Products
Solutions
Developers
Businesses
Pricing

DigitalOcean

Log in

Sign up

Blog

Docs

Get Support

Contact Sales

Tutorials Questions Learning Paths For Businesses Product Docs

CONTENTS

// TUTORIAL //

How To Set Up Replication in MySQL

Updated on June 3, 2021

MySQL

By Mark Drake
Manager, Developer Education

A previous version of this tutorial was written by Etel Sverdlov.

Introduction

When working with databases, it can be useful to have multiple copies of your data. This provides redundancy in case one of the database servers fails and can improve a database's availability, scalability, and overall performance. The practice of synchronizing data across multiple separate databases is called *replication*.

MySQL is a relational database management system, and is the most popular open-source relational database in the world today. It comes installed with a number of built-in replication features, allowing you to maintain multiple copies of your data.

This tutorial outlines how to configure a MySQL instance on one server as a source database and then configure a MySQL instance on another server to function as its replica. It also includes an overview of how MySQL handles replication.

> **Note**: Historically, this type of database replication has been referred to as "master-slave" replication. In a blog post published in July of 2020, the MySQL team acknowledged the negative origin of this terminology and announced their efforts to update the database program and its documentation to use more inclusive language.
>
> However, this is an ongoing process. Although MySQL's documentation and much of the commands in version 8 of the program have been updated to instead refer to the servers in a replication topology as the *source* and its *replicas*, there are places where the negative terminology still appears. This guide will default to the more inclusive *source-replica* terminology wherever possible, but there are a few instances where the older terms unavoidably come up.

Prerequisites

To complete this guide, you will need:

- Two servers running Ubuntu 20.04. Both should have a non-root administrative user with `sudo` privileges and a firewall configured with UFW. Follow our initial server setup guide for Ubuntu 20.04 to set up both servers.
- MySQL installed on each server. This guide assumes that you're using the latest version of MySQL available from the default Ubuntu repositories which, as of this writing, is version 8.0.25. To install this on both servers, follow our guide on How To Install MySQL on Ubuntu 20.04.

Be aware that the procedure outlined in this guide involves designating the MySQL installation on one server as the *source database*, and then configuring the MySQL installation on the other server to be the source's *replica*. To keep things clear, any commands that must be run on the source database's server will have a blue background, like this:

```
source:~$
```
Copy

Likewise, any commands that must be run on the replica MySQL instance's server will have a red background:

```
replica:~$
```
Copy

Lastly, this tutorial includes optional instructions on how to migrate data in an existing database from the source to the replica. This process involves creating a snapshot of the source's database and copying the resulting file to the replica. To do this, we recommend that you set up SSH keys on the source server server and then make sure that the source's public key has been copied to the replica.

Understanding Replication in MySQL

In MySQL, replication involves the source database writing down every change made to the data held within one or more databases in a special file known as the *binary log*. Once the replica instance has been initialized, it creates two threaded processes. The first, called the *IO thread*, connects to the source MySQL instance and reads the binary log events line by line, and then copies them over to a local file on the replica's server called the *relay log*. The second thread, called the *SQL thread*, reads events from the relay log and then applies them to the replica instance as fast as possible.

Recent versions of MySQL support two methods for replicating data. The difference between these replication methods has to do with how replicas track which database events from the source they've already processed.

MySQL refers to its traditional replication method as *binary log file position-based replication*. When you turn a MySQL instance into a replica using this method, you must provide it with a set of binary log coordinates. These consist of the name of the binary log file on the source which the replica must read and a specific position within that file which represents the first database event the replica should copy to its own MySQL instance.

These coordinates are important since replicas receive a copy of their source's entire binary log and, without the right coordinates, they will begin replicating every database event recorded within it. This can lead to problems if you only want to replicate data after a certain point in time or only want to replicate a subset of the source's data.

Binary log file position-based replication is viable for many use cases, but this method can become clunky in more complex setups. This led to the development of MySQL's newer native replication method, which is sometimes referred to as *transaction-based replication*. This method involves creating a global transaction identifier (GTID) for each transaction — or, an isolated piece of work performed by a database — that the source MySQL instance executes.

The mechanics of transaction-based replication are similar to binary log file-based replication: whenever a database transaction occurs on the source, MySQL assigns and records a GTID for the transaction in the binary log file along with the transaction itself. The GTID and the transaction are then transmitted to the source's replicas for them to process.

MySQL's transaction-based replication has a number of benefits over its traditional replication method. For example, because both a source and its replicas preserve GTIDs, if either the source or a replica encounter a transaction with a GTID that they have processed before they will skip that transaction. This helps to ensure consistency between the source and its replicas. Additionally, with transaction-based replication replicas don't need to know the binary log coordinates of the next database event to process. This means that starting new replicas or changing the order of replicas in a replication chain is far less complicated.

Keep in mind that this is only a general explanation of how MySQL handles replication; MySQL provides many options which you can tweak to optimize your own replication setup. This guide outlines how to set up binary log file position-based replication. If you're interested in configuring a different type of replication environment, though, we encourage you to check out MySQL's official documentation .

Step 1 – Adjusting Your Source Server's Firewall

Assuming you followed the prerequisite Initial Server Setup Guide, you will have configured a firewall on both your servers with UFW. This will help to keep both your servers secure, but the source's firewall will block any connection attempts from your replica MySQL instance.

To change this, you'll need to include a UFW rule that allows connections from your replica through the source's firewall. You can do this by running a command like the following **on your source server**.

This particular command allows any connections that originate from the replica server's IP address — represented by `replica_server_ip` — to MySQL's default port number, `3306`:

```
source:~$ sudo ufw allow from replica_server_ip to any port 3306
```

Copy

Be sure to replace `replica_server_ip` with your replica server's actual IP address. If the rule was added successfully you'll see the following output:

```
Output
Rule added
```

Following that, you won't need to make any changes to the replica's firewall rules, since the replica server won't receive any incoming connections and the outgoing connections to the source MySQL server aren't blocked by UFW. You can move on to updating the source MySQL instance's configuration to enable replication.

Step 2 – Configuring the Source Database

In order for your source MySQL database to begin replicating data, you need to make a few changes to its configuration.

On Ubuntu 20.04, the default MySQL server configuration file is named `mysqld.cnf` and can be found in the `/etc/mysql/mysql.conf.d/` directory. Open this file **on the source server** with your preferred text editor. Here, we'll use `nano`:

```
source:~$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

Copy

Within the file, find the `bind-address` directive. It will look like this by default:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
bind-address            = 127.0.0.1
. . .
```

`127.0.0.1` is an IPv4 loopback address that represents **localhost**, and setting this as the value for the `bind-address` directive instructs MySQL to only listen for connections on the **localhost** address. In other words, this MySQL instance will only be able to accept connections that originate from the server where it's installed.

Remember that you're turning your other MySQL instance into a replica of this one, so the replica must be able to read whatever new data gets written to the source installation. To allow this, you must configure your source MySQL instance to listen for connections on an IP address which the replica will be able to reach, such as the source server's public IP address.

Replace `127.0.0.1` with **the source server's** IP address. After doing so, the `bind-address` directive will look like this, with your own server's IP address in place of `source_server_ip`:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
bind-address            =  source_server_ip
. . .
```

Next, find the `server-id` directive, which defines an identifier that MySQL uses internally to distinguish servers in a replication setup. Every server in a replication

environment, including the source and all its replicas, must have their own unique `server-id` value. This directive will be commented out by default and will look like this:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
# server-id             = 1
. . .
```

Uncomment this line by removing the pound sign (`#`). You can choose any number as this directive's value, but remember that the number must be unique and cannot match any other `server-id` in your replication group. To keep things simple the following example leaves this value as the default, `1`:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
server-id               = 1
. . .
```

Below the `server-id` line, find the `log_bin` directive. This defines the base name and location of MySQL's binary log file.

When commented out, as this directive is by default, binary logging is disabled. Your replica server must read the source's binary log file so it knows when and how to replicate the source's data, so uncomment this line to enable binary logging on the source. After doing so, it will look like this:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
log_bin                         = /var/log/mysql/mysql-bin.log
. . .
```

Lastly, scroll down to the bottom of the file to find the commented-out `binlog_do_db` directive:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
.
# bi   g_do_db          = include_database_name
```

Remove the pound sign to uncomment this line and replace `include_database_name` with the name of the database you want to replicate. This example shows the `binlog_do_db` directive pointing to a database named `db`, but if you have an existing database on your source that you want to replicate, use its name in place of `db`:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
binlog_do_db          =  db
```

Note: If you want to replicate more than one database, you can add another `binlog_do_db` directive for every database you want to add. This tutorial will continue on with replicating only a single database, but if you wanted to replicate more it might look like this:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
binlog_do_db          =  db
binlog_do_db          =  db_1
binlog_do_db          =  db_2
```

Alternatively, you can specify which databases MySQL should not replicate by adding a `binlog_ignore_db` directive for each one:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
binlog_ignore_db          =  db_to_ignore
```

After making these changes, save and close the file. If you used `nano` to edit the file, do so by pressing `CTRL + X`, `Y`, and then `ENTER`.

Then restart the MySQL service by running the following command:

```
source:~$ sudo systemctl restart mysql
```

Copy

With this MySQL instance is ready to function as the source database which your other MySQL server will replicate. Before you can configure your replica, though, there

are still a few more steps you need to perform on the source to ensure that your replication topology will function correctly. The first of these is to create a dedicated MySQL user which will perform any actions related to the replication process.

Step 3 – Creating a Replication User

Each replica in a MySQL replication environment connects to the source database with a username and password. Replicas can connect using any MySQL user profile that exists on the source database and has the appropriate privileges, but this tutorial will outline how to create a dedicated user for this purpose.

Start by opening up the MySQL shell:

```
source:~$ sudo mysql
```
Copy

Note: If you configured a dedicated MySQL user that authenticates using a password, you can connect to your MySQL with a command like this instead:

```
source:~$ mysql -u sammy -p
```
Copy

Replace `sammy` with the name of your dedicated user, and enter this user's password when prompted.

Be aware that some operations throughout this guide, including a few that must be performed on the replica server, require advanced privileges. Because of this, it may be more convenient to connect as an administrative user, as you can with the previous `sudo mysql` command. If you want to use a less privileged MySQL user throughout this guide, though, they should at least be granted the `CREATE USER`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, and `REPLICATION_SLAVE_ADMIN` privileges.

From the prompt, create a new MySQL user. The following example will create a user named **replica_user**, but you can name yours whatever you'd like. Be sure to change `replica_server_ip` to your **replica server's public IP address** and to change `password` to a strong password of your choosing:

```
mysql> CREATE USER ' replica_user '@' replica_server_ip ' IDENTIFIED W
```
Copy /sc

Note: This command specifies that **replica_user** will use the `mysql_native_password` authentication plugin. It's possible to instead use MySQL's default authentication mechanism, `caching_sha2_password`, but this would require setting up an encrypted

connection between the source and the replica. This kind of setup would be optimal for production environments, but configuring encrypted connections is beyond the scope of this tutorial. The MySQL documentation includes instructions on how to configure a replication environment that uses encrypted connections if you'd like to set this up.

After creating the new user, grant them the appropriate privileges. At minimum, a MySQL replication user must have the `REPLICATION SLAVE` permissions:

```
mysql> GRANT REPLICATION SLAVE ON *.* TO ' replica_user '@' replica_s Copy ip
```

Following this, it's good practice to run the `FLUSH PRIVILEGES` command. This will free up any memory that the server cached as a result of the preceding `CREATE USER` and `GRANT` statements:

```
mysql> FLUSH PRIVILEGES;
```
Copy

With that, you've finished setting up a replication user on your source MySQL instance. However, **do not exit the MySQL shell**. Keep it open for now, as you'll use it in the next step to obtain some important information about the source database's binary log file.

Step 4 – Retrieving Binary Log Coordinates from the Source

Recall from the Understanding Replication in MySQL section that MySQL implements replication by copying database events from the source's binary log file line by line and implementing each event on the replica. When using MySQL's binary log file position-based replication, you must provide the replica with a set of coordinates that detail the name of the source's binary log file and a specific position within that file. The replica then uses these coordinates to determine the point in the log file from which it should begin copying database events and track which events it has already processed.

This step outlines how to obtain the source instance's current binary log coordinates in order to set your replicas to begin replicating data from the latest point in the log file. To make sure that no users change any data while you retrieve the coordinates, which could lead to problems, you'll need to lock the database to prevent any clients from reading or writing data as you obtain the coordinates. You will unlock everything shortly, but this procedure will cause your database to go through some amount of downtime.

You still have your source server's MySQL shell open from the end of the previous step. From the prompt, run the following command which will close all the open tables in every database on your source instance and lock them:

```
mysql> FLUSH TABLES WITH READ LOCK;
```
Copy

Then run the following operation which will return the current status information for the source's binary log files:

```
mysql> SHOW MASTER STATUS;
```
Copy

You will see a table similar to this example in your output:

```
Output
+------------------+----------+--------------+------------------+------------
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gt
+------------------+----------+--------------+------------------+------------
| mysql-bin.000001 |      899 |  db          |                  |
+------------------+----------+--------------+------------------+------------
1 row in set (0.00 sec)
```

This is the position from which the replica will start copying database events. Record the `File` name and the `Position` value, as you will need these later when you initiate replication.

What you do immediately after obtaining this information depends on whether your source database has any existing data you want to migrate over to your replicas. Jump to whichever of the two following subsections makes the most sense for your situation.

If Your Source Doesn't Have Any Existing Data to Migrate

If your source MySQL instance is a new installation or doesn't have any existing data you want to migrate to your replicas, you can at this point unlock the tables:

```
mysql> UNLOCK TABLES;
```
Copy

If you haven't done so already, you could create the database you've chosen to replicate while you still have the MySQL shell open. In keeping with the example given in Step 2, the following operation will create a database named `db`:

```
mysql> CREATE DATABASE  db ;
```
Copy

```
Output
```

```
Query OK, 1 row affected (0.01 sec)
```

After that, close the MySQL shell:

```
mysql> exit
```

Copy

Following that, you can move on to the next step.

If Your Source Has Existing Data to Migrate

If you have data on your source MySQL instance that you want to migrate to your replicas, you can do so by creating a snapshot of the database with the `mysqldump` utility. However, your database should still be currently locked. If you make any new changes in the same window, the database will automatically unlock. Likewise, the tables will automatically unlock if you exit the client.

Unlocking the tables could lead to problems since it would mean that clients could again change the data in the database. This could potentially lead to a mismatch between your data snapshot and the binary log coordinates you just retrieved.

For this reason, **you must open a new terminal window or tab on your local machine** so you can create the database snapshot without unlocking MySQL.

From the new terminal window or tab, open up another SSH session to the server hosting your **source MySQL instance**:

```
$ ssh  sammy @ source_server_ip
```

Copy

Then, from the new tab or window, export your database using `mysqldump`. The following example creates a dump file named `db.sql` from a database named `db`, but make sure you include the name of your own database instead. Also, be sure to run this command in the bash shell, not the MySQL shell:

```
source:~$ sudo mysqldump -u root  db  >  db .sql
```

Copy

Following that you can close this terminal window or tab and return to your first one, which should still have the MySQL shell open. From the MySQL prompt, unlock the databases to make them writable again:

```
mysq UNLOCK TABLES;
```

Copy

Then you can exit the MySQL shell:

```
mysql> exit
```

Copy

You can now send your snapshot file to your replica server. Assuming you've configured SSH keys on your source server and have added the source's public key to your replica's `authorized_keys` file, you can do this securely with an `scp` command like this:

```
source:~$ scp db.sql  sammy @ replica_server_ip :/tmp/
```

Copy

Be sure to replace `sammy` with the name of the administrative Ubuntu user profile you created on your replica server, and to replace `replica_server_ip` with the replica server's IP address. Also, note that this command places the snapshot in the replica server's `/tmp/` directory.

After sending the snapshot to the replica server, SSH into it:

```
$ ssh  sammy @ replica_server_ip
```

Copy

Then open up the MySQL shell:

```
replica:~$ sudo mysql
```

Copy

From the prompt, create the new database that you will be replicating from the source:

```
mysql> CREATE DATABASE  db ;
```

Copy

You don't need to create any tables or load this database with any sample data. That will all be taken care of when you import the database using the snapshot you just created. Instead, exit the MySQL shell:

```
mysql> exit
```

Copy

Then import the database snapshot:

```
re:~$ sudo mysql  db  < /tmp/ db.sql
```

Copy

Your replica now has all the existing data from the source database. You can complete the final step of this guide to configure your replica server to begin replicating new changes made on the source database.

Step 5 – Configuring the Replica Database

All that's left to do is to change the replica's configuration similar to how you changed the source's. Open up the MySQL configuration file, `mysqld.cnf`, this time **on your replica server**:

```
replica:~$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

Copy

As mentioned previously, each MySQL instance in a replication setup must have a unique `server-id` value. Find the replica's `server-id` directive, uncomment it, and change its value to any positive integer, as long as it's different from that of the source:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
server-id               =  2
```

Following that, update the `log_bin` and `binlog_do_db` values so that they align with the values you set in the source machine's configuration file:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
log_bin                 = /var/log/mysql/mysql-bin.log
. . .
binlog_do_db            =  db
. . .
```

Lastly, add a `relay-log` directive defining the location of the replica's relay log file. Include the following line at the end of the configuration file:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
relay-log               = /var/log/mysql/mysql-relay-bin.log
```

After making these changes, save and close the file. Then restart MySQL on the replica to implement the new configuration:

```
replica:~$ sudo systemctl restart mysql
```
Copy

After restarting the `mysql` service, you're finally ready to start replicating data from your source database.

Step 6 – Starting and Testing Replication

At this point, both of your MySQL instances are fully configured to allow replication. To start replicating data from your source, open up the the MySQL shell **on your replica server**:

```
replica:~$ sudo mysql
```
Copy

From the prompt, run the following operation, which configures several MySQL replication settings at the same time. After running this command, once you enable replication on this instance it will try to connect to the IP address following `SOURCE_HOST` using the username and password following `SOURCE_USER` and `SOURCE_PASSWORD`, respectively. It will also look for a binary log file with the name following `SOURCE_LOG_FILE` and begin reading it from the position after `SOURCE_LOG_POS`.

Be sure to replace `source_server_ip` with your source server's IP address. Likewise, `replica_user` and `password` should align with the replication user you created in Step 2; and `mysql-bin.000001` and `899` should reflect the binary log coordinates you obtained in Step 3.

You may want to type this command out in a text editor before running it on your replica server so that you can more easily replace all the relevant information:

```
mysql> CHANGE REPLICATION SOURCE TO
mysql> SOURCE_HOST=' source_server_ip ',
mysql> SOURCE_USER=' replica_user ',
mysql> SOURCE_PASSWORD=' password ',
mysql> SOURCE_LOG_FILE=' mysql-bin.000001 ',
mysql> SOURCE_LOG_POS= 899 ;
```
Copy

Following that, activate the replica server:

```
mysql> START REPLICA;
```
Copy

If you entered all the details correctly, this instance will begin replicating any changes made to the `db` database on the source.

You can see details about the replica's current state by running the following operation. The `\G` modifier in this command rearranges the text to make it more readable:

```
mysql> SHOW REPLICA STATUS\G;
```
Copy

This command returns a lot of information which can be helpful when troubleshooting:

```
Output
*************************** 1. row ***************************
             Replica_IO_State: Waiting for master to send event
                  Source_Host: 138.197.3.190
                  Source_User: replica_user
                  Source_Port: 3306
                Connect_Retry: 60
              Source_Log_File: mysql-bin.000001
          Read_Source_Log_Pos: 1273
               Relay_Log_File: mysql-relay-bin.000003
                Relay_Log_Pos: 729
        Relay_Source_Log_File: mysql-bin.000001
. . .
```

Note: If your replica has an issue in connecting or replication stops unexpectedly, it may be that an event in the source's binary log file is preventing replication. In such cases, you could run the `SET GLOBAL SQL_SLAVE_SKIP_COUNTER` command to skip a certain number of events following the binary log file position you defined in the previous command. This example only skips the first event:

```
mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
```
Copy

Following that, you'd need to start the replica again:

```
mysql> START REPLICA;
```
Copy

Also, if you ever need to stop replication, note that you can do so by running the following operation on the replica instance:

```
mysql> STOP REPLICA;
```
Copy

Your replica is now replicating data from the source. Any changes you make to the source database will be reflected on the replica MySQL instance. You can test this by creating a sample table on your source database and checking whether it gets replicated successfully.

Begin by opening up the MySQL shell on your source machine:

```
source:~$ sudo mysql
```
Copy

Select the database you chose to replicate:

```
mysql> USE  db ;
```
Copy

Then create a table within that database. The following SQL operation creates a table named `example_table` with one column named `example_column` :

```
mysql> CREATE TABLE example_table (
mysql> example_column varchar(30)
mysql> );
```
Copy

```
Output
Query OK, 0 rows affected (0.03 sec)
```

If you'd like, you can also add some sample data to this table:

```
mysql> INSERT INTO example_table VALUES
mysql> ('This is the first row'),
mysql> ('This is the second row'),
mysql> ('This is the third row');
```
Copy

```
Output
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

After creating a table and optionally adding some sample data to it, go back to your replica server's MySQL shell and select the replicated database:

```
mysql> USE  db ;
```
Copy

Then run the `SHOW TABLES` statement to list all the tables within the selected database:

```
mysql> SHOW TABLES;
```
Copy

If replication is working correctly, you'll see the table you just added to the source listed in this command's output:

```
Output
+---------------+
| Tables_in_db  |
+---------------+
|  example_table  |
+---------------+
1 row in set (0.00 sec)
```

Also, if you added some sample data to the table on the source, you can check whether that data was also replicated with a query like the following:

```
mysql> SELECT * FROM  example_table ;
```
Copy

In SQL, an asterisk (`*`) is shorthand "all columns." So this query essentially tells MySQL to return every column from `example_table`. If replication is working as expected, this operation will return that data in its output:

```
Output
+-----------------------+
| example_column        |
+-----------------------+
| This is the first row  |
| This is the second row |
| This is the third row  |
+-----------------------+
3 rows in set (0.00 sec)
```

If either of these operations fail to return the example table or data that you added to the source, it may be that you have an error somewhere in your replication configuration. In such cases, you could run the `SHOW REPLICA STATUS\G` operation to try finding the

cause of the issue. Additionally, you can consult MySQL's documentation on troubleshooting replication for suggestions on how to resolve replication problems.

Conclusion

By completing this tutorial, you will have set up a MySQL replication environment that uses MySQL's binary log file position-based replication method with one source and one replica. Bear in mind, though, that the procedure outlined in this guide represents only one way of configuring replication in MySQL. MySQL provides a number of different replication options which you can use to produce a replication environment optimized for your needs. There are also a number of third-party tools, such as Galera Cluster, that you can use to expand upon MySQL's built-in replication features.

If you have any further questions about the specific capabilities of replication in MySQL, we encourage you to check out MySQL's official documentation on the subject. If you'd like to learn more about MySQL generally, you could also check out our entire library of MySQL-related content.

Thanks for learning with the DigitalOcean Community. Check out our offerings for compute, storage, networking, and managed databases.

Learn more about us →

About the authors

Mark Drake Author

Manager, Developer Education

Technical Writer @ DigitalOcean

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

10 Comments

Leave a comment...

This textbox defaults to using **Markdown** to format your answer.

You can type **!ref** in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

jason • March 6, 2013

A few suggestions I would like to make in regards to this article:

1. I think it should be mentioned that MySQL has the capabilities to bind to multiple IP addresses and that to take advantage of this feature, an additional “bind-address” line is required.
2. In the following section (regarding server-ids) you mentioned that this line has to be “uncommented” but I’m not sure all users will know that means that they have to remove the “#” preceding it if there is one (which there will be in the installed version of my.cnf.
3. In the section regarding the databases that they want to replicate, I believe it should be mentioned that leaving this line commented will result in all databases being replicated. In some, though admittedly rather rare situations, this is the desired behavior. This, too, will allow new databases

to be replicated automatically without any additionally editing of the configuration files.

4. Some of the wording in the explanations is a bit awkward, specifically: “This is the position from which the slave database will start replicating. Record these numbers, they will come in useful later.” This might be more grammatically correct if stated as “This is the file name and position that the master server is using for logging purposes. Make a note of these, as you will need to enter them later”.
5. After the mysqldump step, there should be an additional step explaining how to send the newly created .sql file to the remote host. SCP would be the easiest way, in my opinion, to accomplish this.

Show replies Reply

rutgergeelen • June 18, 2013

As a backup I would not recommended this approach. If data is corrupted, lost or deleted in the master that will be synced to the slave. Restoring the slave to the master does not get you your data back.

What I miss is the explanation how to use the slave for failover. If somebody could add that that would be helpful (at least for me)

Ruter

Show replies Reply

Kamal Nasser • August 22, 2013

@Sahaya: I believe it’s instant by default. It is configurable: http://alexalexander.blogspot.com/2013/03/mysql-slave-delay-how-to.html

Reply

munees • August 22, 2013

How often slave updated from master? Is it configurable?

Reply

radiva1985 • July 5, 2019

Be careful of mySQL duplicate UUIDs had to change this for mine to work

Show replies Reply

dbconsultoria • March 5, 2015

Great post!

My two cents on it: after you follow all the steps, your replication may not work at first, so check your slave status (“show slave status”), and read the error messages.

If you followed the steps closely, and performed the process in Linux Ubuntu and MySQL 5.6, you might get this message: “The slave I/O thread stops because master and slave have equal MySQL server UUIDs; these UUIDs must be different for replication to work.”

You need to remove the “auto.cnf” from the datadir in the slave machine. The datadir might differ from every linux distro. To find that out in mysql, run: “SELECT @@DATADIR”.

All the best!

Reply

ken555452 • March 21, 2014

If I’m going to have multiple slaves and web servers query the master, would it make more sense to do this:

```
bind-address            = 0.0.0.0
```

And then in iptables:

```
iptables -A INPUT -p tcp -s 12.34.45.67 –dport 3306 -j ACCEPT
iptables -A INPUT -p tcp -s 23.45.67.78 –dport 3306 -j ACCEPT
iptables -A INPUT -p tcp -s 34.56.78.89 –dport 3306 -j ACCEPT
iptables -A INPUT -p tcp –dport 3306 -j DROP
```

Adding a line for each IP I want to allow to connect to the master?

Reply

kevinthulin • February 5, 2014

Great tutorial, I am going to add a 1GB slave server where I will do my backups.

Will the slave server slow down my prod/master server?

Reply

Kamal Nasser • December 5, 2013

@matan: The write will fail as it is not supposed to accept writes. You might want to check out master-master replication: https://www.digitalocean.com/community/articles/how-to-set-up-mysql-master-master-replication

Reply

matan • December 4, 2013

Thanks for the great article. What happens when a slave db is modified? Does it update the master which then in-turn updates any other slave dbs?

Reply

Load More Comments

This work is licensed under a Creative Commons Attribution-NonCommercial- ShareAlike 4.0 International License.

Try DigitalOcean for free

Click below to sign up and get **$200 of credit** to try our products over 60 days!

Sign up

Popular Topics

Ubuntu

Linux Basics

JavaScript

Python

MySQL

Docker

Kubernetes

All tutorials →

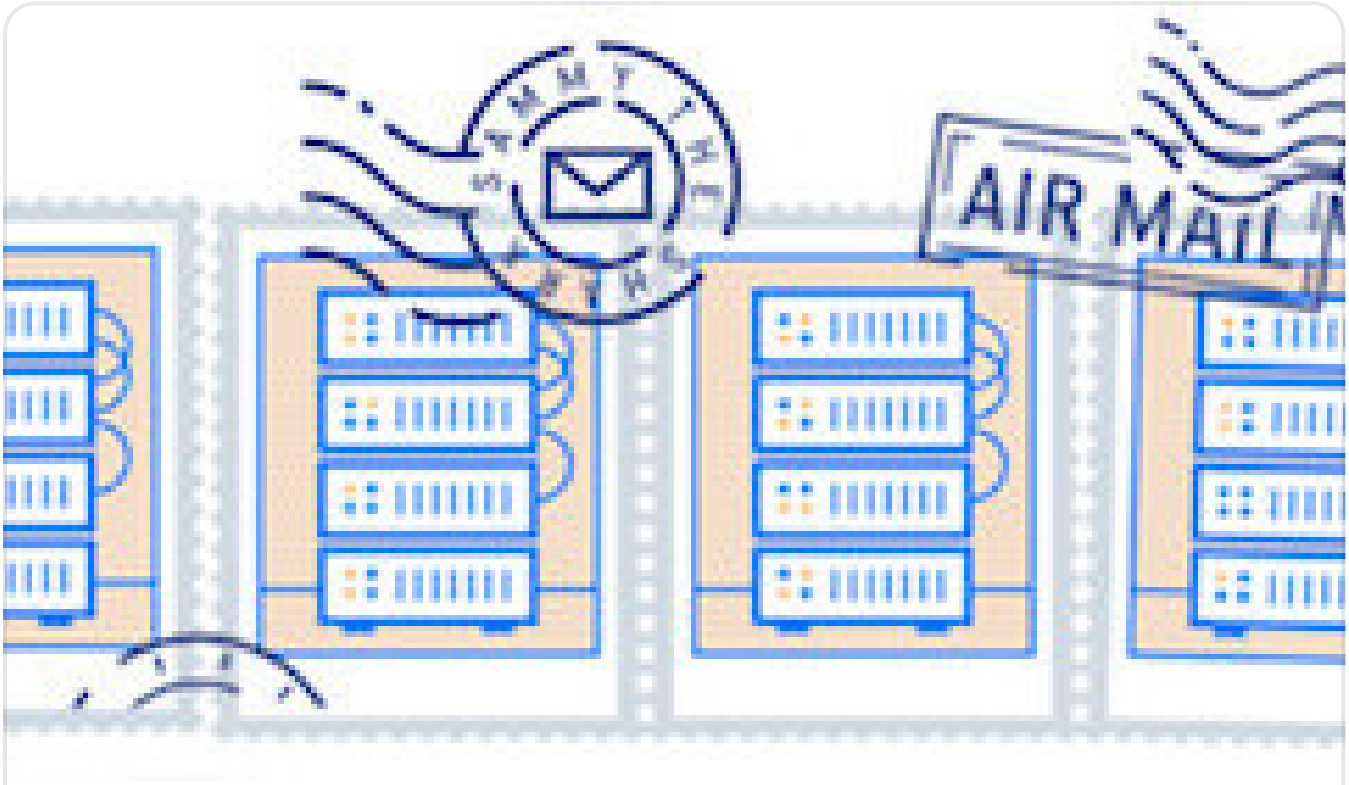
Talk to an expert →

Congratulations on unlocking the whale ambience easter egg! Click the whale button at the bottom left of your screen to toggle some ambient whale noises while you re

Thank you to the Glacier Bay National Park & Preserve and Merrick079 for the sounds behind this easter egg.

Interested in whales, protecting them, and their connection to helping prevent climate change? We recommend checking out the Whale and Dolphin Conservation.

Reset easter egg to be discovered again / Permanently dismiss and hide easter egg



Get our biweekly newsletter

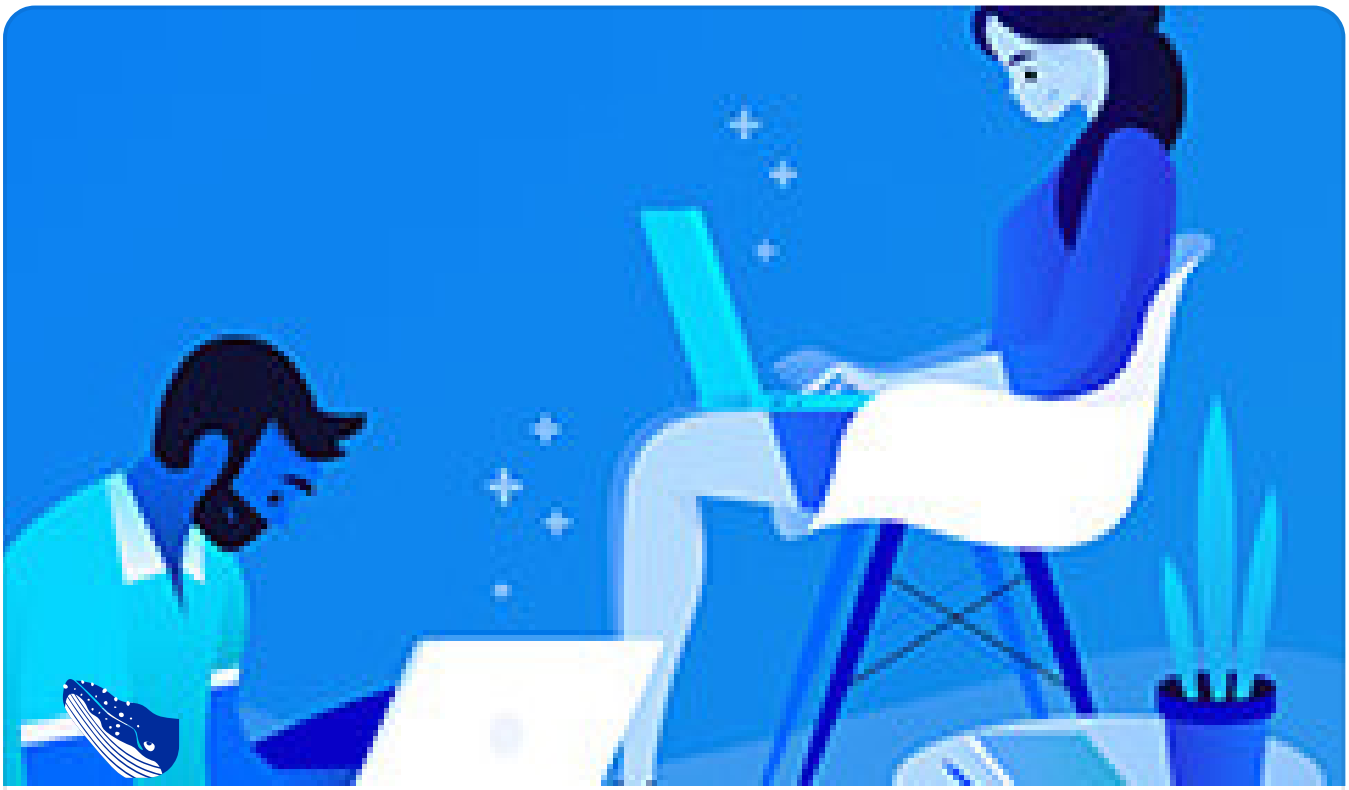
Sign up for Infrastructure as a Newsletter.

Sign up →

Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →



Become a contributor

Get paid to write technical tutorials and select a tech-focused charity to receive a matching donation.

Learn more →

Featured on Community

Kubernetes Course Learn Python 3 Machine Learning in Python

Getting started with Go Intro to Kubernetes

DigitalOcean Products

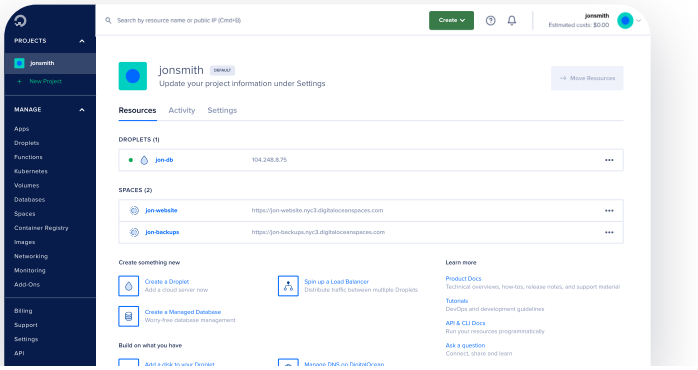
Cloudways Virtual Machines Managed Databases Managed Kubernetes

Block Storage Object Storage Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow — whether you're running one virtual machine or ten thousand.

Learn more



Get started for free

Sign up and get $200 in credit for your first 60 days with DigitalOcean.

Get started

This promotional offer applies to new accounts only.