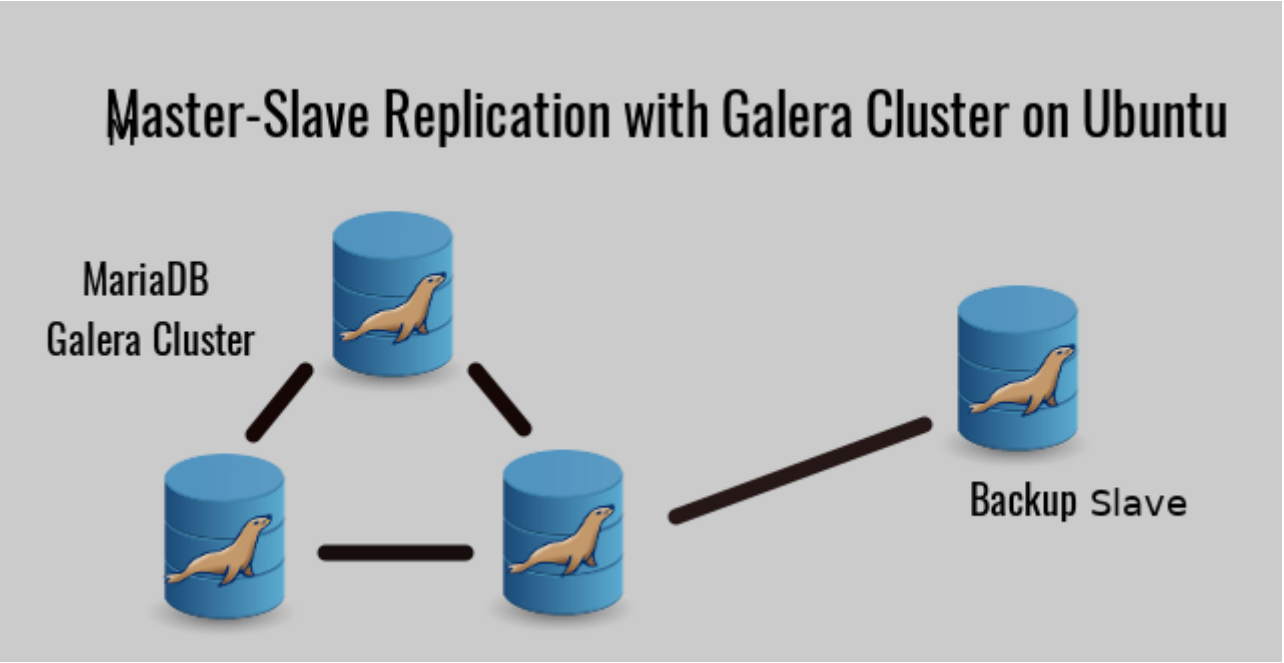
Set Up MariaDB Master-Slave Replication with Galera Cluster on Ubuntu

Previously, we discussed how to set up MariaDB Galera cluster on Ubuntu and how to encrypt replication traffic in Galera cluster. In this tutorial, we will be learning how to add an asynchronous replication slave to Galera cluster, which means the Galera cluster will be acting as a master.

Galera Cluster As a Master

Remember that replication is not a replacement for backup. Although there are multiple nodes in Galera cluster holding a copy of your database, if a DROP DATABASE command is accidentally run on one of the nodes in the cluster, all other nodes will drop the database. We are going to set up master-slave replication and the Galera cluster will be acting a master. You will need another server acting as slave. Once the setup is done, you can take backups on the slave using *mysqldump* and the workload in Galera cluster won't be interrupted when a backup is created.

Hint: Galera cluster uses synchronous replication between the nodes, whereas the traditional MariaDB master-slave replication uses asynchronous replication between master and slave.



Setting up master-slave replication with Galera cluster can be done in 5 steps:

1. Configure binary log and replication on Galera master
2. Enable relay log and replication on the slave
3. Dump databases on the Galera master and import them into the slave
4. (optional) Enable TLS encryption
5. Connect the slave to Galera master

The overall steps are similar to setting up a traditional master-slave replication, but you need to adjust some settings for the Galera master.

Prerequisites

You should have already set up a Galera cluster with at least 3 nodes and you need to prepare another server as slave.

Ideally, the Galera master and slave should use the same MariaDB version. Replication between different MariaDB versions may cause problems. If you run MariaDB database server on Ubuntu, then you should either install MariaDB from Ubuntu repository on all the servers, or install the latest MariaDB version from MariaDB.org repository on all the servers.

When following the steps below, you should apply master configuraions on all nodes in Galera cluster, so if one node goes down, you can quickly configure the slave to replicate from another node.

Step 1: Configure Binary Log and Replication on Galera Master

Master-slave replication is based on the *binary log*. You must enable binary logging on the master server in order for replication to work. The Galera nodes have already enabled binary logging in row format. (Galera only supports row-based binary log format.) We need to edit some other parameters, so open the MariaDB main configuration file.

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

or

```
sudo nano /etc/mysql/my.cnf
```

Add the following lines in the `[mysqld]` unit.

```
# Galera node as master
wsrep_gtid_mode      = on
wsrep_gtid_domain_id = 0
server-id            = 01
log_slave_updates    = on
log-bin              = /var/log/mysql/master-bin
log-bin-index        = /var/log/mysql/master-bin.index
gtid_domain_id       = 1
```

In the above configuration, we enabled GTID mode for write-set replication. `wsrep_gtid_domain_id` and `server-id` need to set to the same value on all nodes in the cluster.

By default, write-set replication in Galera are not written to binary log. In order for a node in Galera cluster to replicate write-sets to an asynchronous slave, `log-slave-updates` must be enabled on the Galera master. If this isn't enabled, then changes replicated from another node in the cluster won't be replicated to the asynchronous slave. It's recommended that you enable `log-slave-updates` on all nodes in the cluster. If one node in the cluster goes offline, you can configure the slave to replicate from another node in the cluster.

The binary log files needs to be set to the same path on all nodes in the cluster. `gitd_domain_id` should be set to a different value on all nodes in Galera cluster and the value should be different from the value of `wsrep_gtid_domain_id`.

Save and close the file. After you apply this configuration on all nodes in Galera cluster, you need to restart the whole cluster for the changes to take effect. To restart the cluster, you need to shut down all MariaDB server one at a time.

```
sudo systemctl stop mariadb
```

Then run the following command on the last node that leaves the cluster to bootstrap the cluster.

```
sudo galera_new_cluster
```

Next, you need to start MariaDB server on other nodes one at a time.

```
sudo systemctl start mariadb
```

When the master node sits on the public Internet, it's recommended to restrict access to port 3306 (default MariaDB port). For example, you can use UFW to create an IP address whitelist, allowing only the slave's IP addresses to connect to port 3306.

```
sudo ufw insert 1 allow in from IP_address_of_slave to any port 3306
```

For how to use UFW, please see the following article:

- Getting started with UFW firwall on Debian, Ubuntu, Linux Mint server

After that, we need to add an replication user on the master server. The slave server will use this user to remotely log into master server and request binary logs. Log into MariaDB monitor.

```
sudo mysql -u root
```

Then create a user and grant `replication slave` privilege to this user. Replace the red text with your preferred username and password.

```
create user 'replicant'@'%' identified by 'replicant_password';

grant replication slave on *.* to replicant;
```

If the slave is going to connect to the master over the public Internet, it's a good practice to enforce TLS encryption.

```
grant replication slave on *.* to replicant require ssl;
```

Then flush the privilege table.

```
flush privileges;
```

This database user will be replicated to other nodes in the cluster.

Step 2: Enable Relay Log and Replication on the Slave

Open the main MariaDB configuration file on the slave.

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

or

```
sudo nano /etc/mysql/my.cnf
```

Find the `Logging and Replication` section in `[mysqld]` unit and Add the following lines.

```
#Replication slave to Galera Cluster
server-id       = 02
relay-log-index = /var/log/mysql/slave-relay-bin.index
relay-log       = /var/log/mysql/slave-relay-bin
gtid_domain_id  = 99

log-bin         = /var/log/mysql/slave-bin
log-bin-index   = /var/log/mysql/slave-bin.index
binlog_format   = mixed
```

In the above configuration, we need to set the `server_id` to a value different than the one on Galera master. The relay log needs to be enabled for master-slave replication. The value of `gtid_domain_id` on slave should be different than the value of `wsrep_gtid_domain_id` and `gtid_domain_id` on Galera master.

Master-slave replication does not require binary logging on the slave, but if you are going to take backups on the slave using *mysqldump*, then you need to enable binary logging. The format of the binary log can be different than the format used on Galera master.

You can also use replication filters like below to replicate specific database. (Note that replication filter should be used on the slave, not on the Gelara master.)

```

```

```
replicate-do-db=db1_name
replicate-do-db=db2_name
```

If the slave is going to act a master of another slave, add the following line as well.

```
log_slave_updates = ON
```

Save and close the file. Then restart the slave MariaDB server for the changes to take effect.

```
sudo systemctl restart mariadb
```

Sometimes MariaDB may fail to restart. Run `systemctl status mariadb` to check the status.

Step 3: Copy Database From the Galera Master to the Slave

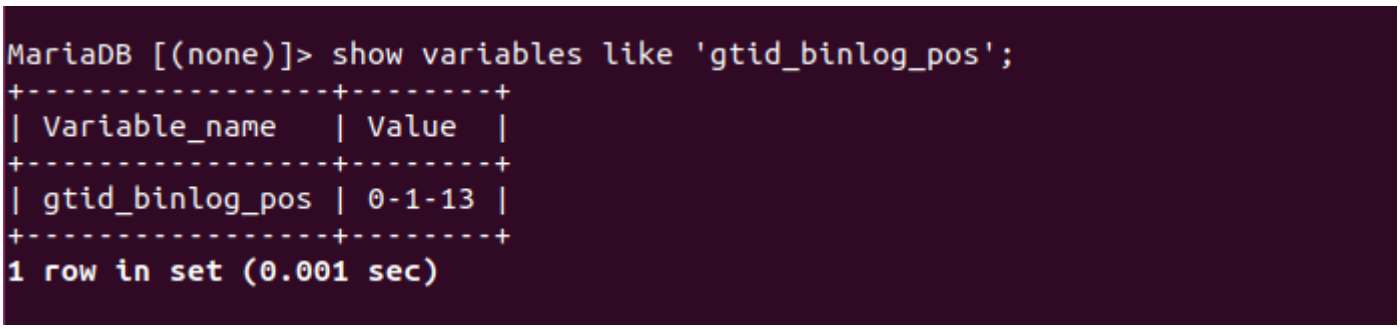
Choose one Galera node as the master and log into the master MariaDB monitor. Run the following command to prevent any further changes to databases.

```
flush tables with read lock;
```

This master node will become desynced from the rest of the cluster, which can continue running as normal. It will sync with other nodes again when we unlock the tables later. Also note that if this master node is serving a website, the above command can cause the website go offline.

Then obtain the binary log GTID (global transaction ID) position with the following command.

```
show variables like 'gtid_binlog_pos';
```



Do NOT exit MariaDB monitor yet, because exiting it now will release the lock. Now open another terminal window and SSH into your master server. Use `mysqldump` utility to dump the database toa `.sql` file.

```
sudo mysqldump -u root database_name > database_name.sql
```

Your can obtain the database name by running the following command at the MariaDB monitor.

```
show databases;
```

You can also dump all databases with the following command:

```
sudo mysqldump -u root --all-databases > all-databases.sql
```

After the database is dumped to disk, you can unlock tables on master server by running the following command at the MariaDB monitor.

```
unlock tables;
```

Use the `scp` command or whatever method you prefer to copy this SQL file to your slave server.

To import the single database, log into the slave MariaDB monitor.

```
sudo mysql -u root
```

Create a blank database with the same name.

```
create database database_name;
```

Exit MariaDB monitor.

```
exit;
```

Import the database into the slave MariaDB server with the following command.

```
sudo mysql -u root database_name < database_name.sql
```

If you are going to enable replication on all databases, then you can import all databases to slave MariaDB server with the following command:

```
sudo mysql -u root < all-databases.sql
```

To keep data consistent with the master, it is advisable to enable read-only mode on the slave.

Replication will work as usual in read-only mode. Open the `50-server.cnf` file on the slave.

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

Add the following line in `[mysqld]` unit to enable read-only mode.

```
read-only = 1
```

Users with SUPER privilege (like root) can still write to the database, so you should be careful when granting privileges to users. If you don't want anyone to be able to change/delete the database, you can add the following line in `[mysqld]` unit.

```
innodb-read-only = 1
```

Save and close the file. Then restart MariaDB for the change to take effect.

```
sudo systemctl restart mariadb
```

Step 4: Enabling TLS Encryption on Gelera Master

Note: If the slave and master are in a private network, your don't have to do this step. Skip to step 5.

If the slave is going to connect to the master over the public Internet, it is necessary to enable TLS encryption to prevent traffic snooping. Your server may have a web server with TLS enabled, so you can use that TLS certificate for MariaDB as well. For example, I have enabled TLS in my Nginx web server with Let's Encrypt. To enable TLS in MariaDB, open the `50-server.cnf` file.

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

Find the `Security Features` section in `[mysqld]` unit. Add the following lines:

```
ssl-ca = /etc/letsencrypt/live/www.linuxbabe.com/chain.pem
ssl-cert = /etc/letsencrypt/live/www.linuxbabe.com/cert.pem
ssl-key = /etc/letsencrypt/live/www.linuxbabe.com/privkey.pem
```

Save and close the file. The `mysql` user needs permission to access the above SSL files, so you need to grant read permission with the following commands.

```
sudo setfacl -R -m "u:mysql:rx" /etc/letsencrypt/archive/

sudo setfacl -R -m "u:mysql:rx" /etc/letsencrypt/live/
```

Then restart the master node for the changes to take effect.

```
sudo systemctl restart mariadb
```

Your may be wondering why you don't need to grant the `www-data` user read permission of the SSL files. That's because Apache or Nginx has a master process running as root user. However, all MariaDB processes are running as mysql user.

After MariaDB restarts, log into MariaDB monitor and run the following command to check if SSL is successfully enabled.

```
show global variables like "have_ssl";
```

If the value is "Yes", then SSL is enabled.

```
MariaDB [(none)]> show global variables like "have_ssl";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
1 row in set (0.002 sec)
```

If the value is "DISABLED", that means there are something wrong in your SSL configuration. Check the MariaDB error log (`/var/log/mysql/error.log`) to find the reason.

The MariaDB server binary from Debian/Ubuntu repository is statically linked with MariaDB's bundled YaSSL library. The MariaDB binary from the MariaDB.org repository is dynamically linked with the system's TLS library, usually OpenSSL. You can log into MariaDB monitor and run the following command to check which SSL library your MariaDB server is using.

```
show variables like "version_ssl_library";
```

```
MariaDB [(none)]> show variables like "version_ssl_library";
+---------------------+----------------------------+
| Variable_name       | Value                      |
+---------------------+----------------------------+
| version_ssl_library | OpenSSL 1.1.0g  2 Nov 2017 |
+---------------------+----------------------------+
1 row in set (0.001 sec)
```

If you would like to use TLS 1.3 in MariaDB, then you need to install MariaDB from MariaDB.org repository and install OpenSSL 1.1.1 on your Ubuntu system.

You can test TLS connection by logging in from the slave with the following command. `--ssl` option enforces secure connection.

```
mysql -h Master_IP_Address -u replicant -p --ssl
```

Once you are logged in, run

```
status;
```

In the output, you can see the SSL cipher in use.

```
MariaDB [(none)]> status;
--------------
mysql  Ver 15.1 Distrib 10.3.13-MariaDB, for debian-linux-gnu (x86_64) using readline 5.2

Connection id:          356
Current database:
Current user:           replicant@
SSL:                    Cipher in use is ECDHE-RSA-AES256-GCM-SHA384
Current pager:          stdout
Using outfile:          ''
Using delimiter:        ;
Server:                 MariaDB
```

Step 5: Connect the Slave to the Master

We are going to use GTID-based replication. Log into the slave MariaDB monitor and run the following command to set the `gtid_slave_pos` variable. Its value should be the same as the `gtid_binlog_pos` variable on the master.

```
MariaDB [(none)]> set global gtid_slave_pos = "0-1-13";
```

Then run the following command to create a connection profile.

```
MariaDB [(none)]> change master 'master01' to
    -> master_host='master_IP_address',
    -> master_user='replicant',
    -> master_password='replicant_password',
    -> master_port=3306,
    -> master_connect_retry=10,
    -> master_use_gtid=slave_pos,
    -> master_ssl=1;
```

In the first line, the connection name is set to master01. `master_use_gtid=slave_pos` enables GTID (Global Transaction ID) in MariaDB replication, so you don't need to use the old-style `master_log_file` and `master_log_pos` any more.

The last line enforces TLS encryption. If the master and slave are in a secure private network, then you don't have to enforce TLS encryption, so you can remove the last line. Notice that the last line ends with a semicolon.

Then start this connection.

```
MariaDB [(none)]> start slave 'master01';
```

Check the status.

```
MariaDB [(none)]> show slave 'master01' status\G;
```

If you see no errors in the output, that means replication is running smoothly. You should see the following two "Yes" indicating everything is going well. If one of them is not "Yes", then something is not right.

```
 Slave_IO_Running: Yes
 Slave_SQL_Running: Yes
```

Now if you make a change in your master database server, it will be replicated to the slave database server. The **slave I/O thread** connects to the master and requests binary log. If there are new transactions in the binary log, the slave I/O thread writes them to the relay log on the slave server. Then the **slave SQL thread** reads the relay log and executes new transactions in the database.

To stop replication, run:

```
MariaDB [(none)]> stop slave 'master01';
```

If you want the replication to restart from a clean state, you can reset the replication.

```
MariaDB [(none)]> reset slave 'master01';
```

By default, when MariaDB server restarts, it resumes all stopped replication tasks.

TroubleShooting

The first time you check the slave status, you might see the following error.

```
Last_SQL_Error: Error 'Duplicate entry
```

This usually happens when the slave connects to the master the first time. If an error occurs, the replication will stop. We can skip this duplicate entry error by adding the following two line in [mysqld] unit in the MariaDB server configuration file (50-server.conf).

```
slave-skip-errors=1062
skip-slave-start
```

Restart MariaDB. Then start the slave replication again.

```
MariaDB [(none)]> start slave 'master01';
```

Check the status.

```
MariaDB [(none)]> show slave 'master01' status\G;
```

After a few moments, the `Seconds_Behind_Master` in the status output will get to zero. After that, you can remove the two lines in `[mysqld]` and restart MariaDB.

If there are other errors, you can add the error code like below.

```
slave-skip-errors = 1062, 1032
```

It’s recommended that you only use the `slave-skip-errors` option when you first start the replication. Use this option later could cause data inconsistency between master and slave.

Check if Replication is Working

First, change some data on the master, then run the following command on the master MariaDB monitor.

```
MariaDB [(none)]> show variables like 'gtid_binlog_pos';
```

```
MariaDB [(none)]> show variables like 'gtid_binlog_pos';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| gtid_binlog_pos | 0-1-86321 |
+-----------------+-----------+
1 row in set (0.001 sec)
```

Next, on the slave MariaDB monitor, run the following command.

```
MariaDB [(none)]> show variables like 'gtid_slave_pos';
```

```
MariaDB [(none)]> show variables like 'gtid_slave_pos';
+----------------+-----------+
| Variable_name  | Value     |
+----------------+-----------+
| gtid_slave_pos | 0-1-86321 |
+----------------+-----------+
1 row in set (0.002 sec)
```

If the two values are the same, then data changes on the master is replicated to the slave. You should change some data on each Galera node to see if they are all replicated to the slave.

Changing the Master Node

If the Galera master node goes offline, you can quickly configure the slave to replicate from another Galera node. First, stop the current slave.

```
MariaDB [(none)]> stop slave 'master01';
```

Then you need to change the IP address to another Galera node in the connection profile.

```
```

```
MariaDB [(none)]> change master 'master01' to
    -> master_host='IP_address_of_another_Galera_node',
    -> master_user='replicant',
    -> master_password='replicant_password',
    -> master_port=3306,
    -> master_connect_retry=10,
    -> master_use_gtid=slave_pos,
    -> master_ssl=1;
```

Then start the slave.

```
MariaDB [(none)]> stop slave 'master01';
```

Because we are using global transaction ID and all Galera nodes share the same `gtid_binlog_pos` at a given time (Recall that we set `wsrep_gtid_domain_id` and `server-id` to the same value on all nodes in the cluster), the slave can easily determine where to resume replication on the new master.

Revision #1
Created 11 July 2021 19:12:31 by Admin
Updated 11 July 2021 19:13:01 by Admin