MySQL Router

Abstract

MySQL Router is lightweight middleware that provides transparent routing between your application and any backend 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 backend MySQL Servers. The pluggable architecture also enables developers to extend MySQL Router for custom use cases.

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

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

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

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

Document generated on: 2016-05-31 (revision: 47872)

Table of Contents

Preface and Legal Notices v
1 General Information 1
    1.1 Supported Features 1
        1.1.1 Connection Routing 1
        1.1.2 Fabric Integration 2
        1.1.3 MySQL Harness: Plugin Capabilities and More 3
    1.2 Architecture 3
    1.3 Using MySQL Router 4
        1.3.1 Deploying MySQL Router 5
        1.3.2 Developing Applications to Use MySQL Router 5
2 Installation 9
    2.1 Installing MySQL Router on Linux 9
    2.2 Installing MySQL Router on OS X 11
    2.3 Installing MySQL Router on Windows 11
    2.4 Installing MySQL Router from Source Code 11
        2.4.1 Prerequisites 12
        2.4.2 Compiling the Source Code 13
        2.4.3 Installing from Source Code 15
        2.4.4 Testing the Installation 16
    2.5 Postinstallation Testing 16
3 Configuration 21
    3.1 Configuration File Locations 21
    3.2 Configuration File Setup 22
        3.2.1 General 23
        3.2.2 Connection Routing (Standalone) 23
        3.2.3 Logging 26
        3.2.4 Configuration File Example 27
4 MySQL Router Application 29
    4.1 User Options 29
    4.2 Starting the Router 29
    4.3 Using the Logging Feature 30
5 Plugins 33
    5.1 Connection Routing Plugin 33
    5.2 Fabric Cache Plug-in 33
6 Use Cases and Examples 37
A MySQL Router Frequently Asked Questions 39
B MySQL Router Unit Tests 41

Preface and Legal Notices

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

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

Legal Notices

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

Documentation Accessibility

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

Access to Oracle Support

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

Chapter 1 General Information

Table of Contents

1.1 Supported Features 1
    1.1.1 Connection Routing 1
    1.1.2 Fabric Integration 2
    1.1.3 MySQL Harness: Plugin Capabilities and More 3
1.2 Architecture 3
1.3 Using MySQL Router 4
    1.3.1 Deploying MySQL Router 5
    1.3.2 Developing Applications to Use MySQL Router 5

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

MySQL Router also connects seamlessly with MySQL Fabric permitting Fabric to store and manage the high availability groups for routing, making it simpler to manage groups of MySQL Servers for redundancy and continued operation.

Thus, you can use the router to build more robust applications with connection routing provided by the router. You can also build your solutions to be Fabric aware and allow Fabric to maintain the list of servers available for connection routing.

The router is a single executable that runs on the same platform as the application. While it can be run in the background, you can turn on logging features and view the logs console (or log to a file), which may be helpful for application development and tuning.

Like the MySQL server, Router uses a configuration file with information for how to perform routing. The configuration file can contain several sections to permit several applications to use a single router instance. We will see more about the configuration file in later sections.

The next section explains the main features of MySQL Router in more detail.

1.1 Supported Features

The MySQL Router 2.0.3 has three major features; simple connection routing, Fabric integration connection routing using the Fabric cache plugin, and support for plugins via the MySQL Harness.

Note

While Router does not support SSL connections, it is possible to secure network traffic using 3rd party tools that provide SSL tunneling services, such as stunnel.

1.1.1 Connection Routing

Connection routing (also called standalone routing to distinguish it from Fabric integrated connection routing - see below) is the ability to permit the redirection of connections sent to the router to an available MySQL server. Connection routing is simply the redirection of the MySQL packets in their entirety without inspection.

This means you can setup your application to connect to the router and should the current MySQL server fail, retry the connection and the router will select a new MySQL server to redirect the connection.

We call this 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 (such as, the server goes offline), the application will encounter a connection failure. However, a new connection attempt will trigger the router to find another server to connect.

We specify the servers we want to use in a routing strategy in the configuration file. For example, the following section tells the router to listen for connections on port 7002 of the localhost, and then redirect those connections to any of the servers in the list named by the `destinations` option, including servers running on the localhost listening on ports 3306, 3307, and 3308. Finally, we use the `mode` option to tell the router to allow both readers and writers. For more information about the available modes, see the section entitled, Configuration File Setup below.

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

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

When a server is no longer reachable, the router moves to the next one in the list. When the list is exhausted, the router halts redirecting.

1.1.2 Fabric Integration

Using MySQL Fabric with the router is another form of connection routing (also called Fabric integration) but instead of configuring the destinations manually, the list is provided by MySQL Fabric. Recall Fabric allows you to manage a farm of MySQL servers by grouping the servers in high availability groups for redundancy (allowing for automatic failover) and for easier management. Thus, Fabric maintains a list of the servers in the farm including their roles and how each is configure. Fabric can supply this list to the router via the Fabric cache.

With the router, we can use Fabric to supply the routes for connections to an appropriated MySQL Server based on information provided in Fabric via the Fabric cache. Essentially, we connect the router to Fabric and get the high availability feature for free.

Fortunately, this is not complicated to set up. All you need is another routing strategy like we saw with simple redirect connection routing. In this case, we set up a new section in the configuration file as demonstrated below.

We set up the router to listen on port `19906` on the localhost with a `mode` set to `read-only`, which is typical in an application written to use Fabric (or replication in general). We need two sections in the configuration file like so:

```
[fabric_cache:webfarm]
address = fabric.example.com
user = fabricuser

[routing:webfarm]
bind_port = 19906
destinations = fabric+cache://webfarm/group/homepage/
mode = read-only
```

The first section sets the address and user for the Fabric connection. The second sets the routing strategy.

Notice here we set the `destinations` option to point to the Fabric cache via a URI. When you set up a Fabric integration like this, you will be prompted for the Fabric password when you start the router.

We used the section name "`fabric_cache`" to indicate this is a fabric integrated connection routing strategy. Finally, notice we provide the Fabric user account using the `user` option.

1.1.3 MySQL Harness: Plugin Capabilities and More

One of the interesting features of MySQL Router is not something the average user will see - it is under the hood. More specifically, the router is designed to permit the use of plugins to enable features. The plugin technology is called *MySQL Harness*.

MySQL Harness provides a number of advantages for the router including the ability to load new functionality without having to alter the original `mysqlrouter` application, which means faster development times for new features and the possibility of selectively loading features. The harness also provides logging, which enables developers to turn on a log of events without having to write any specialized logging code.

The features provided my MySQL Harness include the following:

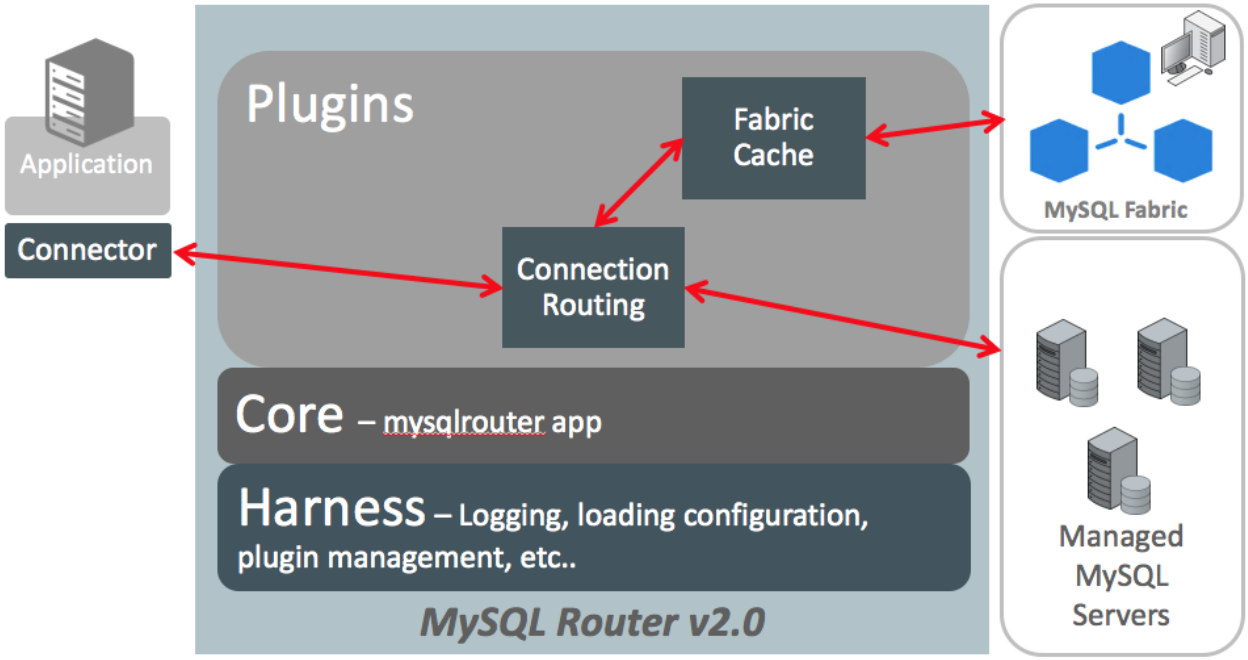
- Dependency tracking for plugins
- Version handling for plugins
- Loading, starting, and stopping plugins
- Permits creation of third-party plugins
- Provides a configuration file library for reading configuration files
- Provides platform agnostic file support
- Provides a robust logging facility

While the harness feature set and usage may not be obvious initially, the long-term benefits of developing with a plugin-based design are well documented.

1.2 Architecture

The architecture of the MySQL Router is a single application built as a host incorporating the MySQL Harness plugin feature. Major features for MySQL Router are loaded as plugins. For example, both the connection routing and Fabric cache components are plugins. The following figure illustrates how the router is built.

Figure 1.1 MySQL Router Architecture



Notice the application, `mysqlrouter`, is based on the harness. Plugins are therefore enabled within the application.

Notice also the arrows pointing to boxes outside of the router application. On the left we see the application that is connected to the router. On the right, we see two connection destinations. On the top is Fabric cache and below that a representation of a set of MySQL servers in a replication topology.

Now, let us follow the arrows as they pass through the router.

Start with the connection routing plugin. This is simple redirection. Thus, when the application connects to the router, the router reads the destinations from the configuration file and redirects to one of the servers in the list.

Now let us see how the Fabric cache plugin is used. To use the Fabric cache, the routing strategy specifies the URL for the Fabric installation. In this case, the application would connect to the router, then the router would get the destination list from Fabric, and then redirect the connection to the one of the servers in the list.

As you can see, the router is designed to be modular and take advantage of such architectures when implementing features. Thus, future features of the router will appear along with the two plugins shown in this drawing.

For more information about the Fabric Cache and Connection Routing plugins, see the section entitled, Plugins.

1.3 Using MySQL Router

Now that we understand what the router does and how it is configured at a high level, let us discuss how best to use the router in your environment.

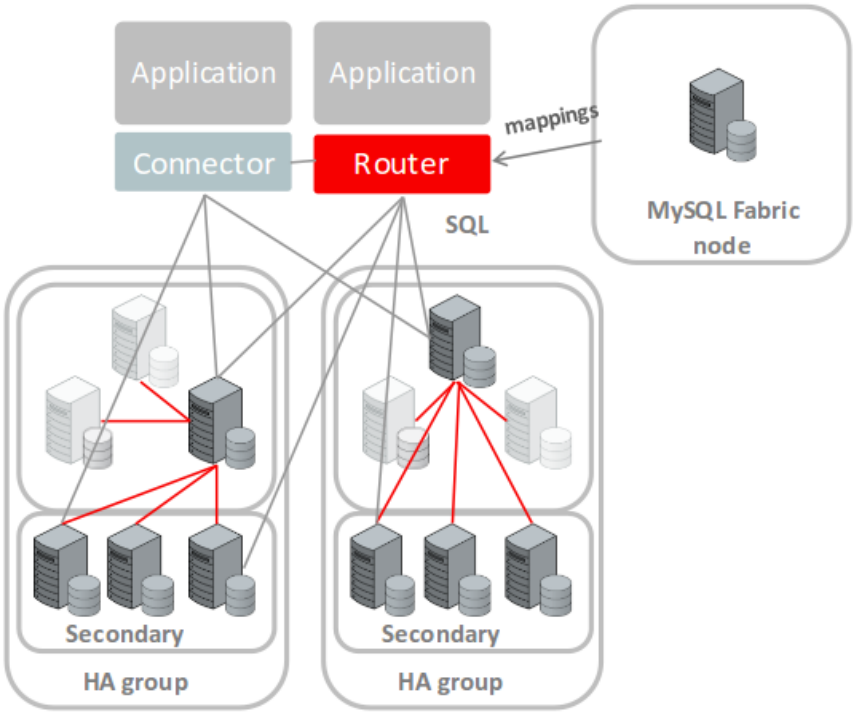
1.3.1 Deploying MySQL Router

In its current form, the MySQL Router is best used alongside the application. That is, you should install the router on the same machines that hosts the application. While this is not a rule or requirement, it is the recommended practice.

Since the router is on the same machine as the application, this means you can write your application to monitor the `mysqlrouter` executable and restart it should you need to. For example, if the list of servers in the `destinations` option is exhausted, you can restart the router with a new destinations list or restart the router to retry the servers in the list.

The following graphic represents how you could employ the router with your application:

Figure 1.2 Positioning MySQL Router



You can run several instances of the router throughout your network. You do not need to isolate the router to a single machine or even a single instance. This is because there is nothing in the router to give it affinity for any one server or machine. It is essentially agnostic in that manner.

1.3.2 Developing Applications to Use MySQL Router

Developing applications for use with the router does not require any special application library or interface. Aside from managing the MySQL Router instance, your application can be written as if the router were not there.

The only difference you may want to do is how you use connections to the MySQL server. If your application uses a single connection made at startup and does not test for connection errors, you may need to change that behavior.

This is because the router merely redirects connections when the connection is attempted. The router does not read packets or perform any form of analysis. Thus, if a server goes down, the router will return a connection error to the host application.

Thus, 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, using MySQL Router will be seamless.

The following gives you a better idea of why you may want to use the router, and a look into how it is used from an application.

User Stories

There are several key user stories for MySQL Router. These include the following.

- As a developer, I want my application connect to a service so it gets a connection to, by default, the current Master (primary) of a highly available MySQL Fabric group.
- As an administrator, I want to set up multiple services so MySQL Router listens on a different port for each highly available Fabric group.
- 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/application.
- As an administrator, I want to configure a mode for each connection routing service so I can specify whether a Master (primary) or Slave (secondary) is returned.

Workflow with Router

The workflow for using the router is as follows.

1. MySQL Client/Connector connects to MySQL Router on, for example, port 8500.
2. Router checks which MySQL Server is available.
3. Router bridges the connection from client to the MySQL server.

Connecting using the Router

Applications can connect to the router on the machine and port specified in the router's configuration file. For example, port 8500. The following shows an example in Python using the MySQL Connector/Python connector library.

```
cnx = mysql.connector.connect(host='router.example.com', port=8500, user='scott', password='tiger')
cur = cnx.cursor()
cnx.execute("SELECT ...")
```

Recommendations

The following lists recommendations for using MySQL Router.

- Install and run the router on the same host as the application.
- Bind the router to the localhost using `bind_port = 127.0.0.1:<port>` in the configuration file.
- MySQL Router works best with MySQL Fabric, but can be used without Fabric.

Now that we know what the MySQL Router is and how it is used, let us dive deeper into installing and configuring the router.

Chapter 2 Installation

Table of Contents

2.1 Installing MySQL Router on Linux 9
2.2 Installing MySQL Router on OS X 11
2.3 Installing MySQL Router on Windows 11
2.4 Installing MySQL Router from Source Code 11
2.4.1 Prerequisites 12
2.4.2 Compiling the Source Code 13
2.4.3 Installing from Source Code 15
2.4.4 Testing the Installation 16
2.5 Postinstallation Testing 16

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

Requirements

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

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

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

- MySQL Client Libraries development packages. For example, on Ubuntu this is the `libmysqlclient-dev` package.

- CMake 2.8.9 or later.

2.1 Installing MySQL Router on Linux

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

Installation options include:

- **Official MySQL Yum or APT repository packages**: These binaries are built by the MySQL Release team. For additional information about installing these, see Yum or APT. They contain the most recent version of MySQL Router.

- **Download official MySQL packages**: Downloads are available at http://dev.mysql.com/downloads/router.

- **Download the source code and compile yourself**: The source code is available at http://dev.mysql.com/downloads/router as a `tar.gz` or RPM package. Alternatively, the source code is also available on GitHub.

Note

Binaries are not available for all platforms, but MySQL Router can be built and installed from source code. See the section Installing MySQL Router from Source Code for more details.

For information about compiling MySQL Router on Linux, see Section 2.4.2, “Compiling the Source Code”.

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

Installing DEB packages

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

Using the APT Package Manager

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

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

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

- Update your APT repository:

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

- Next, install MySQL Router. For example:

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

Manually Installing a Package

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

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

package`.deb` is the MySQL Router package name; for example, `mysql-router-`*version*`1ubu1404-amd64.deb`, where *version* is the MySQL Router version number.

Installing RPM packages

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

Using the Yum Package Manager

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

```
shell> sudo rpm -Uvh mysql-community-release-el7-7.noarch.rpm
```

- Next, install MySQL Router. For example:

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

Manually Installing an RPM Package

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

package`.rpm` is the MySQL Router package name; for example, `mysql-router-`*version*`-1fc10.x86_64.rpm`, where *version* is the MySQL Router version number.

Uninstalling

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

Uninstalling DEB packages

To uninstall a Debian package, use this command:

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

This command does not remove the configuration files. If you wish to also remove the configuration files, use this command:

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

Uninstalling RPM packages

To uninstall an RPM package, use this command:

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

This command does not remove the configuration files.

What Is Not Removed

By default, 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.ini
/etc/apparmor.d/usr.sbin.mysqlrouter
```

2.2 Installing MySQL Router on OS X

See the Linux installation instructions.

For information about how to compile MySQL Router, see Section 2.4.2, “Compiling the Source Code”.

2.3 Installing MySQL Router on Windows

Windows binaries are not yet available.

2.4 Installing MySQL Router from Source Code

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

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

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

The default CMake installation prefixes are used. These are different for each platform, but for most (if not all) Unices this is "`/usr/local`". It is possible to alter the installation path with the CMake variable "**CMAKE_INSTALL_PREFIX**". For example:

```
shell> mkdir build && cd build
shell> cmake .. -DINSTALL_LAYOUT=STANDALONE -DCMAKE_INSTALL_PREFIX=/opt/mysql/router2.0
```

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

Note

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

Download and unpack the source files. Then:

Linux and OS X

```
shell> tar xzf mysql-router-2.0.3-src.tar.gz
shell> cd mysql-router-2.0.3-src
```

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

Note

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

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

2.4.1 Prerequisites

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

- CMake 2.8.9 or higher
- MySQL Server 5.5 or higher client libraries and header files
- Code development tools including gcc, make, and assorted utilities for C++ 11 including GCC 4.8 and later, glibc 2.17 and later, and clang 3.3 and later
- Python (optional, for some unit tests)

Note

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

2.4.2 Compiling the Source Code

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

Note

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

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

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

Note

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

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

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

```
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Performing Test support_11
-- Performing Test support_11 - Success
-- Performing Test support_0x
-- Performing Test support_0x - Success
-- Found MySQL Libraries 5.6.27; using <path to server code>/lib/libmysqlclient.so
-- Loading module 'router'
-- Loading module 'fabric_cache'
-- Loading module 'routing'
-- Configuring done
-- Generating done
-- Build files have been written to: <path to router code>/build
```

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

```
shell>  make
Scanning dependencies of target harness-archive
[  2%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/loader.cc.o
[  5%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/utilities.cc.o
[  8%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/config_parser.cc.o
[ 11%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/designator.cc.o
[ 14%] Building CXX object harness/harness/CMakeFiles/harness-archive.dir/src/filesystem-posix.cc.o
Linking CXX static library libmysqlharness.a
[ 14%] Built target harness-archive
Scanning dependencies of target harness-library
[ 17%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/loader.cc.o
[ 20%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/utilities.cc.o
[ 22%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/config_parser.cc.o
[ 25%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/designator.cc.o
[ 28%] Building CXX object harness/harness/CMakeFiles/harness-library.dir/src/filesystem-posix.cc.o
Linking CXX shared library libmysqlharness.so
[ 28%] Built target harness-library
Scanning dependencies of target logger
[ 31%] Building CXX object harness/plugins/logger/CMakeFiles/logger.dir/logger.cc.o
Linking CXX shared library ../../../stage/lib/mysqlrouter/logger.so
[ 31%] Built target logger
Scanning dependencies of target keepalive
[ 34%] Building CXX object harness/plugins/keepalive/CMakeFiles/keepalive.dir/src/keepalive.cc.o
Linking CXX shared library ../../../stage/lib/mysqlrouter/keepalive.so
[ 34%] Built target keepalive
Scanning dependencies of target router_lib
[ 37%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/router_app.cc.o
[ 40%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/arg_handler.cc.o
[ 42%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/utils.cc.o
[ 45%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/datatypes.cc.o
[ 48%] Building CXX object src/router/src/CMakeFiles/router_lib.dir/plugin_config.cc.o
Linking CXX shared library ../../../stage/lib/libmysqlrouter.so
[ 48%] Built target router_lib
Scanning dependencies of target mysqlrouter
[ 51%] Building CXX object src/router/src/CMakeFiles/mysqlrouter.dir/main.cc.o
Linking CXX executable ../../../stage/bin/mysqlrouter
[ 51%] Built target mysqlrouter
Scanning dependencies of target fabric_cache
[ 54%] Building CXX object src/fabric_cache/CMakeFiles/fabric_cache.dir/src/fabric_cache_plugin.cc.o
[ 57%] Building CXX object src/fabric_cache/CMakeFiles/fabric_cache.dir/src/plugin_config.cc.o
[ 60%] Building CXX object src/fabric_cache/CMakeFiles/fabric_cache.dir/src/fabric_factory.cc.o
[ 62%] Building CXX object src/fabric_cache/CMakeFiles/fabric_cache.dir/src/fabric.cc.o
[ 65%] Building CXX object src/fabric_cache/CMakeFiles/fabric_cache.dir/src/fabric_cache.cc.o
[ 68%] Building CXX object src/fabric_cache/CMakeFiles/fabric_cache.dir/src/utils.cc.o
[ 71%] Building CXX object src/fabric_cache/CMakeFiles/fabric_cache.dir/src/cache_api.cc.o
Linking CXX shared library ../../stage/lib/mysqlrouter/fabric_cache.so
[ 71%] Built target fabric_cache
```

```
Scanning dependencies of target cache_test
[ 74%] Building CXX object src/fabric_cache/CMakeFiles/cache_test.dir/fabric_cache_dev.cc.o
Linking CXX executable ../../stage/bin/cache_test
[ 74%] Built target cache_test
Scanning dependencies of target routing
[ 77%] Building CXX object src/routing/CMakeFiles/routing.dir/src/routing_plugin.cc.o
[ 80%] Building CXX object src/routing/CMakeFiles/routing.dir/src/plugin_config.cc.o
[ 82%] Building CXX object src/routing/CMakeFiles/routing.dir/src/mysql_routing.cc.o
[ 85%] Building CXX object src/routing/CMakeFiles/routing.dir/src/utils.cc.o
[ 88%] Building CXX object src/routing/CMakeFiles/routing.dir/src/destination.cc.o
[ 91%] Building CXX object src/routing/CMakeFiles/routing.dir/src/dest_fabric_cache.cc.o
[ 94%] Building CXX object src/routing/CMakeFiles/routing.dir/src/dest_first_available.cc.o
[ 97%] Building CXX object src/routing/CMakeFiles/routing.dir/src/uri.cc.o
[100%] Building CXX object src/routing/CMakeFiles/routing.dir/src/routing.cc.o
Linking CXX shared library ../../stage/lib/mysqlrouter/routing.so
[100%] Built target routing
```

2.4.3 Installing from Source Code

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

```
shell>  sudo make install
[ 14%] Built target harness-archive
[ 28%] Built target harness-library
[ 31%] Built target logger
[ 34%] Built target keepalive
[ 48%] Built target router_lib
[ 51%] Built target mysqlrouter
[ 71%] Built target fabric_cache
[ 74%] Built target cache_test
[100%] Built target routing
Install the project...
-- Install configuration: ""
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/loader.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/filesystem.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/plugin.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/config_parser.h
-- Installing: /usr/local/lib/libmysqlharness.a
-- Installing: /usr/local/lib/libmysqlharness.so.0
-- Up-to-date: /usr/local/lib/libmysqlharness.so
-- Set runtime path of "/usr/local/lib/libmysqlharness.so.0" to "$ORIGIN/../lib"
-- Installing: /usr/local/lib/mysqlrouter/keepalive.so
-- Set runtime path of "/usr/local/lib/mysqlrouter/keepalive.so" to "$ORIGIN"
-- Installing: /usr/local/lib/mysqlrouter/logger.so
-- Set runtime path of "/usr/local/lib/mysqlrouter/logger.so" to "$ORIGIN"
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/logger.h
-- Up-to-date: /usr/local/share/doc/mysqlrouter/README.txt
-- Up-to-date: /usr/local/share/doc/mysqlrouter/License.txt
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/plugin_config.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/utils.h
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/datatypes.h
-- Installing: //var
-- Installing: //var/local
-- Installing: //var/local/mysqlrouter
-- Installing: //var/local/mysqlrouter/log
-- Installing: //var
-- Installing: //var/local
-- Installing: //var/local/mysqlrouter
-- Installing: //var/local/mysqlrouter/run
-- Installing: /usr/local/etc
-- Installing: /usr/local/etc/mysqlrouter
-- Installing: /usr/local/bin/mysqlrouter
-- Set runtime path of "/usr/local/bin/mysqlrouter" to "$ORIGIN/../lib"
-- Installing: /usr/local/lib/libmysqlrouter.so.1
```

```
-- Up-to-date: /usr/local/lib/libmysqlrouter.so
-- Set runtime path of "/usr/local/lib/libmysqlrouter.so.1" to "$ORIGIN/../lib"
-- Installing: /usr/local/lib/mysqlrouter/fabric_cache.so
-- Set runtime path of "/usr/local/lib/mysqlrouter/fabric_cache.so" to "$ORIGIN:<path to server code>/lib"
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/fabric_cache.h
-- Installing: /usr/local/lib/mysqlrouter/routing.so
-- Set runtime path of "/usr/local/lib/mysqlrouter/routing.so" to "$ORIGIN"
-- Up-to-date: /usr/local/include/mysql/mysqlrouter/routing.h
```

2.4.4 Testing the Installation

You can ensure the installation succeeded by running the following command. You should see a similar output on your system. An example of setting the Router for simple routing is shown in the Postinstallation Testing section.

Note

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

```
shell>  mysqlrouter --help

MySQL Router v2.0.3 on Linux (64-bit) (GPL community edition)
Copyright (c) 2015, Oracle and/or its affiliates. All rights reserved.

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

Start MySQL Router.

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

Usage: mysqlrouter [-v|--version] [-h|--help]
                   [-c|--config=<path>]
                   [-a|--extra-config=<path>]

Options:
  -v, --version
        Display version information and exit.
  -h, --help
        Display this help and exit.
  -c <path>, --config <path>
        Only read configuration from given file.
  -a <path>, --extra-config <path>
        Read this file after configuration files are read from either
        default locations or from files specified by the --config
        option.
```

Note

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

2.5 Postinstallation Testing

In this section, we will see how to test your MySQL Router installation by setting up the Router to perform simple routing. In this case, the Router will act as an intermediate node redirecting client connections to a list of servers. If one server fails, clients will be redirected to the next available server in the list.

To illustrate this concept, we will use a simple Python script and a set of three MySQL Servers. Ordinarily, these servers would be setup in a replication topology but for this example we want to test simple routing. Thus, the servers are not setup in a replication topology. Thus, this test is only a test and not indicative of an actual application or use case for MySQL Router

Setup the Servers

Begin by starting three MySQL Servers. You can do this in a variety of ways from using the `mysql-test-run.pl` script, installing three servers on three machines or the same machine, or using MySQL Utilities to clone a running server. The following shows how to start three servers cloned from the original server. The original server is listening on port 3306. The cloned servers will be running on the localhost and listening on ports 13001, 13002, and 13003 respectfully.

```
shell>  mysqlserverclone --source=root:secret@localhost --new-data=./test_13001 --new-port=13001 --root=se
# Cloning the MySQL server located at localhost:3306
# Creating new data directory...
# Configuring new instance...
# Locating mysql tools...
# Setting up empty database and mysql tables...
# Starting new instance of the server...
# Testing connection to new instance...
# Success!
# Setting the root password...
# Connection Information:
#  -uroot -psecret --socket=/test_13001/mysql.sock
#...done.
shell>  mysqlserverclone --source=root:secret@localhost --new-data=./test_13002 --new-port=13002 --root=se
# Cloning the MySQL server located at localhost:3306
# Creating new data directory...
# Configuring new instance...
# Locating mysql tools...
# Setting up empty database and mysql tables...
# Starting new instance of the server...
# Testing connection to new instance...
# Success!
# Setting the root password...
# Connection Information:
#  -uroot -psecret --socket=/test_13002/mysql.sock
#...done.
shell>  mysqlserverclone --source=root:secret@localhost --new-data=./test_13003 --new-port=13003 --root=se
# Cloning the MySQL server located at localhost:3306
# Creating new data directory...
# Configuring new instance...
# Locating mysql tools...
# Setting up empty database and mysql tables...
# Starting new instance of the server...
# Testing connection to new instance...
# Success!
# Setting the root password...
# Connection Information:
#  -uroot -psecret --socket=/test_13003/mysql.sock
#...done.
```

Note

For a complete explanation of how to use `mysqlserverclone`, see `mysqlserverclone` — Clone Existing Server to Create New Server.

Next, we make a minor change to each server so that we can see the redirect in action. More specifically, we create a different database on each server to uniquely identify each server as shown.

```
shell>  mysql -uroot -p -h 127.0.0.1 --port=13001 -e "CREATE DATABASE test_13001"
shell>  mysql -uroot -p -h 127.0.0.1 --port=13002 -e "CREATE DATABASE test_13002"
shell>  mysql -uroot -p -h 127.0.0.1 --port=13003 -e "CREATE DATABASE test_13003"
```

As you can see, we created a database with the port of the server in the name. This way, we can tell to which server the Router has sent the client.

Set Up the Router

Next, we set up MySQL Router to redirect to these servers. We must set up a configuration file to tell the Router how to route the connections. The following shows a simple configuration to match this test.

A complete explanation of the configuration file and all of the options is describe in a later chapter. For now, we will use a simplified configuration.

```
[DEFAULT]
logging_folder =

[logger]
level = INFO

[routing:basic_redirect]
bind_port = 7001
mode = read-write
destinations = localhost:13001,localhost:13002,localhost:13003
```

The first section, `DEFAULT`, lists the paths to the Router installation and MySQL server. These paths are common for most Unix and Linux platforms (e.g. Ubuntu) but other platforms may differ. Be sure to change these to match your system. We left the optional `logging_folder` blank, in order to view logger output in the console. That is also default behavior.

The second section, `logger`, is used to set the logging level. In this case, we turn on basic logging to the console.

The third section, `routing:basic_redirect`, is where the magic happens. Here we see the Router is instructed to listen on the localhost and port 7001 using the `bind_port` setting. We turn on `read-write` mode for reading and writing. The `destinations` option is where we specify the list of servers to use in the redirect. In this case, we see servers on the localhost at ports 13001, 13002, and 13003.

What this means is the client will connect to the Router on port 7001 and it will redirect client connections to the first available server in the `destinations` list.

If you are following along, save this file to a file named `my_router.ini`.

Now, let us start the Router. For this, we tell the Router which configuration file to read as shown below.

```
shell>   mysqlrouter -c my_router.ini
2015-10-20 19:50:08 INFO    [7f9631018700] routing:basic_failover started: listening on localhost:7001; read-w
```

Here we see the Router has started and is waiting for connections on port 7001.

Testing the Router

Finally, let us test the Router. We will start a simple client script using Python and connect to the Router. While the script is running, we kill one of the servers then test how the Router redirects.

Begin by launching a Python console and entering the following statements.

```
```

```
shell>  python
Python 2.7.9 (default, Apr  2 2015, 15:33:21)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import mysql.connector
>>> cnx = mysql.connector.connect(host='localhost',port=7001,user='root',password='secret')
>>> cur = cnx.cursor()
>>> cur.execute("SHOW DATABASES")
>>> print cur.fetchall()
[(u'information_schema',), (u'mysql',), (u'performance_schema',), (u'test_13001',)]
```

Notice we connected to the Router on port 7001. When we executed a simple query, `SHOW DATABASES`, we see the result includes the `test_13001` database and therefore we know the Router has redirected the client to the server on port 13001 as shown in the configuration file.

Next, we shutdown the server running on port 13001 as follows.

Note

If you don't want to use the database name method, you can also execute the query, `SHOW VARIABLES LIKE 'port'` to get the port for the connected server.

```
shell>  mysqladmin shutdown -uroot -p -h 127.0.0.1 --port=13001
```

Next, return to the Python console and attempt to run the query again. What you should see is an error stating the connection has dropped. Indeed, this is the case because the client was connected (redirected via the Router) to the server running on port 13001, which has just been shutdown.

```
>>> cur.execute("SHOW DATABASES")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/dist-packages/mysql/connector/cursor.py", line 511, in execute
    self._handle_result(self._connection.cmd_query(stmt))
  File "/usr/local/lib/python2.7/dist-packages/mysql/connector/connection.py", line 486, in cmd_query
    result = self._handle_result(self._send_cmd(ServerCmd.QUERY, query))
  File "/usr/local/lib/python2.7/dist-packages/mysql/connector/connection.py", line 270, in _send_cmd
    return self._socket.recv()
  File "/usr/local/lib/python2.7/dist-packages/mysql/connector/network.py", line 228, in recv_plain
    raise errors.InterfaceError(errno=2013)
mysql.connector.errors.InterfaceError: 2013: Lost connection to MySQL server during query
```

If this were an actual application, the developer would have written the code to catch errors like this and retry the connection. Another valid technique is to connect, execute, then disconnect which will ensure the Router will redirect without having to detect the error (but good error handling is always advisable and prudent).

Since this is only a test, we can simply reconnect to the Router, which will redirect us to the next server in the list. Reenter the same commands as before to connect and run the same query.

```
>>> cnx = mysql.connector.connect(host='localhost',port=7001,user='root',password='secret')
>>> cur = cnx.cursor()
>>> cur.execute("SHOW DATABASES")
>>> print cur.fetchall()
[(u'information_schema',), (u'mysql',), (u'performance_schema',), (u'sys',), (u'test_13002',)]
```

Notice this time the Router has redirected the client to the server running on port 13002, as intended.

If you then shutdown that server and reconnect, you will see the Router has redirected the client to the last server in the list.

```
>>> cnx = mysql.connector.connect(host='localhost',port=7001,user='root',password='secret')
>>> cur = cnx.cursor()
>>> cur.execute("SHOW DATABASES")
>>> print cur.fetchall()
[(u'information_schema',), (u'mysql',), (u'performance_schema',), (u'sys',), (u'test_13003',)]
```

We have now demonstrated the Router in action performing simple redirects to a list of servers.

Chapter 3 Configuration

Table of Contents

3.1 Configuration File Locations 21
3.2 Configuration File Setup 22
3.2.1 General 23
3.2.2 Connection Routing (Standalone) 23
3.2.3 Logging 26
3.2.4 Configuration File Example 27

The format of the configuration file resembles the traditional INI file format with sections and options.

3.1 Configuration File Locations

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

Default Configuration File Locations

By default, MySQL Router scans specific locations for its configuration files.

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.

```
shell> mysqlrouter --help

MySQL Router v2.0.3 on Linux (64-bit) (GPL community edition)
Copyright (c) 2015, Oracle and/or its affiliates. All rights reserved.

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

Start MySQL Router.

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

  Usage: mysqlrouter [-v|--version] [-h|--help]
                     [-c|--config=<path>]
                     [-a|--extra-config=<path>]
  ...
```

Important

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

On Linux and OS X, MySQL Router scans the following locations, although these locations are system dependent:

1. `/etc/mysqlrouter/mysqlrouter.ini`

Note

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

2. `$HOME/.mysqlrouter.ini`

User-Defined and Extra Configuration Files

Two command line options help control these configuration file locations:

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

  Only one `-c, --config` configuration file can be loaded at any given time.

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

For example:

```
shell> mysqlrouter -c /custom/path/to/router.ini -a /another/config.ini
```

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 `--extra-config` options. For example:

```
shell> mysqlrouter -a a.ini -c b.ini -a c.ini -a d.ini
```

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

Each loaded configuration file will override configuration settings from previously read files.

Default Configuration File Locations (Linux)

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

- Generic Linux (standalone-layout) : `./mysqlrouter.ini`
- Default, installing under `/usr/local` : `/usr/local/etc/mysqlrouter.ini`
- RPM and Debian : `/etc/mysqlrouter/mysqlrouter.ini`

3.2 Configuration File Setup

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.

Note

An example router ini file is added to the system's share/doc/ directory for MySQL Router.

Below are the available configuration options divided by section, with an example configuration file to follow. There is one additional section specific to using the Fabric integration connection routing with the Fabric Cache plugin. These options are discussed in the Fabric Cache Plug-in section below.

3.2.1 General

The following options are available for the general section identified by `[DEFAULT]`.

Folder Paths

Information related to the folder options, and setting these is optional.

- `logging_folder`: 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, or not setting it, sends the messages to the console (**stdout**).

  Default value: ""

- `plugin_folder`: 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`: path to the MySQL Router runtime files.

  Default value: `/usr/local/`

- `config_folder`: 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 there own configuration files.

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

An example `mysqlrouter.ini` file that relies on most defaults, and is set to send logs to `/var/log/mysqlrouter/mysqlrouter.log`:

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

3.2.2 Connection Routing (Standalone)

The following options are available to routing strategy sections identified by `[routing:<section_key>]`.

Bind Address

Information related to the `bind_address` option:

- Routing entries can bind 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.

- Binding to a specific IPv4 or IPv6 address allows and ensures that MySQL Router is not starting and routing the service on a 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) will bind to all network interfaces (IPs) on the host. You can also use IPv6 addresses.

```
bind_address = 127.0.0.1:7001
```

Note

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

Bind Port

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

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

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

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

```
[routing:example_3]
bind_port = 8001
```

```
[routing:example_4]
bind_address = 127.0.0.1:7001
```

Connect Timeout

Information related to the `connect_timeout` option. This is used by the routing plugin when connecting to the destination MySQL server. The default value is 1 second. The value cannot be unlimited, and an invalid value results in a configuration error. The valid range is between 1 and 65536. You should keep this value low.

For example, when using `read-write` mode, the value can be a bit higher if you want to wait for the server (Master) to become available. When using `read-only` mode, connecting to a slave it is good to use a lower value since the Router will select a new server during connection routing.

```
connect_timeout = 1
```

Destinations

Information related to the `destinations` option:

- Provides a comma-separated list of destination addresses that should be used when establishing connections. The default port is 3306.

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

- Behavior depends on the mode option.

Modes

Information related to the `mode` option. Setting this parameter is **required**, and each mode has different scheduling. Two modes are supported:

- **read-write**: Typically used for routings to a MySQL master.

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

  The first successful MySQL server contacted is saved in memory as the first to try for future incoming connections. This is a temporary state, in that it won't be remembered after MySQL Router is restarted.

```
[routing:example_strategy]
bind_port = 7001
destinations = master1.example.com,master2.example.com,master3.example.com
mode = read-write
```

- **read-only**: Typically used for routings to a MySQL slave.

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

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

  Unavailable MySQL server's are quarantined. Their availability is checked, and when available they are put back on the available *destinations* list. The destinations order is maintained.

```
[routing:ro_route]
bind_port = 7002
destinations = slave1.example.com,slave2.example.com,slave3.example.com
mode = read-only
```

Note

Both modes are available for connection routing with or without MySQL Fabric (standalone or Fabric integration).

Max Connections

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

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

```
max_connections = 512
```

Max Connection Errors

Information related to the `max_connect_errors` option. The default value is 100, and the valid range is between 1 and 2^32 (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 Router attempts to discover if connection related errors are present.

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

```
max_connect_errors = 100
```

Note

This option was added in Router 2.0.3.

Client Connection Timeout

Information related to the `client_connect_timeout` option. The default value is 9, which is one less than the MySQL 5.7 default. The valid range is between 2 and 31536000.

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

```
client_connect_timeout = 9
```

Note

This option was added in Router 2.0.3.

3.2.3 Logging

The following options are available for the logging section identified by `[logger]`

Logger

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

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

Note

The MySQL Harness plugin also supports logging modes of *WARNING*, *ERROR*, and *FATAL*, but these are not available in MySQL Router 2.0.2.

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

By default, level=INFO is set and used if [logger] is not set in your configuration file.

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 *[DEFAULTS]* section, and for additional information see Folder Paths.

3.2.4 Configuration File Example

Here is a basic example using connection routing to multiple masters. In this setup, all traffic will go to a.example.com, unless it is down (fails) then traffic will go to the next address on the list, which is b.example.com. If that fails, then c.example.com. This behavior is dictated by the `read-write` mode.

The section key (we use "failover" below) is optional, but such descriptions are helpful for debugging, and it allows multiple configuration sections for the plugin.

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

[logger]
level = DEBUG

[routing:failover]
bind_port = 7001
mode = read-write
destinations = a.example.com,b.example.com,c.example.com
```

In a similar example, we change the mode from `read-write` to `read-only` to implement a form of round-robin load-balancing:

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

[logger]
level = DEBUG

[routing:balancing]
bind_port = 7002
mode = read-only
destinations = x.example.com,y.example.com,z.example.com
```

In that case, the connections are distributed over the available servers (destinations) in a round-robin fashion rather than sending all connections to the first server.

Chapter 4 MySQL Router Application

Table of Contents

4.1 User Options 29
4.2 Starting the Router 29
4.3 Using the Logging Feature 30

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

4.1 User Options

There are a number of options available for controlling the application. Each is listed below with a brief explanation.

- `[-v|--version]` : displays the version number of the application and exits.
- `[-h|--help]` : displays the list of options and additional information about the application and exist.
- `[-c|--config=$lt;path$gt;]` : 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.
- `[-a|--extra-config=$lt;path$gt;]` : 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.

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. The following excerpt of the `--help` output shows an example from a Ubuntu 15.04 machine.

```
shell>  mysqlrouter --help

MySQL Router v2.0.3 on Linux (64-bit) (GPL community edition)
Copyright (c) 2015, Oracle and/or its affiliates. All rights reserved.

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

Start MySQL Router.

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

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

4.2 Starting the Router

Recall the Router requires a configuration file to setup the routing strategies and to provide paths for its plugins as well as other controls such as logging. Recall also that the Router will search a predetermined

list of default paths for some platforms. However, the best way to start the Router is to provide the configuration file with the `--config=` option. An example follows.

```
shell>  mysqlrouter --config=/path/to/file/my_router.ini
2015-10-22 10:51:34 INFO    [7f5f66768700] routing:basic_redirect started: listening on localhost:7001; read-w
2015-10-22 10:51:34 INFO    [7f5f65f67700] routing:read_only_redirect started: listening on localhost:7002; re
```

Here we see the Router has started as a console application and has reported its routing strategies. Notice that the Router is listening on two ports and that the name of each routing strategy is listed.

For example, the first strategy is listed as `routing:basic_redirect`, is listening on port `7001` and its mode is `read-write`. The corresponding section in the configuration file is shown below.

```
[routing:basic_redirect]
bind_address = localhost:7001
mode = read-write
destinations = localhost:13001,localhost:13002,localhost:13003
```

Notice that the name, port, and mode were taken directly from the configuration file. In this way, you can quickly determine which routing strategies are active. This could be particularly handy if you are running several instances of the Router or are using the extra configuration file option.

Similarly, the second strategy is listed as `routing:read_only_redirect`, is listening on port `7002` and its mode is `read-only`. The corresponding section in the configuration file is shown below.

```
[routing:read_only_redirect]
bind_address = localhost:7002
mode = read-only
destinations = localhost:3006,localhost:3307
```

To stop the Router, issue a **CTRL** + **C** in the console or issue a `SIGTERM`signal to stop the application. For example, you could issue the following command to send the `SIGTERM` to the application. Note that we use the `.pid` file for the application.

```
shell>  kill -SIGTERM <pid>
```

In this case, the <pid> is the process id of the `mysqlrouter` application.

4.3 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, you must turn it on in the configuration file under a section named `logging`. An example is shown below.

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

By default, all logs are sent to the console. Alternatively, you can optionally store the log in a file. To do so, specify a `logging_folder` path for the logging file in the `DEFAULTS` section in the configuration file as shown below. The logging file will be named `mysqlrouter.log`.

```
[DEFAULT]
logging_folder = /path/to/folder
plugin_folder = /usr/local/lib/mysqlrouter
```

```
config_folder = /etc/mysql
runtime_folder = /var/run
```

When logging to a file, the console output is suppressed and all messages go to the file except for a reminder notice as shown below. Thus, you can either log messages to the console, which is the default, or specify a logging path in the configuration file to log to a file.

```
Logging to /path/to/folder/mysqlrouter.log
```

There are two common logging modes:

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

The following example shows what the messages would look like for a logging mode of `DEBUG`. Notice both `INFO` and `DEBUG` messages are shown.

```
2015-10-22 11:21:07 INFO    [7fdd091f8700] routing:read_only_redirect started: listening on localhost:7002
2015-10-22 11:21:07 INFO    [7fdd119f9700] routing:basic_redirect started: listening on localhost:7001; rea
2015-10-22 11:27:09 DEBUG   [7fdd0bfff700] Trying server localhost:13001 (index 0)
2015-10-22 11:27:09 DEBUG   [7fdd0bfff700] routing:basic_redirect [127.0.0.1]:49661 - [127.0.0.1]:13001
2015-10-22 11:27:41 DEBUG   [7fdd0bfff700] Routing stopped (up:301b;down:201b)
2015-10-22 11:28:07 DEBUG   [7fdd0bfff700] Trying server localhost:13001 (index 0)
2015-10-22 11:28:07 DEBUG   [7fdd0bfff700] MySQL Server localhost:13001: Connection refused (111)
2015-10-22 11:28:07 DEBUG   [7fdd0bfff700] Trying server localhost:13002 (index 1)
2015-10-22 11:28:07 DEBUG   [7fdd0bfff700] routing:basic_redirect [127.0.0.1]:49663 - [127.0.0.1]:13002
```

This example was taken from a log file rather than console output. Notice in the debug statements we see that the routing redirects were recorded. Thus, if you want to see when these occur, you may want to use the `DEBUG` logging mode.

The next chapter discusses the plugins used with MySQL Router.

Chapter 5 Plugins

Table of Contents

5.1 Connection Routing Plugin 33
5.2 Fabric Cache Plug-in 33

Introduction to MySQL Router plugins.

Each plugin is loaded and configured from the configuration file, and then started by the *Router Harness*.

5.1 Connection Routing Plugin

The *Connection Routing* plugin performs connection-based routing, meaning that it forwards the packets to the server without looking at them. This is a simplistic approach that provides a high throughput.

Note

MySQL Fabric usage is optional, and the example below does not use Fabric.

A simple connection-based routing setup is shown below. These options were documented in the Configuration File Setup chapter.

```
[DEFAULT]
logging_folder = /var/log/mysql/router
config_folder  = /usr/local/etc/mysqlrouter
plugin_folder  = /usr/local/lib/mysqlrouter
runtime_folder = /usr/local/

[logger]
level = INFO

[routing]
bind_address = 127.0.0.1:7002
destinations = slave1.example.com,slave2.example.com,slave3.example.com
mode = read-only
```

Here we use connection routing to round-robin MySQL connections to three MySQL servers. The read-only mode causes round-robin behavior, and logs are sent to `/var/log/mysql/router/mysqlrouter.log`.

5.2 Fabric Cache Plug-in

The *Fabric Cache* plug-in maintains the connection with the MySQL Fabric instance, and caches the connection information for the servers under Fabric control.

Note

This section describes a plugin for accessing information from a Fabric installation. See the online reference manual for MySQL Fabric for additional information about Fabric installation and configuration.

Fabric Cache Destinations

The Fabric cache destination is a special string used to define the connection to Fabric. It is a uniform resource identifier (URI). The URI is specified with the *destinations* option for the routing strategy in the

configuration file. The following presents an example and explanation of each part. They must appear in the order shown.

```
destinations = plugin://section_key/target/group_id
```

- *plugin* : This string instructs the router to use a particular plugin. Currently, the only plugin available is the Fabric Cache plugin for Fabric integrated connection routing. Thus, we use the string *fabric+cache*.
- *section_key* : The section key for the Fabric Cache plugin configuration options. For example, `[fabric_cache:web]`. Thus, your configuration file will contain another section with this key.
- *target* : The type of Fabric object we are request. Currently, the only value is *group*.
- *group_id* : The Fabric HA group identifier.

The following shows a complete example of the entries in the configuration file. Notice how the parts match for the

```
[fabric_cache:web]
address = 192.168.14.29
user = routeruser

[routing:web]
bind_port = 7001
destinations = fabric+cache://web/group/webforum
mode = read-write
```

This URI is used to retrieve the list of MySQL servers through the Fabric Cache plugin, which stores and manages information fetched from MySQL Fabric. When Fabric becomes unavailable, Fabric Cache will keep trying to connect to Fabric will keep retrying regularly to connect to Fabric.

A Fabric Cache section without a key means that the "fabric+cache" URI is used with an empty authority.

Destinations read from different locations are configured using the `destinations` option. The Routing Plugin determines if a URI or a comma-separated value should be used.

Only one Fabric Cache plug-in can be active. An empty "host" part of the URI means the first Fabric Cache plugin without a key is used. Specifying a host in the URI will result in an error, as demonstrated here:

```
[fabric_cache]
...
[routing:production_rw]
...
mode = read-write

# Works, no host
destinations = fabric+cache:///hagroup/dev1

# Fails, a host is specified
destinations = fabric+cache://cache_2/hagroup/dev1
```

With Fabric, *allow_primary_reads* can not be passed to Fabric in read-write mode, as it only functions with read-only mode.

Additional Fabric Cache examples:

```
[fabric_cache:fabric1]
```

```
address = fabric1.example.com
user = fabric_user

[routing:homepage_rw]
mode = read-write
destinations = fabric+cache://fabric1/group/homepage

[routing:homepage_ro]
mode = read-only
destinations = fabric+cache://fabric1/group/homepage?allow_primary_reads=yes

[routing:homepage_ro]
mode = read-only
destinations = fabric+cache:///group/homepage   # empty authority
```

Chapter 6 Use Cases and Examples

MySQL Router use cases and examples.

HA with a Connector

One basic use-case provides MySQL Fabric support to legacy connectors. Here we create a configuration file named `fabric_ha.ini` with sections for the *Fabric Cache* and *Connection Router* plugins.

```
[DEFAULT]
logging_folder =
plugin_folder = /usr/local/lib/mysqlrouter
config_folder = /etc/mysql
runtime_folder = /var/run

[logger]
level = INFO

[fabric_cache:my_cache]
address = fabric.example.com
user = admin

[routing]
bind_address = 127.0.0.1:7002
destinations = fabric+cache://my_cache/group/my_group
mode = read-write
```

- Our *[DEFAULT]* section defines default paths to several folders.
- Our *[logger]* instructs the logger to log information messages.
- Our *[fabric_cache:my_cache]* section enables and configures the *Fabric Cache* plugin. We fetch the farm data from the MySQL Fabric instance at fabric.example.com and cache it into memory. It assigns a key named *my_cache* to the section so that it can be referenced later.
- Our *[routing]* section enables and configures connection routing to listen (bind to) port 7002 on localhost in order to fetch destination information from the Fabric Cache given by the key *my_cache*, and route the packets to the primary in the high-availability (HA) group named *my_group*.

Now, start MySQL Router loading our configuration file:

```
shell> mysqlrouter --config=fabric_ha.ini
```

You will be prompted for the Fabric `user` password, and after this you can connect to the bind address of the router using a normal MySQL connector or client.

Appendix A MySQL Router Frequently Asked Questions

A.1 Where do I install MySQL Router? 39
A.2 Can I run more than one instance of the router application? 39
A.3 How do I make the router application highly available? 39
A.4 Does the router inspect packets? 39
A.5 Does the router impact performance? 39
A.6 Can I bind the router to multiple IP addresses? 39
A.7 What is the difference between round-robin and first-available scheduling modes? 39

A.1. Where do I install MySQL Router?

You should install the router on the same machine as your application.

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

Yes.

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

Currently, there are no high availability features for the router application. You can use a script or similar mechanism to monitor the router and restart it if needed provided the router has not stopped due to no more servers available to redirect.

A.4. Does the router inspect packets?

No.

A.5. Does the router impact performance?

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

A.6. 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.7. What is the difference between round-robin and first-available scheduling modes?

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

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

Appendix B MySQL Router Unit Tests

Testing MySQL Router requires the appropriate flags during the compilation step. While Google Mock and Google Test are bundled to execute the tests, your system requires GCOV installed to optionally execute the code coverage tests. Change the paths and commands according to your needs.

```
shell> cd ~/src
shell> tar xfvz mysqlfabric-2.0.3.tar.gz
shell> cd 2.0.3
shell> mkdir buildtests
shell> cd buildtests
shell> cmake .. -DENABLE_GCOV=yes -DENABLE_TESTS=yes
shell> make
```

Now, execute the unit tests:

```
shell> make test
```

Optionally, also execute a code coverage report.

Note

This requires GCOV

```
shell> sh ../tests/gcoverage.sh -o /path/to/html/report

shell> firefox /path/to/html/report/index.html
```