MariaDB

- How to Set Up MariaDB Galera Cluster on Ubuntu 20.04
- How to Encrypt Replication Traffic in MariaDB Galera Cluster on Ubuntu
- Set Up MariaDB Master-Slave Replication with Galera Cluster on Ubuntu

How to Set Up MariaDB Galera Cluster on Ubuntu 20.04

This tutorial will be showing you how to set up MariaDB Galera Cluster on Ubuntu 20.04/18.04 server. MariaDB is an open-source drop-in replacement for MySQL database server.

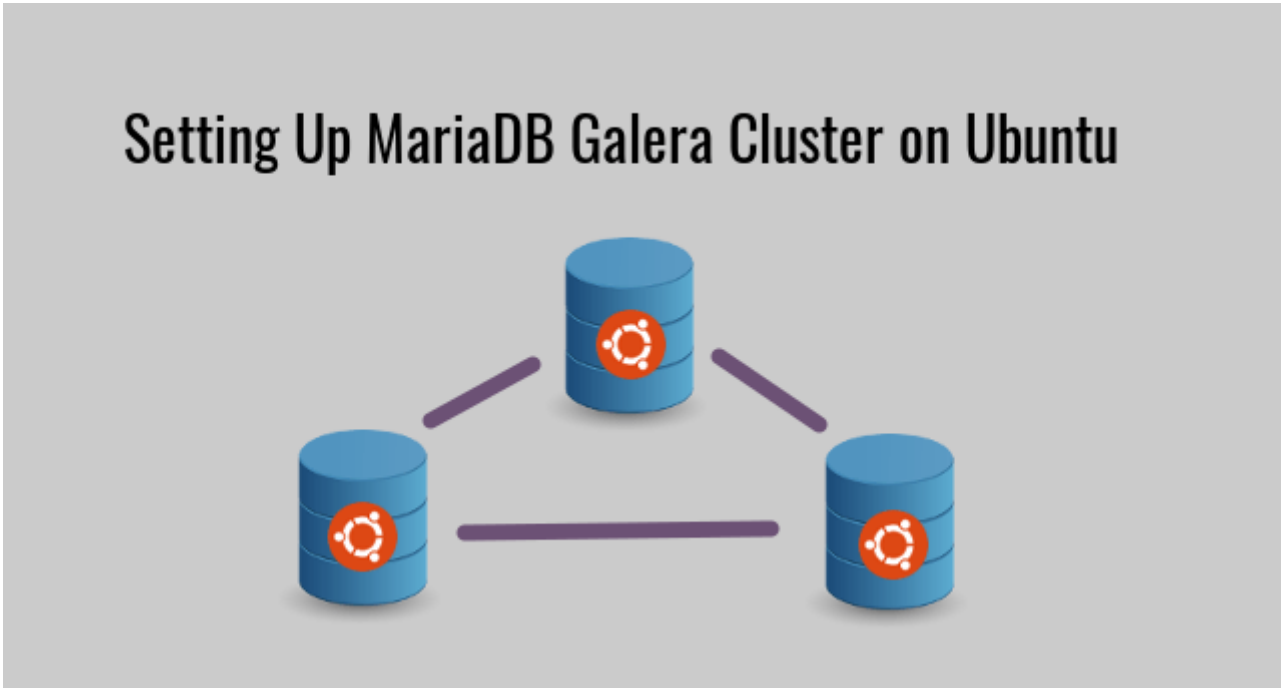
What is Galera Cluster?

Previously, I talked about master-slave replication in MariaDB. It's a setup where data modifications on the master server will be replicated to the slave, but changes on the slave will not be replicated to the master. Galera Cluster is a synchronous multi-master cluster for MySQL, MariaDB, and Percona database servers to implement high-performance and high-availability for data redundancy. A multi-master cluster allows read and writes to any node in the cluster. Data modifications on any node are replicated to all other nodes.

Galera cluster is an open-source data replication and clustering technology developed by Codership , a Finnish company. There are 3 versions of Galera Cluster:

- **Galear cluster for MySQL**: the original Galera developed by Codership
- **MariaDB Galera cluster**: a fork of Codership Galera. MariaDB is a Codership-certified partner.
- **Percona XtraDB cluster**: another fork of Codership Galera

In this tutorial, we will be using the MariaDB Galera cluster.



Features and Benefits of MariaDB Galera Cluster

- High availability. If any individual node in the cluster fails, the other nodes can continue providing service without the need for manual failover procedures.
- High data consistency. Galera cluster uses synchronous replication, so no slave lag or diverged data is allowed between the nodes and no data is lost after a node crash. Transactions are committed in the same order on all nodes.
- Active-active multi-master topology.
- Automatic transaction conflict detection to make data consistent across nodes.
- Read and write to any cluster node. The cluster acts like a standalone MariaDB server.
- Both read and write scalability. No need to split read and write on different nodes.
- Automatic membership control. Failed nodes drop from the cluster.
- Automatic node provisioning. New nodes can join with just a few configuration lines. No need to manually dump the database and import it on new nodes.
- Multiple-threaded slave, **true parallel replication**, on row level.
- Transparent to applications. Direct client connections, native MariaDB look & feel.
- Great support for cloud and WAN environments to build geo-distributed database cluster (across countries and continents).
- Smaller client latencies.
- Easy to set up.

With Galera cluster, you can eliminate a single point of failure and achieve better performance at the same time. Galera cluster performs well both in LAN and WAN environments. You can have

nodes in the cloud, even on small server instances, across multiple data centers and different continents. Together with an open-source file synchronization tool like Syncthing and an anycast CDN + load balancing service like Clouflare, website owners can bring their contents as close to visitors as possible no matter where visitors are located.

Prerequisites of Setting Up Galera Cluster on Ubuntu

How many nodes should you put in the cluster? Well, there is no upper limit, but you should always choose an odd number: 3, 5, 7, and so on to prevent the **split-brain problem**, which I will talk about in a future article. Galera cluster requires at least 3 nodes to be crash-safe. To follow this tutorial, you will need at least 3 MariaDB database servers running on Ubuntu, each server with at least 512MB RAM. Use 1GB RAM or above on each server for smooth operation and better performance. It's recommended to use the same hardware configuration on every node because the cluster will be as slow as the slowest node.

In this tutorial, I'm using 3 Vultr VPS (Virtual Private Server) in 3 different data centers (Silicon Valley, Frankfurt, and Singapore), so my database will still be available in case there is a power outage/network problem in one of the data centers.

Note: 1) Galera cluster only runs on Linux and Unix-like OS. Microsoft Windows is not supported. 2) If your Galera cluster spans continents, there will be latency from 100ms to 300ms. The latency between my 3 servers is around 165ms.

All MariaDB servers must be using InnoDB or XtraDB storage engine, because Galera cluster only supports these two storage engines. Any writes to tables of other engines will not be replicated to other nodes. MyISAM currently isn't supported because it's a non-transactional storage engine.

To check the default storage engines used by your database, log into MariaDB monitor and run the following statement:

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

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

Note that a database may have tables that use different storage engines. To check, run the following statement. Replace “database_name” with your real database name.

```
MariaDB [(none)]> select table_name,engine from information_schema.tables where table_schema
='database_name' and engine ='myISAM';
```

If you found a table using storage engine other than InnoDB, you can change it to InnoDB. For example, to change a table from using MyISAM to InnoDB, run

```
MariaDB [(none)]> use database_name;


MariaDB [(none)]> alter table table_name engine = InnoDB;
```

The first statement selects a particular database and the second statement will change the storage engine of a table to InnoDB. The tables in the default 3 databases (`information_schema`, `mysql` and `performance_schema`) don’t use InnoDB/XtraDB storage engine and there’s no need to change it.

I also recommend reading known limitations of MariaDB Galera cluster before setting up a Galera cluster.

In the following instructions, you need to execute step 1 ~ step 4 on all cluster nodes.

Step 1: Install the Latest Stable Version of MariaDB on Each Node

MariaDB is available from the default Ubuntu repository. However, it’s recommended to install the latest stable version from mariadb.org repository, so you can use the latest and greatest features.

For example, MariaDB 10.4 and 10.5 support Galera 4, which has several cool features such as

- Streaming replication for large transaction (2G+) support. It allows running transactions of unlimited size in a cluster.
- Group commit.
- Improved foreign key support.
- Improved network resiliency to handle poor network connections. Great for clusters that span multiple data centers.
- Rolling cluster upgrades.

Step 2: Configuring Each Node in the Cluster

Prior to MariaDB 10.1, sysadmins need to install the `mariadb-galera-server` package in order to set up a cluster. As of MariaDB 10.1, the Galera cluster feature is bundled into MariaDB. If you have MariaDB 10.4 or above running on Ubuntu 18.04/20.04, you just need to install one more package: `galera-4` – the Galera wsrep (write-set replication) provider library, which is developed by Codership.

```
sudo apt install galera-4
```

Usually, this package is automatically installed when you install MariaDB server on Ubuntu. Now run the following command to edit the MariaDB Galera configuration file on each node. (If the `60-galera.cnf` file doesn't exist on your Ubuntu server, then edit `/etc/mysql/my.cnf` or `/etc/my.cnf` config file.)

```
sudo nano /etc/mysql/mariadb.conf.d/60-galera.cnf
```

Change the configuration to the following in the `[galera]` unit.

```
[galera]
# Mandatory settings
wsrep_on                 = ON
wsrep_provider           = /usr/lib/galera/libgalera_smm.so
wsrep_cluster_name       = "MariaDB Galera Cluster"
wsrep_cluster_address    =
"gcomm://IP_address_of_node1,IP_address_of_node2,IP_address_of_node3"
binlog_format            = row
default_storage_engine   = InnoDB
innodb_autoinc_lock_mode = 2
innodb_force_primary_key = 1
innodb_doublewrite       = 1

# Allow server to accept connections on all interfaces.
bind-address = 0.0.0.0

# Optional settings
```

```
wsrep_slave_threads            = 4
innodb_flush_log_at_trx_commit = 0
wsrep_node_name                = MyNode1
wsrep_node_address             = "IP_address_of_this_node"

# By default, MariaDB error logs are sent to journald, which can be hard to digest sometimes.
# The following line will save error messages to a plain file.
log_error = /var/log/mysql/error.log
```

Mandatory settings

1. The first variable enables write-set replication.
2. The second variable specifies the location of the wsrep library. Usually, it's `/usr/lib/galera/libgalera_smm.so`. The `/usr/lib/libgalera_smm.so` file is a symbolic link.
3. The third variable sets a name for the cluster. You need to use the same cluster name on every node in the cluster.
4. The fourth variable defines the IP address of every node in the cluster, separated by comma.
5. Binary log is needed for Galera cluster and its format must be `ROW`. Statement-based or mixed replication isn't supported.
6. Galera only supports InnoDB or its fork XtraDB, so it's important to set the `default_storage_engine` variable.
7. The `innodb_autoinc_locak_mode` variable has 3 possible values: 0, 1 or 2. We must set it to 2 (interleaved lock mode) for Galera cluster.
8. Galera cluster requires all tables having a primary key (Invisible primary key is not supported). So it's a good idea to enforce a primary key on every table. CREATE TABLE without primary key will not be accepted, and will return an error. This is also true when you import a database.
9. InnoDB doublewrite buffer is enabled by default and it should not be changed when using Galera wsrep provider library version 3.

You must set the bind-address to `0.0.0.0` to make MariaDB server listen on the public IP address, so it can communicate with other nodes.

Optional Settings

1. The first variable sets the number of threads that will be used to process writesets from other nodes. More threads can speed up replications. The default value is 1, but a good starting point is 4x the number of CPU cores. It's recommended that you use the same CPU cores on each node and set `wsrep_slave_threads` to the same value on each node.
2. The second variable ensures that the InnoDB log buffer is written to file once per second, rather than on each transaction commit, to improve performance. Note that if all cluster

nodes goes down at the same time, the last second of transaction will be lost because of this line. If the cluster nodes are spread across different data centers, then no need to worry about this.
3. The third variable sets a name for an individual node, so you can easily identify each node when viewing the logs.
4. The last variable sets the IP address for an individual node.

You can also add the following lines in this file so that MariaDB will log error messages to a text file.

```
log_error = /var/log/mysql/error.log
```

If you are running MariaDB 10.1 server, you also need to add the following line to disable XA transactions because it's not supported by Galera.

```
innodb_support_xa = 0
```

If you run MariaDB 10.3 or above, you should not add this line because it's on by default and it can't be disabled. It's said to be fully supported in MariaDB 10.4 Galera cluster.

Note: Old version of Galera cluster doesn't support query cache (query_cache_size). It's supported by all current versions of MariaDB Galera.

Save and close the file. Don't restart MariaDB server now.

Step 3: Opening Network Ports in Firewall on Each Node

Galera cluster requires constant communication between all the nodes due to the use of synchronous replication and they communicate with each other using the following TCP ports.

- 3306 (standard MariaDB port)
- 4444 (SST port)
- 4567 (Galera replication port)
- 4568 (IST port)

You need to configure firewall to allow traffic to these ports from the IP addresses of the cluster nodes. If you are using UFW, you can run the following commands on each node.

```
sudo ufw insert 1 allow in from IP_Address_of_node1
```

```
sudo ufw insert 1 allow in from IP_Address_of_node2

sudo ufw insert 1 allow in from IP_Address_of_node3
```

If you use iptables, then run the following commands.

```
sudo iptables -I INPUT -p tcp --source IP_address_of_node1 -j ACCEPT

sudo iptables -I INPUT -p tcp --source IP_address_of_node2 -j ACCEPT

sudo iptables -I INPUT -p tcp --source IP_address_of_node3 -j ACCEPT
```

Step 4: Configuring AppArmor for mysqld

AppArmor is enabled by default on Ubuntu and it can block communication on non-standard MariaDB ports, preventing Galera cluster from working, so we need to add AppArmor policy to allow MariaDB to open additional non-standard ports with the following commands.

```
cd /etc/apparmor.d/disable/

sudo ln -s /etc/apparmor.d/usr.sbin.mysqld

sudo systemctl restart apparmor
```

Note that newer versions of MariaDB server package for Ubuntu ship with an empty AppArmor profile (`/etc/apparmor.d/usr.sbin.mysqld`), effectively disabling AppArmor for MariaDB, so you don't need to run the above commands anymore.

Step 5: Starting the Cluster

Now we need to start the cluster **primary component** on the first node. Choose a node that has your database as the first node and stop the MariaDB server on the first node.

```
sudo systemctl stop mariadb
```

Then run the following command to start the **primary component** on the first node. (Note: If you

don’t stop MariaDB first, then the following command has no effect.)

```
sudo galera_new_cluster
```

Now you can log into MariaDB monitor.

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

And check the cluster size.

```
show status like 'wsrep_cluster_size';
```

You will see that there’s only 1 node in the cluster.

```
MariaDB [(none)]> show status like 'wsrep_cluster_size';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 1     |
+--------------------+-------+
1 row in set (0.001 sec)
```

To add other nodes to the cluster, simply restart MariaDB server on other nodes. (Note: if the other nodes have other databases, then those databases will be deleted. Only databases from the first node will exist.)

```
sudo systemctl restart mariadb
```

This command may take a while to complete, because when the new nodes join the cluster, they need to do a snapshot state transfer (SST), i.e. copy the databases from the first node, which can consume a lot of RAM and bandwidth. You can check the SST log with:

```
sudo journalctl -eu mariadb
```

After the other two nodes successfully joined the cluster, the cluster size changes to 3.

```
MariaDB [(none)]> show status like 'wsrep_cluster_size';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
1 row in set (0.002 sec)
```

I once had a node that failed to join the cluster and MariaDB log shows the following error.

```
Internal MariaDB error code: 1146
```

I simply restart MariaDB again and the error was gone.

If you import a new database on any of the nodes now, this database will be replicated to other nodes. To check if data modifications has been synced, run the following statement at the MariaDB monitor.

```
show status like 'wsrep_local_state_comment';
```

```
MariaDB [(none)]> show status like 'wsrep_local_state_comment';
+----------------------------+--------+
| Variable_name              | Value  |
+----------------------------+--------+
| wsrep_local_state_comment | Synced |
+----------------------------+--------+
1 row in set (0.001 sec)
```

You can check other Galera status with:

```
show status like 'wsrep%';
```

If any of the nodes, including the the first one, crashes and be kicked out of the cluster as a result, you just need to restart the MariaDB server and the crashed node will rejoin the cluster. You must not run the `sudo galera_new_cluster` command again unless the cluster shuts down (All nodes in the cluster are offline).

Tips For WordPress Users

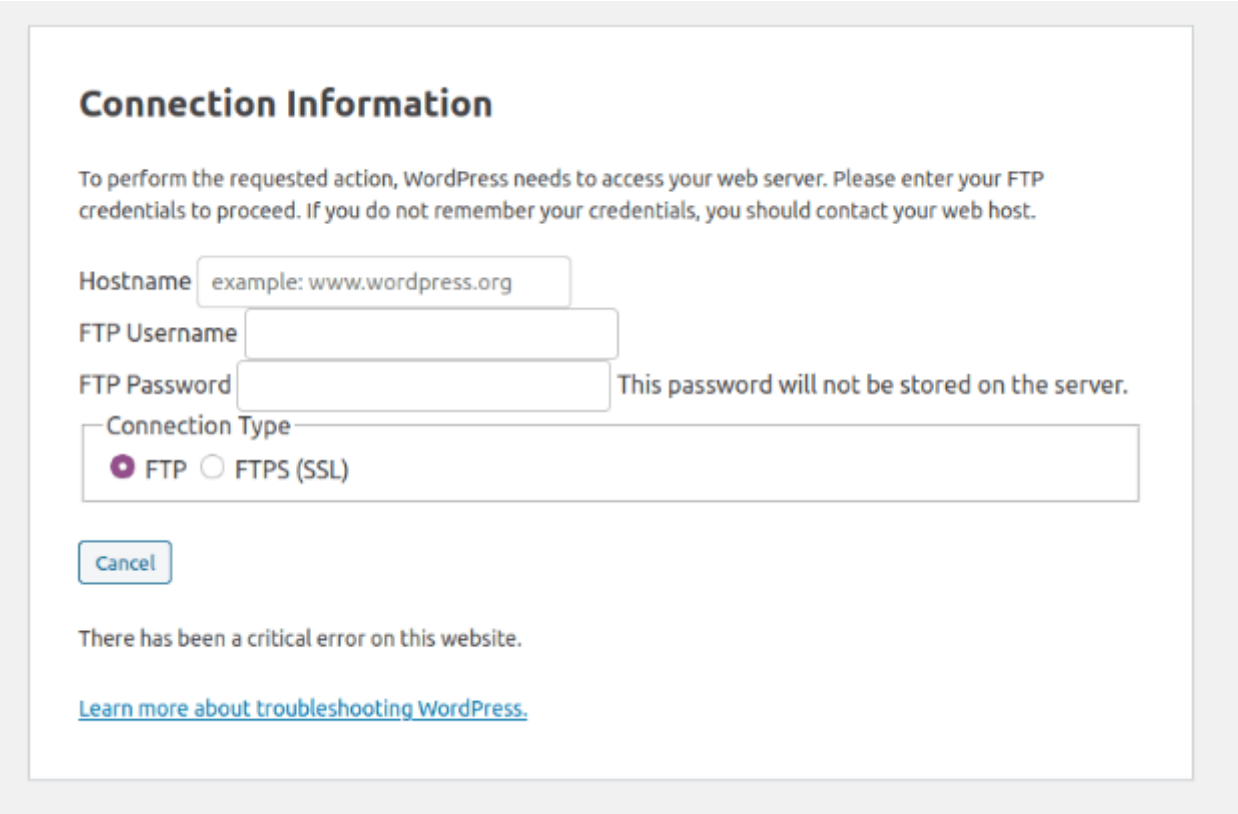
Galera cluster works well with WordPress. Every time you publish a new article, change settings in the WordPress dashboard, or post a new comment on the articles, the changes will be replicated to all nodes in the cluster. However, you need to take care of a few things to make it work smoothly.

As mentioned before, Galera cluster requires every table in the database having a primary key. WordPress core tables all have primary key. However, some plugins may create tables without primary key in your WordPress database. Here’s what you should do to make sure all tables have primary key.

First, I recommend using the Plugins Garbage Collector to remove leftover tables in your

WordPress database. Then you should dump your WordPress database and import it on one of the Gelera nodes. If all of the tables have primary key, the import will finish without error. If any table in the WordPress database doesn't have a primary key, then the import will fail. This is because we added the `innodb_force_primary_key = 1` parameter in MariaDB Galera configuration.

If you see following error when trying to leave a comment on your WordPress website,



then you need to add the following line in your `wp-config.php` file.

```
define('FS_METHOD', 'direct' );
```

MariaDB Galera cluster works well with Nginx FastCGI cache.

Dropping a Node From MariaDB Galera Cluster

First, log into MariaDB monitor and run the following statement:

```
show status like 'wsrep_local_state_comment';
```

If the state is synced, you can safely dropping the node from cluster by stopping MariaDB server.

```
sudo systemctl stop mariadb
```

On the other two nodes, run the following statement at the MariaDB monitor.

```
show status like 'wsrep%';
```

The `wsrep_cluster_size` changes to 2 and the IP address of the dropped nodes isn't listed in `wsrep_incoming_address` any more, which indicates the node has been successfully dropped.

To rejoin the cluster, simply restart MariaDB again.

```
sudo systemctl restart mariadb
```

If you don't want a node to join the cluster again, then delete the Galera related settings in the main configuration file and restart MariaDB.

To drop a node without stopping MariaDB server, you need to lock the tables.

```
MariaDB [(none)]> flush tables with read lock;
```

This way you can create a backup using *mysqldump*. After that, unlock the tables for this node to rejoin the cluster.

```
MariaDB [(none)]> unlock tables;
```

Adding New Nodes to the Cluster

Galera cluster requires at least 3 nodes to be crash-safe and it's recommended that you add more nodes to the cluster to make it more robust. You don't need to shut down the cluster in order to add new nodes to the cluster. Instead, you need to

1. Add the Galera configurations in the `60-galera.conf` file on the new nodes, open network port in firewall and update AppArmor policy.
2. Add the IP addresses of new nodes in the `wsrep_cluster_address` variable on each node.
3. Restart MariaDB server on existing nodes in the cluster one by one. (Only restart the next MariaDB server after the previous one has finished restarting.)
4. Restart MariaDB server on the new nodes so that they can join the cluster.

Hint: Always deploy an odd number of nodes to Galera cluster.

Shutting Down or Restarting MariaDB Galera Cluster

The cluster disappears when all nodes are offline at the same time. To shut down the cluster, you need to shut down all nodes. First, make sure that your application isn't using the database and the `wsrep_local_state_comment` is synced. Then shut down MariaDB server one by one.

To restart the Galera Cluster, run the following command on the last node to leave the cluster.

```
sudo galera_new_cluster
```

Then start MariaDB server on other nodes one by one.

```
sudo systemctl start mariadb
```

Galera Cluster Health

Sometimes there will be a **network partition** in the cluster due to network connectivity failure. For example, if one node loses network connectivity with the other two nodes, then this node will change from a **primary** component to **non-primary** component. The other two nodes can connect to each other and they will still be in the primary component.

You can check this status with the following statements at the MariaDB monitor.

```
show status like 'wsrep_cluster_status';
```

```
MariaDB [(none)]> show status like 'wsrep_cluster_status';
+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
1 row in set (0.001 sec)
```

When a node is in non-primary component, both read and write queries will be disabled on that node. To rejoin it to the primary component, simply restart MariaDB server on the node in non-

primary component.

```
sudo systemctl restart mariadb
```

Improving Galera Cluster Performance

Don’t use swap space.

Do not use swap space on your server. It will make MariaDB slow. If your server is short of RAM, add more physical RAM to it instead of using swap space. You can use the following command to disable swap space on Linux.

```
sudo swapoff -a
```

You should keep enough free RAM on your server, so MariaDB won’t be killed due to out-of-memory problem.

Monitor Latency

You can monitor Galera cluster replication latency by executing the following command at the MariaDB shell. Obviously, the smaller the latency, the faster the replication will be.

```
show status like 'wsrep_evs_repl_latency';
```

The replication latency is the sum of:

- network latency
- time spent on applying writeset.

Applier Thread Count

If there are more applier threads on each node, then the replication will be faster. MariaDB uses 1 applier thread by default. In this tutorial, we set it to use 4 threads as a starting point in the `/etc/mysql/mariadb.conf.d/60-galera.cnf` file.

```
wsrep_slave_threads = 4
```

As a rule of thumb, the optimal value should be

- 4x the number of your CPU cores
- and less than the value of `wsrep_cert_deps_distance`.

`wsrep_cert_deps_distance` is a MariaDB status variable that shows the average number of writesets that can be safely applied simultaneously. Its value is constantly changing. You need to be very careful because if the number of applier threads exceeds this value, your database could become desynchronized.

On my MariaDB database server, the value is highly unstable. I have seen it changing from 13 to 132. So I never use more than 8 applier threads.

```
wsrep_slave_threads = 8
```

Troubleshooting Tips

You can check the MariaDB error log (`/var/log/mysql/error.log` and `sudo journalctl -eu mariadb`) to debug problems. The default error log verbosity is 2. You can increase the verbosity level during debugging by executing the following SQL command at the MariaDB monitor. The log_warnings variable can be changed at run time, without restarting MariaDB server.

```
SET GLOBAL log_warnings=3;
```

The maximum verbosity level is 9. After increasing this value, you might see some warning messages like the one below.

```
[Warning] Aborted connection 14 to db: 'db_name' user: 'db_user' host: 'localhost' (This
connection closed normally)
```

This is a warning message, not an error message. You can decrease the log verbosity and ignore this warning message.

Setting Up a Galera Arbitrator

What if you are on a budget and don’t have money to spend on the same hardware specs for the third node? You can use a Galera Arbitrator as a replacement, so the third node doesn’t have to be as powerful as the other nodes. A Galera Arbitrator will be counted as a valid node in the cluster,

but does not replicate the database changes. It doesn't need to run a MariaDB database server, but will run the lightweight `garbd.service`. It's very easy to set up.

Install the `galera-arbitrator-4` package.

```
sudo apt install galera-arbitrator-4
```

We need to edit the config file before starting it.

```
sudo nano /etc/default/garb
```

Add the following lines

```
GALERA_NODES="IP_address_of_node1,IP_address_of_node2,IP_address_of_node3"

GALERA_GROUP="MariaDB Galera Cluster"
```

Save and close the file. Then start `garbd.service`.

```
sudo systemctl start garbd
```

Enable auto-start at system boot time.

```
sudo systemctl enable garbd
```

Check its status.

```
systemctl status garbd
```

If everything went well, the garbd service should be running and you can see the following message in the log (`sudo journalctl -eu garb`).

```
INFO: Member 1.0 (garb) synced with group.
INFO: Shifting JOINED -> SYNCED (TO: 638250)
```

How to Encrypt Replication Traffic in MariaDB Galera Cluster on Ubuntu

Previously, we discussed how to set up MariaDB Galera multi-master cluster on Ubuntu. By default, replication traffic in MariaDB Galera cluster is unencrypted. This tutorial will be showing you how to Enable TLS encryption so nodes in the cluster can communicate with each other securely over the public Internet. There are 2 types of replication traffic in Galera cluster:

- State transfer
- Write-set replication

What is State Transfer?

State transfer is the process of copying a database from one node to another. The node sending data is called a donor. The node receiving data is called a joiner. There are two kinds of state transfer:

- Snapshot state transfer (SST): copy entire database
- Incremental state transfer (IST): copy data modifications only

SST, aka node provisioning, is used when a new node joins the cluster because it contains no data. IST is used when a disconnected node rejoins the cluster.

Understanding SST

There are 5 SST methods:

- wsrep_sst_rsync: default method
- wsrep_sst_mysqldump: slowest method
- wsrep_sst_mariabackup: non-blocking method
- wsrep_sst_xtrabackup
- wsrep_sst_xtrabackup-v2

Rsync is the default SST method and it's the fastest. It's recommended that you do not use **xtrabackup** or **xtrabackup-v2** method because they don't work with MariaDB 10.3 + and they don't support GTID and data at rest encryption.

Personally, I prefer **mariabackup** because it doesn't block the donor node during the state transfer, which is an advantage inheritated from the xtrabackup method. Rsync and mysqldump require the donor made read-only during the transfer, via the `flush tables with read lock` command. The mariabackup SST method is a fork of the xtrabackup-v2 SST method and it relies on the `mariabackup` and `socat` command-line utility.

To set **mariabackup** as the SST method, first install `socat` and `mariabackup` on each node in the cluster and the joiner node.

```
sudo apt install socat
```

If you installed the latest version of MariaDB from mariadb.org repository, then you need to install `mariabackup` with the following command.

```
sudo apt install mariadb-backup
```

Then add the following two lines in the `[galera]` unit of the `/etc/mysql/mariadb.conf.d/60-galera.cnf` file, on each node in the cluster and the joiner node.

```
[galera]
...
...
wsrep_sst_method = mariabackup
wsrep_sst_auth = mariabackup:your_password
```

The first line specifies that `mariabackup` will be used for SST method and the second line defines the username and password, which is needed for authentication.

Next, log into MariaDB monitor on a node that's already in the cluster and create the database user and grant necessary permissions. (Note that you do not need to run the following commands on each node. The data will be replicated to other nodes automatically.)

```
create user 'mariabackup'@'localhost' identified by 'your_password';
grant reload, process, lock tables, replication client on *.* to 'mariabackup'@'localhost';
```

Flush privilege tables and exit;

```
flush privileges;


exit;
```

Restart MariaDB server on each node one by one and then restart the MariaDB server on the joiner node.

```
sudo systemctl restart mariadb
```

Encrypting Mariabackup SST

You can use openssl to create a self-signed TLS certificate, but I'm going to use the existing Let's Encrypt TLS certificate configured for my web server. To enable TLS encryption on the mariabackup SST, open the MariaDB main configuration file (`/etc/mysql/mariadb.conf.d/50-server.cnf` or `/etc/mysql/my.cnf`), and add the following lines **at the end of the file**. Replace the path name as necessary.

```
[sst]
encrypt=4
tkey=/etc/letsencrypt/live/linuxbabe.com/privkey.pem
tcert=/etc/letsencrypt/live/linuxbabe.com/cert.pem
tca=/etc/letsencrypt/live/linuxbabe.com/chain.pem
```

The `encrypt` variable has 5 possible values: 0, 1, 2, 3, 4. `0` means encryption is disabled. `1`, `2` and `3` doesn't work any more. 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/ /etc/letsencrypt/live/
```

Note that you need to use the same TLS certificate and private key on all your nodes in the cluster and then restart the entire Galera Cluster for the changes to take effect.

Note that during SST, extra files in `/var/lib/mysql/` directory added by user will be deleted.

Understanding IST

IST is used when a disconnected node rejoins the cluster. All nodes in the cluster maintain a Galera cache (GCache), aka write-set cache, which contains the write sets committed by a node. When a disconnected node rejoins the cluster, it requests an incremental state transfer from a donor's GCache. IST does not lock the donor.

Obviously, IST is faster than SST, because it transfers modifications only. However, if the donor's GCache isn't big enough to store all the write sets the joiner needs, an SST will be initiated instead. To avoid SST, you can increase the size for GCache. The default value is 128MB, which you can check with the following command at the MariaDB monitor.

```
show variables like 'wsrep_provider_options'\G
```

GCache related values are as follows:

```
gcache.dir = /var/lib/mysql/; gcache.keep_pages_size = 0; gcache.mem_size = 0; gcache.name = /var/lib/mysql//galera.cache; gcache.page_size = 128M; gcache.recover = no; gcache.size = 128M;
```

The GCache file is located at `/var/lib/mysql/galera.cache` and its size is preallocated. To increase the size of GCache, you need to add the following line in `[mysqld]` unit of MariaDB configuration file.

```
wsrep_provider_options="gcache.size = 1G"
```

Replace 1G with your preferred cache size. Make sure your server has enough RAM. Setting a big GCache size with a small amount of RAM can cause your MariaDB server to fail to start. It's also recommended to set the `gcache.recover` parameter to `yes`, so a node can attempt to recover its Gcache on startup and continue serving IST to other joining nodes.

```
wsrep_provider_options="gcache.size = 1G; gcache.recover = yes"
```

Apply the same configuration on each node in the cluster and restart them one at a time.

Encrypting IST

I tried to use Let's Encrypt TLS certificate to encrypt IST traffic, but I always get the following error in the error log file when a node joins the cluster.

```
[ERROR] WSREP: handshake with remote endpoint ssl://xx.xx.xx.xx:4567 failed:
asio.ssl:337047686: 'certificate verify failed' ( 337047686: 'error:1416F086:SSL
routines:tls_process_server_certificate:certificate verify failed')
```

So I created a self-signed server certificate and private key. The steps are as follows.

Create a directory to hold SSL files.

```
sudo mkdir /etc/ssl/mysql/
```

Change directory.

```
cd /etc/ssl/mysql/
```

Generate the CA key file:

```
sudo openssl genrsa 2048 > ca-key.pem
```

Generate the CA certificate file. We use a very long expiration day (100 years) to avoid shutting down the cluster.

```
sudo openssl req -new -x509 -nodes -days 36500 -key ca-key.pem -out ca.pem
```

You will need to answer a few questions. I just entered the country code and the organization name.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example CA
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

Generate the server key file. You will also need to answer a few questions and enter a passphrase to protect the key.

```
sudo openssl req -newkey rsa:2048 -days 36500 -nodes -keyout server-key.pem -out server-req.pem
```

Remove the passphrase.

```
sudo openssl rsa -in server-key.pem -out server-key.pem
```

Generate the server certificate file:

```
openssl x509 -req -in server-req.pem -days 36500 -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem
```

Then copy all the files to other nodes in the cluster with `scp` utility and it's recommended to store them in the same directory (`/etc/ssl/mysql/`). Also make sure the mysql user has permission to read all the SSL files.

```
sudo chown mysql:mysql /etc/ssl/mysql/ -R
sudo chmod 400 /etc/ssl/mysql/*
sudo chmod 700 /etc/ssl/mysql/
```

After that, open the MariaDB main configuration file (`/etc/mysql/mariadb.conf.d/50-server.cnf` or `/etc/mysql/my.cnf`), and add the following lines in the `[mysqld]` unit.

```
wsrep_provider_options="socket.ssl_key=/etc/ssl/mysql/server-key.pem;socket.ssl_cert=/etc/ssl/mysql/server-cert.pem;socket.ssl_ca=/etc/ssl/mysql/ca.pem"
```

If you have other wsrep provider options, they need to be combined together like below.

```
wsrep_provider_options="gcache.size = 1G; gcache.recover = yes;socket.ssl_key=/etc/ssl/mysql/server-key.pem;socket.ssl_cert=/etc/ssl/mysql/server-cert.pem;socket.ssl_ca=/etc/ssl/mysql/ca.pem"
```

Configure each node in the cluster and restart the entire cluster for the changes to take effect. If everything goes well, your node will be able to join the cluster and you can see message like below in your MariaDB log (`sudo journalctl -eu mariadb`), which indicates IST is encrypted. (IST uses TCP port 4568.)

```
2019-03-19 14:40:03 1 [Note] WSREP: IST sender using ssl
2019-03-19 14:40:03 0 [Note] WSREP: async IST sender starting to serve ssl://xx.xx.xx.xx:4568
```

```
sending 166613-167529
```

Encrypting Write-set Replication Traffic

Write-set replication is the normal synchronous replication whereby a node sends data modifications to all other nodes. Actually, write-set replication uses the same encryption machenism as IST. If you enabled TLS encryption for IST, write-set replication is also encrypted and you will see messages like below in MariaDB log (`sudo journalctl -eu mariadb`), which indicates write-set replication is secured with TLS. (Write-set replication uses TCP port 4567.)

```
SSL handshake successful, remote endpoint ssl://xx.xx.xx.xx:4567 local endpoint
ssl://xx.xx.xx.xx:37118 cipher: TLS_AES_256_GCM_SHA384 compression: none
```

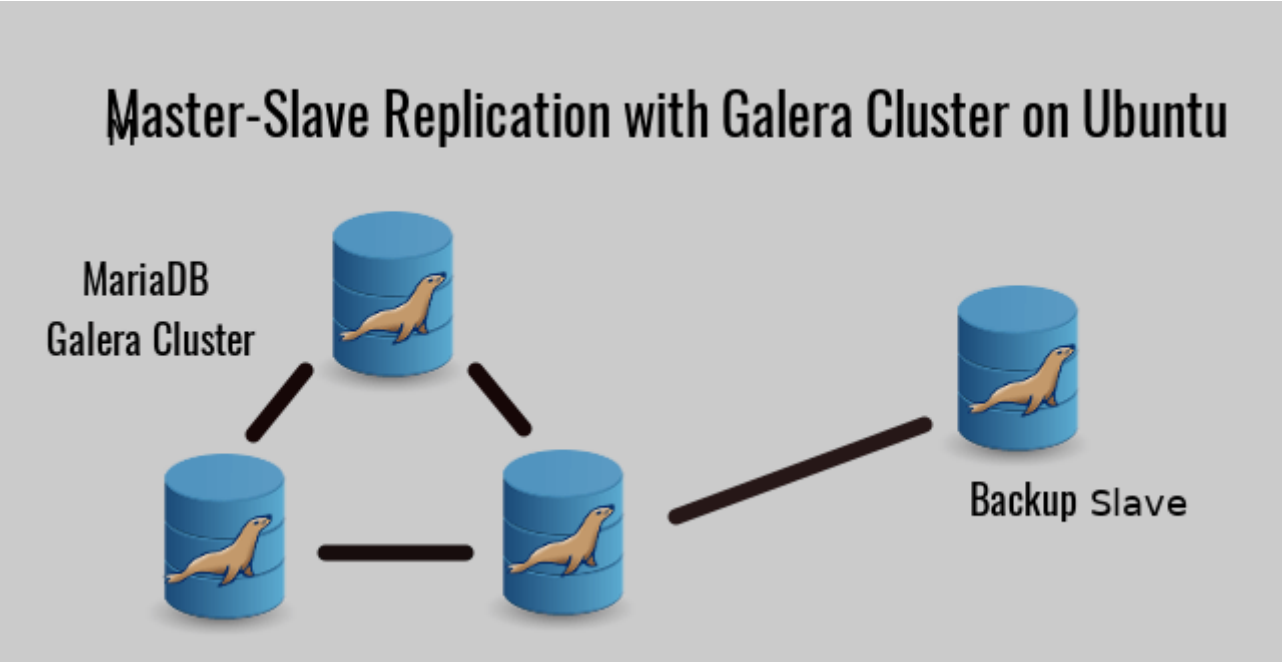
Set Up MariaDB Master-Slave Replication with Galera Cluster on Ubuntu

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

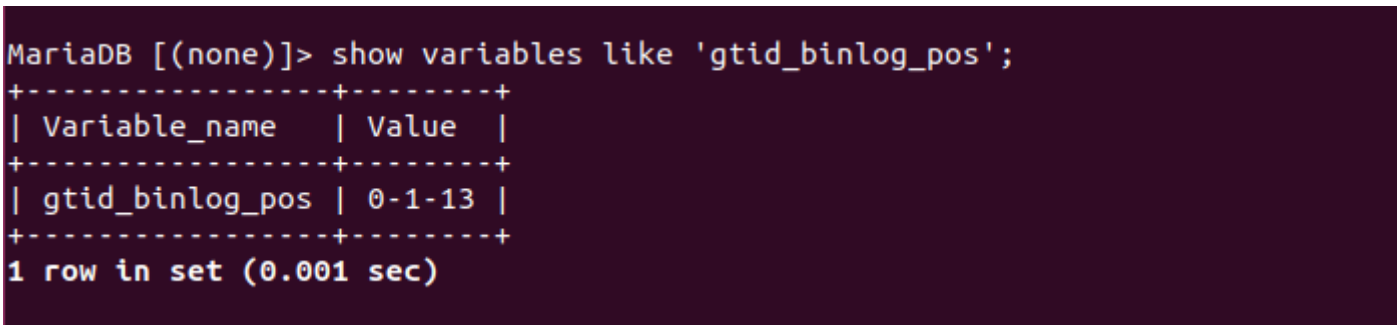
Choose one Galera node as the master and log into the master MariaDB monitor. Run the following command to prevent any further changes to databases.

```
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.