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.

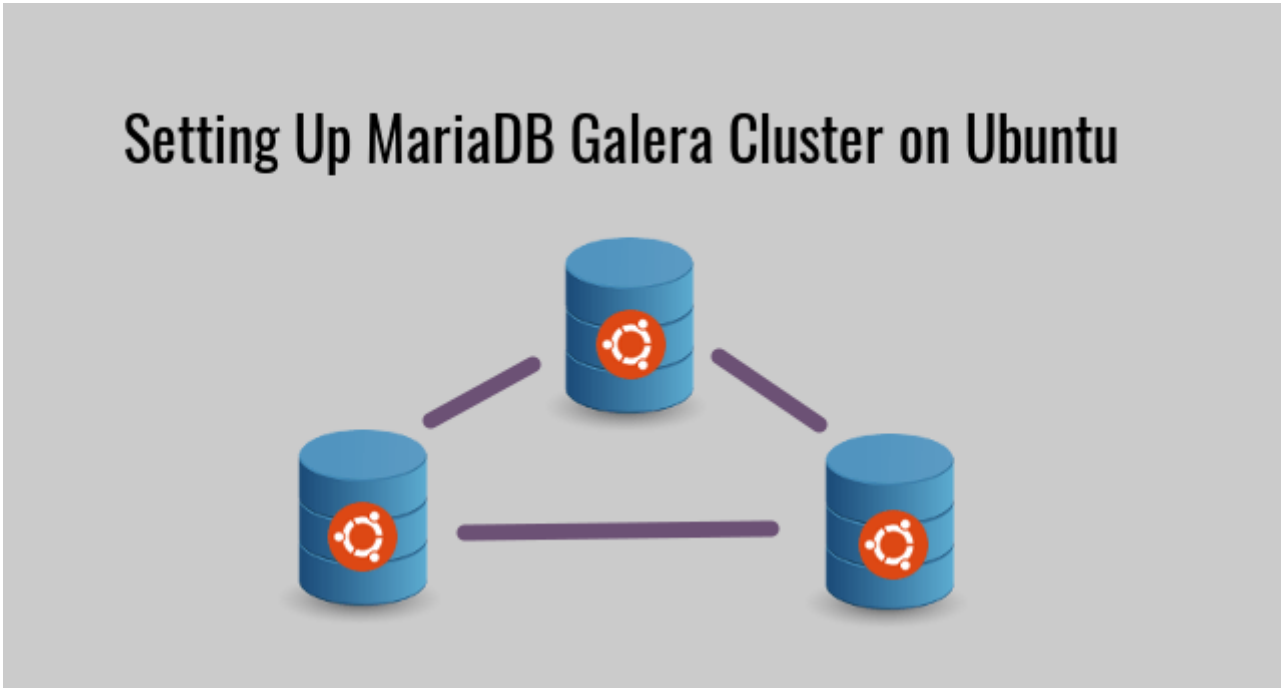
What is Galera Cluster?

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

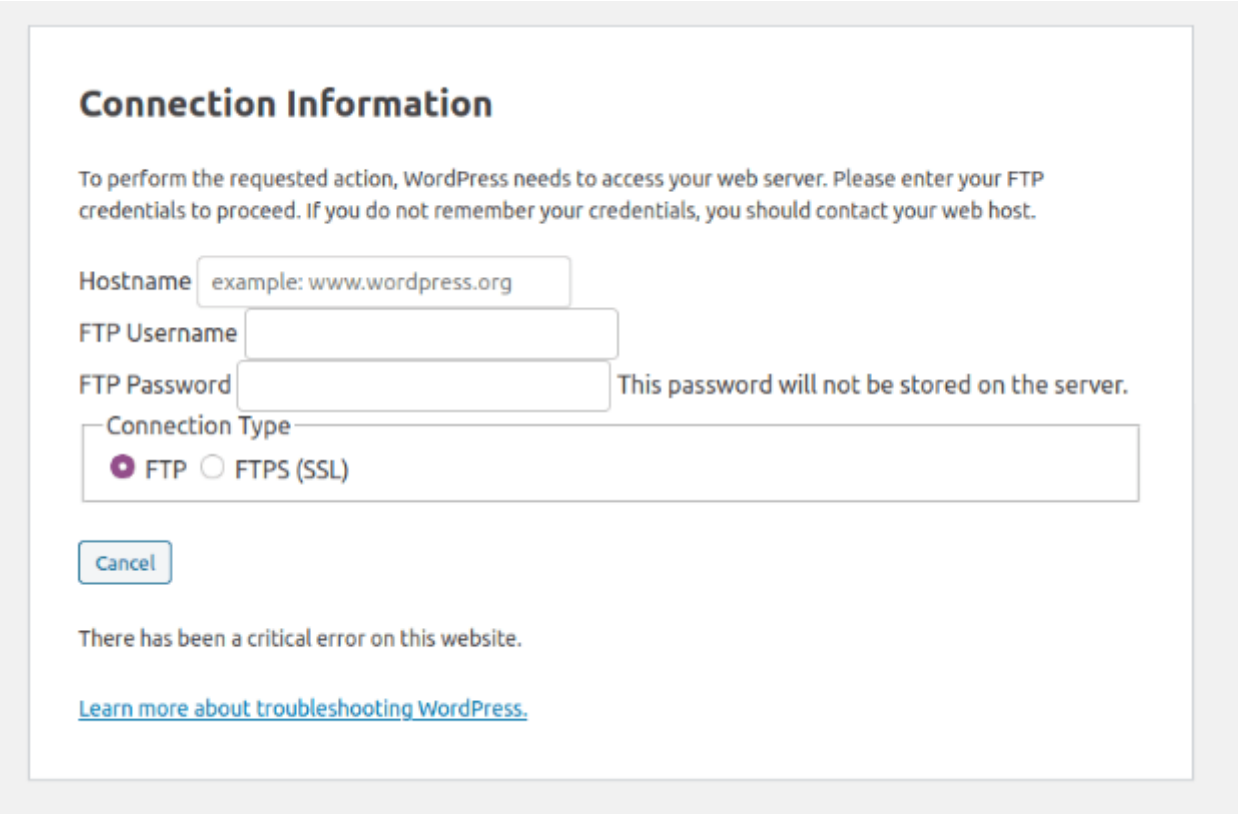
Galera cluster works well with WordPress. Every time you publish a new article, change settings in the WordPress dashboard, or post a new comment on the articles, the changes will be replicated to all nodes in the cluster. However, you need to take care of a few things to make it work smoothly.

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)
```

Revision #1
Created 11 July 2021 19:10:23 by Admin
Updated 11 July 2021 19:11:24 by Admin