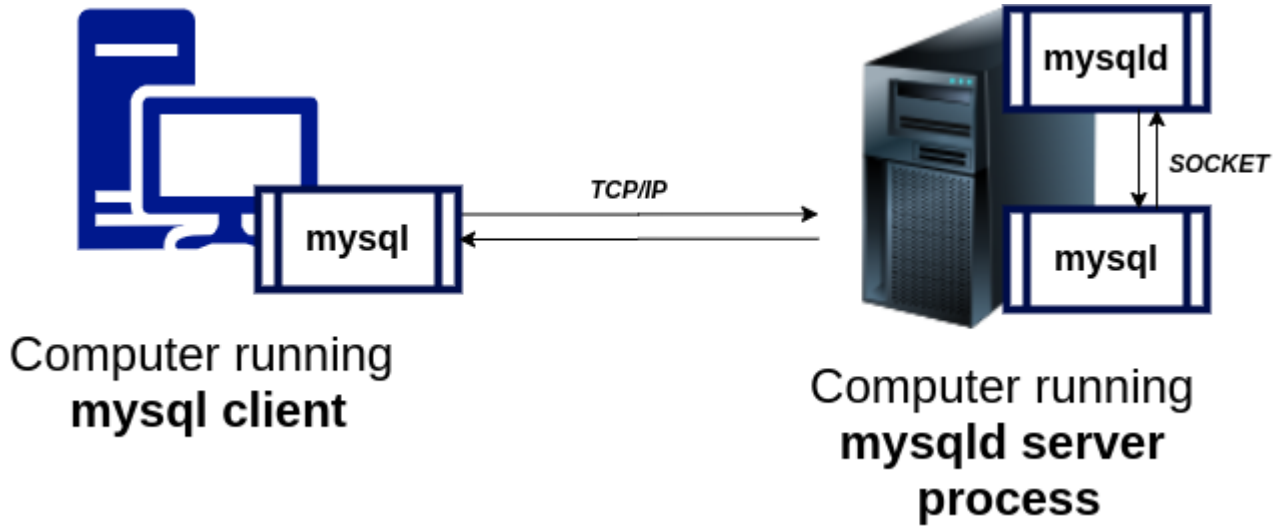
Omkar Manjrekar · Follow
Aug 2, 2020 · 10 min read

Running multiple MySQL instances on Ubuntu

MySQL uses a client-server architecture. The client program `mysql` communicates with the server program `mysqld`. The server program usually starts (as a service named mysql) on startup of our *Ubuntu* (or *Windows*) system and keeps running in the background all the time.



The figure shows mysql client communicating with mysqld server over TCP (when on separate machines) and over unix SOCKET (when on the same machine).

To access the server using the client program **mysql** we can run this command:

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

This is assuming the server **mysqld** is running on our system at ***host***: ***localhost*** and ***port: 3306,*** which is the default.

So how do we go about setting up a separate server or running multiple instances? Let's find

1. What a **MySQL** server needs (at the least) to run?
2. Creating these things (if required).
3. And providing it to MySQL

A lil backstory before we start: I wanted to implement MySQL master slave database replication and to test the setup I thought why not create another server on the same machine. Easy right? Or so I thought. But for some or the other reason the server would pick up wrong configuration and debugging became difficult due to mixed timestamps in log file. I spent the next 1 day (about ~14 hours) understanding how to run another MySQL server by reading from the official documentation. This post is an effort to make that easy for others (and avoid them having to go through hell). *WHY should setting up another MySQL instance be difficult anyways?*

PS: The system configuration that I had used for this post was Ubuntu 16.04 with MySQL 5.7 and MySQL 8.0.21. Even though the specifics of the commands may change with the Operating System, the procedure will still be the same.

1. What MySQL server needs at the least to run?

Just two things. A **place to store data (data dir)** and a **medium** (a **socket** (and a **TCP port**)) for a client to connect to the server. That's it.

2. Creating these things

2.1. Data directory

MySQL stores all of its data in the data directory. The default data directory is `/var/lib/mysql` . If you are curious to find out how does the directory look run this command on your machine:

```
$ ls -l /var/lib/mysql/
```



```
-rw-r----- 1 mysql mysql      544 Jul 19 12:07 binlog.index
-rw------- 1 mysql mysql     1676 Nov 29  2019 ca-key.pem
-rw-r--r-- 1 mysql mysql     1112 Nov 29  2019 ca.pem
-rw-r--r-- 1 mysql mysql     1112 Nov 29  2019 client-cert.pem
-rw------- 1 mysql mysql     1680 Nov 29  2019 client-key.pem
-rw-r--r-- 1 mysql mysql        0 May  9 09:53 debian-5.7.flag
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 elsi_master_2105
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 elsi_tbt_2105
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 eyantra_iot
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 eysip
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 flo
-rw-r----- 1 mysql mysql   196608 Jul 19 23:39 #ib_16384_0.dblwr
-rw-r----- 1 mysql mysql  8585216 Jul 12 01:56 #ib_16384_1.dblwr
-rw-r----- 1 mysql mysql     5219 Jul 14 02:31 ib_buffer_pool
-rw-r----- 1 mysql mysql 79691776 Jul 19 23:39 ibdata1
-rw-r----- 1 mysql mysql 50331648 Jul 19 23:39 ib_logfile0
-rw-r----- 1 mysql mysql 50331648 Jul 19 23:39 ib_logfile1
-rw-r----- 1 mysql mysql 12582912 Jul 15 01:40 ibtmp1
drwxr-x--- 2 mysql mysql     4096 Jul 15 01:40 #innodb_temp
-rw-r----- 1 mysql mysql     1604 Jul 12 13:54 kirito.err
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:56 mysql
-rw-r----- 1 mysql mysql      361 Jul 11 11:57 mysql-bin.index
-rw-r--r-- 1 root  root       134 Jul 12 02:09 mysqld_multi.log
-rw-r----- 1 mysql mysql 29360128 Jul 19 23:39 mysql.ibd
-rw-r--r-- 1 mysql mysql        6 Jul 12 01:57 mysql_upgrade_info
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 payment
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:54 performance_schema
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 personal
-rw------- 1 mysql mysql     1676 Nov 29  2019 private_key.pem
-rw-r--r-- 1 mysql mysql      452 Nov 29  2019 public_key.pem
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 rpg
-rw-r--r-- 1 mysql mysql     1112 Nov 29  2019 server-cert.pem
-rw------- 1 mysql mysql     1680 Nov 29  2019 server-key.pem
drwxr-x--- 2 mysql mysql    12288 Jul 12 01:53 sys
drwxr-x--- 2 mysql mysql     4096 Jul 12 01:53 testing_slave
-rw-r----- 1 mysql mysql 12582912 Jul 19 23:39 undo_001
-rw-r----- 1 mysql mysql 12582912 Jul 19 23:38 undo_002
```

You will see a bunch of files and folders, including the folders storing the databases you created. In the output above, I have highlighted the databases I created and the ones created by MySQL at initialization.

So how do we "*create*" and "*initialize*" a NEW data directory?

```
$ mkdir -p /opt/second_server_data/
```

2.1.2. Next is initializing the directory.

First change permissions on the created directory so that MySQL is able to create files or folders the way it wants. We will copy permissions from existing data directories. Alternatively, one can change the **user** and the **group** for the data directory to ***mysql***.

```
$ chmod --reference /var/lib/mysql /opt/second_server_data/
$ chown --reference /var/lib/mysql /opt/second_server_data/
```

On some systems like Ubuntu, file-system access to programs (in our case **mysqld, mysqld_safe, …**) can be restricted by other mechanisms such as **AppArmor**. **AppArmor** is a linux kernel security module that allows you to restrict program capabilities on a per-program basis. We need to **add rules** to the policy file to **stop the operating system from restricting MySQL to access** our newly created data directory, socket files, etc.

```
$ sudo nano /etc/apparmor.d/usr.sbin.mysqld
```

Put the content anywhere **within opening and closing braces**.

```
/opt/second_server_data/ r,
/opt/second_server_data/** rwk,

/var/run/mysqld/second_server.pid rw,
/var/run/mysqld/second_server.sock rw,
/var/run/mysqld/second_server.sock.lock rw,

/run/mysqld/second_server.pid rw,
/run/mysqld/second_server.sock rw,
/run/mysqld/second_server.sock.lock rw,
```

PS: If you see the file, similar rules will exist for the default mysql server.

```
$ mysqld --initialize --user=mysql --datadir=/opt/second_server_data/
```

Once done you can check the contents of the directory.

```
$ sudo ls -l /opt/second_server_data/
```

```
total 164288
-rw-r----- 1 mysql mysql       56 Jul 20 01:14 auto.cnf
-rw------- 1 mysql mysql     1676 Jul 20 01:14 ca-key.pem
-rw-r--r-- 1 mysql mysql     1112 Jul 20 01:14 ca.pem
-rw-r--r-- 1 mysql mysql     1112 Jul 20 01:14 client-cert.pem
-rw------- 1 mysql mysql     1676 Jul 20 01:14 client-key.pem
-rw-r----- 1 mysql mysql   196608 Jul 20 01:15 #ib_16384_0.dblwr
-rw-r----- 1 mysql mysql  8585216 Jul 20 01:14 #ib_16384_1.dblwr
-rw-r----- 1 mysql mysql     5564 Jul 20 01:15 ib_buffer_pool
-rw-r----- 1 mysql mysql 12582912 Jul 20 01:15 ibdata1
-rw-r----- 1 mysql mysql 50331648 Jul 20 01:15 ib_logfile0
-rw-r----- 1 mysql mysql 50331648 Jul 20 01:14 ib_logfile1
drwxr-x--- 2 mysql mysql     4096 Jul 20 01:15 #innodb_temp
drwxr-x--- 2 mysql mysql     4096 Jul 20 01:14 mysql
-rw-r----- 1 mysql mysql 25165824 Jul 20 01:15 mysql.ibd
drwxr-x--- 2 mysql mysql     4096 Jul 20 01:14 performance_schema
-rw------- 1 mysql mysql     1680 Jul 20 01:14 private_key.pem
-rw-r--r-- 1 mysql mysql      452 Jul 20 01:14 public_key.pem
-rw-r--r-- 1 mysql mysql     1112 Jul 20 01:14 server-cert.pem
-rw------- 1 mysql mysql     1680 Jul 20 01:14 server-key.pem
drwxr-x--- 2 mysql mysql     4096 Jul 20 01:14 sys
-rw-r----- 1 mysql mysql 10485760 Jul 20 01:15 undo_001
-rw-r----- 1 mysql mysql 10485760 Jul 20 01:15 undo_002
```

NOTE: When the data directory contents are generated, **mysqld** will also generate **default root password** which automatically goes to error log. You can find it either at `/var/log/syslog` or at `/var/log/mysql/error.log`

```
2020-07-19T19:44:00.835128Z 0 [System] [MY-013169] [Server]
/usr/sbin/mysqld (mysqld 8.0.20) initializing of server in progress as
process 15603
```

```
password is generated for root@localhost: pYo*3Epssb?g
```

PS: A more sophisticated way to find where the error log is stored is by looking up server system variables. These store information about configuration. After loginning to your base mysql server, type:

```
mysql> SHOW VARIABLES LIKE 'log_error';
```

```
+---------------+---------------------------+
| Variable_name | Value                     |
+---------------+---------------------------+
| log_error     | /var/log/mysql/error.log |
+---------------+---------------------------+
1 row in set (0.00 sec)
```

2.2. Socket (and Port)

Given correct permissions MySQL is able to create these on its own. Easy right!? :P

3. Providing these things to MySQL server program

Now that we created the required resources to **run a MySQL server** we need to provide these to it as configuration. MySQL server program **mysqld** or most other programs that are shipped with MySQL can be configured using command-line options, option files or configuration files (.cnf) or server system variables.

Lets see the two frequently used methods, *command-line* and *option files*.

3.1. Command-line

Lets run the MySQL server using command line options using the program **mysqld_safe.**

```
sudo mysqld_safe --no-defaults --datadir=/opt/second_server_data/ --
port=3315 --mysqlx=0 --socket=/var/run/mysqld/second_server.sock
```

in the command above are straight-forward.

1. **no-defaults:** Makes sure no options are loaded from option files.
2. **data-dir:** Path to the data directory that we created in **2.1.**
3. **port:** TCP port. This is optional and you can do fine with just **socket** (use skip-networking option to disallow TCP).
4. **mysqlx:** X plugin is an interface to allow MySQL function as a document store. Since we don't need it, we will disable it.
5. **socket:** Specify location of socket file

NOTE: For debugging the options mysqld_safe or mysqld program is using, you can use `ps aux | grep mysqld` and verify.

Testing whether it worked

Using MySQL client program to test. We can initiate a connection either over socket or TCP respectively.

```
$ mysql --socket=/var/run/mysqld/second_server.sock -u root -p
```

OR

```
$ mysql -h 127.0.0.1 -P 3315 -u root -p
```

And then enter the password received in the previous step.

This password is auto-generated and needs to be changed before you can perform any operation.

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'second_server';
```

Go ahead and create a database, tables and information into it for testing.



3.2. Option files

We can do the same as above using option files. **Option files** can save one a lot of effort of writing configuration options so its a good place to provide common ones. Option files are stored inside `/etc/mysql/` directory named as `*.cnf`.

At the start of the configuration option there is a group name written inside square brackets. E.g. `[mysql]` , `[mysqld]` , `[mysqld_safe]` . This means that the configuration will only apply to that particular program.

We will create a new file in `/etc/mysql/` named `my.cnf2` with copy the following lines.

```
[mysqld_safe]
server-id       = 2
port            = 3315
socket          = /var/run/mysqld/second_server.sock
datadir         = /opt/second_server_data/
mysqlx          = 0
```

As you must've assumed the group name here is `[mysqld_safe]` since we are using **mysqld_safe** program to run it. Configurations with other group names provided in the same file will not be applied.

To run the server with option file we use **mysqld_safe** with defaults-file option that references the options file we created.

```
$ sudo mysqld_safe --defaults-file=/etc/mysql/my.cnf2
```

That's all about it. Now you have one (default) server running on port 3306 and another on port 3315. In order to create more servers you can repeat the above steps with different *data directory*, *socket* (and *port*) and run them with **mysqld_safe.**

There's one last thing. Starting these servers manually will become a pain as the number of servers grow. To overcome this difficulty mysql provides **mysqld_multi** program.

mysqld_multi

mysqld_multi, as the name suggests, allows running and managing multiple MySQL servers.

It searches for groups named as `[mysqldN]` in file `my.cnf` by default, where **N** is a positive integer. You can use **defaults-file** option to override the file `my.cnf` . Let's create a file `my_multi.cnf` in `/etc/mysql/` with following content. You may create another server following the same steps above.

```
[mysqld1]
server-id       = 1
port            = 3315
socket          = /var/run/mysqld/second_server.sock
datadir         = /opt/second_server_data/
mysqlx          = 0

[mysqld2]
server-id       = 2
port            = 3330
socket          = /var/run/mysqld/mysqld_slave.sock
datadir         = /opt/data/mysql_slave
mysqlx          = 0

[mysqld_multi]
mysqld          = /usr/bin/mysqld_safe
mysqladmin      = /usr/bin/mysqladmin
user            = multi_admin
pass            = multipass
```

While starting the servers is not a problem in order to stop them you'll need to have *SHUTDOWN* permissions on the server. For that create a MySQL user on each server with `username as` **`multi_admin`** and `password as` **`multipass`**`.`

```
$ CREATE USER 'multi_admin'@'%' IDENTIFIED BY 'multipass';
```

And then grant the shutdown permissions.

```
$ sudo mysqld_multi --defaults-file=/etc/mysql/my_multi.cnf --no-log
start
```

There are other operations you can perform such as **stop** to stop a specific or all the servers, **report** to show the status of servers (running/not running), reload to **reload** the configuration file and restart the servers with new configuration.

We can also automate the multiple servers to run at startup by creating a service using **systemd**.

Thanks for reading 👏

This has been my first post on medium (or anywhere…) so thanks a lot for reading readers. If you have any suggestions for me, questions or face any problems in the steps above I am willing to answer them in the comments below. Thanks again, bye!

References

1. https://dev.mysql.com/doc/refman/5.7/en/data-directory-initialization.html
2. https://dev.mysql.com/doc/refman/5.7/en/multiple-unix-servers.html
3. https://dev.mysql.com/doc/refman/8.0/en/mysqld-multi.html
4. https://blogs.oracle.com/jsmyth/apparmor-and-mysql#:~:text=MySQL%20accesses%20files%20in%20various,%2Fvar%2Frun%2Fmysqld.
5. https://dev.mysql.com/doc/refman/5.7/en/server-logs.html
6. https://www.toptal.com/mysql/mysql-master-slave-replication-tutorial
7. https://askubuntu.com/questions/919054/how-do-i-run-a-single-command-at-