_pool status	rest_connection_pool	GET
swagger.json	Get swagger file containing available paths and information	rest_api	GET

metadata

GET `/metadata`

Get list of the metadata cache instances

Available Responses

200 Description: List of metadata cache instances

Response Schema

items array

Contains 'name' fields; the name of the metadata instance

Example 200 response data:

```
{
  "items": [
    {
      "name": "myCluster"
    }
  ]
}
```

GET `/metadata/{metadataName}/config`

Get configuration of the metadata cache of a cluster's replicaset

Available Responses

200 Description: Config of metadata cache

Response Schema

clusterName string

		Optional, name of the replication group
	timeRefreshInMs	integer
		TTL number
	groupReplicationId	string
		Optional
	nodes	array
		An array; items include the hostname (string) and port (integer) properties

404 Description: Cache not found

Path Parameters

metadataName (required) string

Name of cluster

Example 200 response data:

```
{
  "clusterName": "myCluster",
  "timeRefreshInMs": 500,
  "groupReplicationId": "e57e9c11-abfe-11ea-b747-0800278566cb",
  "nodes": [
    {
      "hostname": "127.0.0.1",
      "port": 3310
    },
    {
      "hostname": "127.0.0.1",
      "port": 3320
    },
    {
      "hostname": "127.0.0.1",
      "port": 3330
    }
  ]
}
```

GET `/metadata/{metadataName}/status`

Get metadata cache status for a cluster's replicaset

Available Responses

200 Description: Status of the metadata cache

Response Schema

lastRefreshHostname	string
lastRefreshPort	integer

	timeLastRefreshFailed	string
	timeLastRefreshSucceeded	string
	refreshSucceeded	integer
	refreshFailed	integer

404 Description: Cache not found

Path Parameters

metadataName (required) string

Name of the cluster

Example 200 response data:

```
{
  "refreshFailed": 0,
  "refreshSucceeded": 798,
  "timeLastRefreshSucceeded": "2020-06-11T21:17:37.270303Z",
  "lastRefreshHostname": "127.0.0.1",
  "lastRefreshPort": 3310
}
```

router

GET `/router/status`

Get status of router

Available Responses

200 Description: Status of Router

Response Content-Type: *application/json*

Response Schema

	hostname	string
		Name of the host the application is running on; it may be empty if a host is not configured
	processId	integer
		Process ID of the application
	productEdition	string
		Product edition, such as "MySQL Community - GPL"
	timeStarted	string

		A date-time string that the application was started, such as "2020-06-11T22:08:30.978640Z"
	version	string
		Version of the application, such as "8.0.22"

Example 200 response data:

```
{
  "processId": 6435,
  "productEdition": "MySQL Community - GPL",
  "timeStarted": "2020-06-11T21:10:49.420619Z",
  "version": "8.0.20",
  "hostname": "boat"
}
```

routes

GET `/routes`

Get list (names) of the routes supported by MySQL Router

Available Responses

200 Description: List of the supported routes

Response Schema

items	array
	A list of routes

Example 200 response data:

```
{
  "items": [
    {
      "name": "myCluster_ro"
    },
    {
      "name": "myCluster_rw"
    },
    {
      "name": "myCluster_x_ro"
    },
    {
      "name": "myCluster_x_rw"
    }
  ]
}
```

GET `/routes/{routeName}/config`

Get config of a route

Available Responses

200 Description: Config of a route

Response Schema

bindAddress	string
	Address the route is listening on
bindPort	integer
	TCP port the router is listening on
clientConnectTimeoutInMs	integer
	Connection timeout for incoming connections
destinationConnectTimeoutInMs	integer
	Connection timeout for outgoing connections
maxActiveConnections	integer
	Maximum number of active connections
maxConnectErrors	integer
	Maximum number of adjacent connection errors before the client gets blocked
protocol	string
	Protocol, either 'classic' or 'x'
socket	string
	Listening socket or named pipe
routingStrategy	string
	The routing strategy used; such as "round-robin", "round-robin-with-fallback", "first-available", or "next-available" as defined by Router's strategy configuration option

404 Description: Route not found

Path Parameters

routeName (required)	string
	Name of a route

Example 200 response data:

```
{
```

```
  "bindAddress": "0.0.0.0",
  "bindPort": 6446,
  "clientConnectTimeoutInMs": 9000,
  "destinationConnectTimeoutInMs": 15000,
  "maxActiveConnections": 512,
  "maxConnectErrors": 100,
  "protocol": "classic",
  "routingStrategy": "first-available"
}
```


GET `/routes/{routeName}/status`

Get status of a route

Available Responses

200	Description: Status of a route	
	Response Schema	
	activeConnections	integer
		Number of active connections on the route
	totalConnections	integer
		Number of connections handled by the route
	blockedHosts	integer
		Number of blocked hosts
404	Description: Route not found	

Example 200 response data:


```
{
  "activeConnections": 1,
  "totalConnections": 1,
  "blockedHosts": 0
}
```


Path Parameters

routeName (required)	string	
	Name of a route	

GET `/routes/{routeName}/health`

Get health of a route

Available Responses

200	Description: Health of a route	
	Response Schema	
	isAlive	boolean

404 Description: Route not found

Path Parameters

routeName (required) string

Name of a route

Example 200 response data:

```
{
  "isAlive": true
}
```

GET `/routes/{routeName}/destinations`

Get destinations of a route

Available Responses

200 Description: Destinations of a route

Response Schema

items array

Contains 'address' (string, IP address of the destination node), and 'port' (integer, port of the destination node)

404 Description: Route not found

Path Parameters

routeName (required) string

Name of a route

Example 200 response data:

```
{
  "items": [
    {
      "address": "127.0.0.1",
      "port": 3320
    },
    {
      "address": "127.0.0.1",
      "port": 3330
    }
  ]
}
```

GET `/routes/{routeName}/connections`

Get connections of a route

Available Responses

200 Description: Connections of a route

Response Schema

	items	array Each items entry contains the following: • bytesFromServer: integer, number of bytes sent from server to the client over the given connection • BytesToServer: integer, number of bytes sent from the client to the server over the given connection • sourceAddress: string, adddress:port pair of the connection source (client) • destinationAddress: string, adddress:port pair of the connection destination (server) • timeStarted: string, timepoint of the connection initialization • timeConnectedToServer: string, timepoint when the connection successfully established • timeLastSentToServer: string, timepoint when there was last data sent from client to server on the given connection • timeLastReceivedFromServer: string, timepoint when there was last data sent from server to client on the given connection
404	Description: Route not found	

Path Parameters

routeName (required)	string Name of a route	

Example 200 response data:


```
{
  "items": [
    {
      "bytesFromServer": 2952,
      "bytesToServer": 743,
      "sourceAddress": "127.0.0.1:54098",
      "destinationAddress": "127.0.0.1:3310",
      "timeStarted": "2020-06-11T21:28:20.882204Z",
```

```
      "timeConnectedToServer": "2020-06-11T21:28:20.882513Z",
      "timeLastSentToServer": "2020-06-11T21:28:20.886969Z",
      "timeLastReceivedFromServer": "2020-06-11T21:28:20.886968Z"
    }
  ]
}
```

GET `/routes/{routeName}/blockedHosts`

Get blocked host list for a route

Available Responses

200	Description: Blocked host list for a route	
	Response Schema	
	items	array
		IP addresses that are currently blocked by the routing core
404	Description: Route not found	

Path Parameters

routeName (required)	string	
	Name of a route	

Example 200 response data:

```
{
  "items": []
}
```

connection_pool

GET `/connection_pool/{name}/config`

Shows `maxIdleServerConnections` as defined by the `max_idle_server_connection` configuration option. This is the maximum number (integer) of idling server connections in the connection pool.

Shows `idleTimeout` as defined by the `idle_timeout` configuration option. This is the timeout in seconds (integer) before connections in the connection pool are closed.

GET `/connection_pool/{name}/status`

Shows `reusedConnections` as a count (integer) of client connections that reused a server connection since the application started.

Shows `idleServerConnections` as a count (integer) of idling server connections currently in the connection pool.

swagger.json

GET `/swagger.json`

Get a swagger (OpenAPI) file for the local REST API instance. Accessing the file does not require authentication; anyone with access to the REST API can generate and view it. The OpenAPI content depends on the active REST API plugins.

Example 200 response data:


```
{
  "swagger": "2.0",
  "info": {
    "title": "MySQL Router",
    "description": "API of MySQL Router",
    "version": "20190715"
  },
  "basePath": "/api/20190715",
  "tags": [
    {
      "name": "connectionpool",
      "description": "Connection Pool"
    },
    {
      "name": "cluster",
      "description": "InnoDB Cluster"
    },
    {
      "name": "app",
      "description": "Application"
    },
    {
      "name": "routes",
      "description": "Routes"
    }
  ],
  "paths": {
    "/connection_pool/{connectionPoolName}/status": {
      "get": {
        "tags": [
          "connectionpool"
        ],
        "description": "Get status of a route",
        "responses": {
          "200": {
            "description": "status of a route",
            "schema": {
              "$ref": "#/definitions/ConnectionPoolStatus"
            }
          },
          "404": {
            "description": "route not found"
          }
        }
      },
      "parameters": [
        {
          "$ref": "#/parameters/connectionPoolNameParam"
        }
      ]
    },
    "/connection_pool/{connectionPoolName}/config": {
      "get": {
        "tags": [
          "connectionpool"
        ],
        "description": "Get config of a route",
        "responses": {
          "200": {
            "description": "config of a route",
```

```
          "schema": {
            "$ref": "#/definitions/ConnectionPoolConfig"
          }
        },
        "404": {
          "description": "route not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/connectionPoolNameParam"
      }
    ]
  },
  "/connection_pool": {
    "get": {
      "tags": [
        "connectionpool"
      ],
      "description": "Get list of the connection pools",
      "responses": {
        "200": {
          "description": "list of the connection pools",
          "schema": {
            "$ref": "#/definitions/ConnectionPoolList"
          }
        }
      }
    }
  },
  "/metadata/{metadataName}/config": {
    "get": {
      "tags": [
        "cluster"
      ],
      "description": "Get config of the metadata cache of a replicaset of a cluster",
      "responses": {
        "200": {
          "description": "config of metadata cache",
          "schema": {
            "$ref": "#/definitions/MetadataConfig"
          }
        },
        "404": {
          "description": "cache not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/metadataNameParam"
      }
    ]
  },
  "/metadata/{metadataName}/status": {
    "get": {
      "tags": [
        "cluster"
      ],
      "description": "Get status of the metadata cache of a replicaset of a cluster",
      "responses": {
        "200": {
          "description": "status of metadata cache",
          "schema": {
            "$ref": "#/definitions/MetadataStatus"
          }
```

```
        },
        "404": {
          "description": "cache not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/metadataNameParam"
      }
    ]
  },
  "/metadata": {
    "get": {
      "tags": [
        "cluster"
      ],
      "description": "Get list of the metadata cache instances",
      "responses": {
        "200": {
          "description": "list of the metadata cache instances",
          "schema": {
            "$ref": "#/definitions/MetadataList"
          }
        }
      }
    }
  },
  "/router/status": {
    "get": {
      "tags": [
        "app"
      ],
      "description": "Get status of the application",
      "responses": {
        "200": {
          "description": "status of application",
          "schema": {
            "$ref": "#/definitions/RouterStatus"
          }
        }
      }
    }
  },
  "/routing/status": {
    "get": {
      "tags": [
        "routing"
      ],
      "description": "Get status of the routing plugin",
      "responses": {
        "200": {
          "description": "status of the routing plugin",
          "schema": {
            "$ref": "#/definitions/RoutingGlobalStatus"
          }
        }
      }
    }
  },
  "/routes/{routeName}/config": {
    "get": {
      "tags": [
        "routes"
      ],
      "description": "Get config of a route",
      "responses": {
```

```
        "200": {
          "description": "config of a route",
          "schema": {
            "$ref": "#/definitions/RouteConfig"
          }
        },
        "404": {
          "description": "route not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/routeNameParam"
      }
    ]
  },
  "/routes/{routeName}/status": {
    "get": {
      "tags": [
        "routes"
      ],
      "description": "Get status of a route",
      "responses": {
        "200": {
          "description": "status of a route",
          "schema": {
            "$ref": "#/definitions/RouteStatus"
          }
        },
        "404": {
          "description": "route not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/routeNameParam"
      }
    ]
  },
  "/routes/{routeName}/health": {
    "get": {
      "tags": [
        "routes"
      ],
      "description": "Get health of a route",
      "responses": {
        "200": {
          "description": "health of a route",
          "schema": {
            "$ref": "#/definitions/RouteHealth"
          }
        },
        "404": {
          "description": "route not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/routeNameParam"
      }
    ]
  },
  "/routes/{routeName}/destinations": {
    "get": {
```

```
      "tags": [
        "routes"
      ],
      "description": "Get destinations of a route",
      "responses": {
        "200": {
          "description": "destinations of a route",
          "schema": {
            "$ref": "#/definitions/RouteDestinationList"
          }
        },
        "404": {
          "description": "route not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/routeNameParam"
      }
    ]
  },
  "/routes/{routeName}/connections": {
    "get": {
      "tags": [
        "routes"
      ],
      "description": "Get connections of a route",
      "responses": {
        "200": {
          "description": "connections of a route",
          "schema": {
            "$ref": "#/definitions/RouteConnectionsList"
          }
        },
        "404": {
          "description": "route not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/routeNameParam"
      }
    ]
  },
  "/routes/{routeName}/blockedHosts": {
    "get": {
      "tags": [
        "routes"
      ],
      "description": "Get blocked host list for a route",
      "responses": {
        "200": {
          "description": "blocked host list for a route",
          "schema": {
            "$ref": "#/definitions/RouteBlockedHostList"
          }
        },
        "404": {
          "description": "route not found"
        }
      }
    },
    "parameters": [
      {
        "$ref": "#/parameters/routeNameParam"
```

```
      }
    ]
  },
  "/routes": {
    "get": {
      "tags": [
        "routes"
      ],
      "description": "Get list of the routes",
      "responses": {
        "200": {
          "description": "list of the routes",
          "schema": {
            "$ref": "#/definitions/RouteList"
          }
        }
      }
    }
  }
},
"definitions": {
  "ConnectionPoolStatus": {
    "type": "object",
    "properties": {
      "reusedServerConnections": {
        "type": "integer"
      },
      "idleServerConnections": {
        "type": "integer"
      }
    }
  },
  "ConnectionPoolConfig": {
    "type": "object",
    "properties": {
      "idleTimeoutInMs": {
        "type": "integer"
      },
      "maxIdleServerConnections": {
        "type": "integer"
      }
    }
  },
  "ConnectionPoolSummary": {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      }
    }
  },
  "ConnectionPoolList": {
    "type": "object",
    "properties": {
      "items": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/ConnectionPoolSummary"
        }
      }
    }
  },
  "MetadataStatus": {
    "type": "object",
    "properties": {
      "lastRefreshHostname": {
        "type": "string"
```

```
      },
      "lastRefreshPort": {
        "type": "integer"
      },
      "timeLastRefreshFailed": {
        "type": "string",
        "format": "data-time"
      },
      "timeLastRefreshSucceeded": {
        "type": "string",
        "format": "data-time"
      },
      "refreshSucceeded": {
        "type": "integer"
      },
      "refreshFailed": {
        "type": "integer"
      }
    }
  },
  "MetadataConfig": {
    "type": "object",
    "properties": {
      "clusterName": {
        "type": "string"
      },
      "timeRefreshInMs": {
        "type": "integer"
      },
      "groupReplicationId": {
        "type": "string"
      },
      "nodes": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "hostname": {
              "type": "string"
            },
            "port": {
              "type": "integer"
            }
          }
        }
      }
    }
  },
  "MetadataSummary": {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      }
    }
  },
  "MetadataList": {
    "type": "object",
    "properties": {
      "items": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/MetadataSummary"
        }
      }
    }
  },
```

```
    "ClusterNodeSummary": {
      "type": "object",
      "properties": {
        "groupUuid": {
          "type": "string"
        },
        "serverUuid": {
          "type": "string"
        }
      }
    },
    "ClusterNodeList": {
      "type": "object",
      "properties": {
        "items": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/ClusterNodeSummary"
          }
        }
      }
    },
    "ClusterSummary": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        }
      }
    },
    "ClusterList": {
      "type": "object",
      "properties": {
        "items": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/ClusterSummary"
          }
        }
      }
    },
    "RouterStatus": {
      "type": "object",
      "properties": {
        "timeStarted": {
          "type": "string",
          "format": "data-time"
        },
        "processId": {
          "type": "integer"
        },
        "version": {
          "type": "string"
        },
        "hostname": {
          "type": "string"
        },
        "productEdition": {
          "type": "string"
        }
      }
    },
    "RoutingGlobalStatus": {
      "totalMaxConnections": "number of total connections allowed",
      "currentMaxConnections": "number of current total connections"
    },
    "RouteHealth": {
```

```
      "type": "object",
      "properties": {
        "isAlive": {
          "type": "boolean"
        }
      }
    },
    "RouteStatus": {
      "type": "object",
      "properties": {
        "activeConnections": {
          "type": "integer"
        },
        "totalConnections": {
          "type": "integer"
        },
        "blockedHosts": {
          "type": "integer"
        }
      }
    },
    "RouteConfig": {
      "type": "object",
      "properties": {
        "bindAddress": {
          "type": "string"
        },
        "bindPort": {
          "type": "integer"
        },
        "clientConnectTimeoutInMs": {
          "type": "integer"
        },
        "destinationConnectTimeoutInMs": {
          "type": "integer"
        },
        "maxActiveConnections": {
          "type": "integer"
        },
        "maxConnectErrors": {
          "type": "integer"
        },
        "protocol": {
          "type": "string"
        },
        "socket": {
          "type": "string"
        },
        "routingStrategy": {
          "type": "string"
        },
      }
    },
    "RouteSummary": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        }
      }
    },
    "RouteList": {
      "type": "object",
      "properties": {
        "items": {
          "type": "array",
          "items": {
```

```
          "$ref": "#/definitions/RouteSummary"
        }
      }
    }
  },
  "RouteDestinationSummary": {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      }
    }
  },
  "RouteDestinationList": {
    "type": "object",
    "properties": {
      "items": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/RouteDestinationSummary"
        }
      }
    }
  },
  "RouteBlockedHostSummary": {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      }
    }
  },
  "RouteBlockedHostList": {
    "type": "object",
    "properties": {
      "items": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/RouteBlockedHostSummary"
        }
      }
    }
  },
  "RouteConnectionsSummary": {
    "type": "object",
    "properties": {
      "timeStarted": {
        "type": "string",
        "format": "date-time",
        "description": "timepoint when connection to server was initiated"
      },
      "timeConnectedToServer": {
        "type": "string",
        "format": "date-time",
        "description": "timepoint when connection to server succeeded"
      },
      "timeLastSentToServer": {
        "type": "string",
        "format": "date-time",
        "description": "timepoint when there was last data sent from client to server"
      },
      "timeLastReceivedFromServer": {
        "type": "string",
        "format": "date-time",
        "description": "timepoint when there was last data sent from server to client"
      },
      "bytesFromServer": {
```

```
          "type": "integer",
          "description": "bytes sent to destination"
        },
        "bytesToServer": {
          "type": "integer",
          "description": "bytes received from destination"
        },
        "destinationAddress": {
          "type": "string",
          "description": "address of the destination of the connection"
        },
        "sourceAddress": {
          "type": "string",
          "description": "address of the source of the connection"
        }
      }
    },
    "RouteConnectionsList": {
      "type": "object",
      "properties": {
        "items": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/RouteConnectionsSummary"
          }
        }
      }
    }
  },
  "parameters": {
    "connectionPoolNameParam": {
      "name": "connectionPoolName",
      "in": "path",
      "description": "name of a connection pool",
      "required": true,
      "type": "string"
    },
    "metadataNameParam": {
      "name": "metadataName",
      "in": "path",
      "description": "name of cluster",
      "required": true,
      "type": "string"
    },
    "clusterNameParam": {
      "name": "clusterName",
      "in": "path",
      "description": "name of cluster",
      "required": true,
      "type": "string"
    },
    "routeNameParam": {
      "name": "routeName",
      "in": "path",
      "description": "name of a route",
      "required": true,
      "type": "string"
    }
  }
}
```

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 servers 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 MySQL AdminAPI.

A.4. Does the router inspect packets?

No.

A.5. Does the router impact performance?

Introducing a component in a communication stream incurs a certain amount of overhead; this is affected heavily by workload. Fortunately, performance testing on the current release has shown approximately 1% within the same speed as a direct connection for simple redirect connection routing.

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

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

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

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

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

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

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

Router 8.0 introduced the `routing_strategy` option. It offers the *first-available*, *next-available*, *round-robin* and *round-robin-with-fallback* strategies. See the `routing_strategy` documentation for additional details.

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

Over 50,000 as of MySQL Router 8.0.22, depending on the system's poll (poll or linux_epoll) limits and also depending on the number of available CPU cores/threads.

Earlier MySQL Router versions had had a limit closer to 5000, depending on the operating system's poll() limits.

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

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