MariaDB Configuration

All docs related to MariaDB configurations

[How-To] Change the Default Data Directory for MariaDB Server

Purpose

This documentation will focus on the process of changing the default data directory from its mysql default to a desired location. This will not focus on how to create the remote directory or how to mount it to the db server.

Prerequisites

List of prerequisites:

- Root user or sudo user
- MariaDB Server
- MariaDB admin creds

Changing the default MySQL/MariaDB Data Directory

Note: We are going to assume that our new data directory is `/mnt/mysql-data`. It is important to note that this directory should be owned by `mysql:mysql`.

```
# mkdir /mnt/mysql-data
# chown -R mysql:mysql /mnt/mysql-data
```

For your convenience, we’ve divided the process into 5 easy-to-follow steps:

Step 1: Identify Current MySQL Data Directory

To begin, it is worthy and well to identify the current data directory using the following command. Do not just assume it is still `/var/lib/mysql` since it could have been changed in the past.

```
# mysql -u root -p -e "SELECT @@datadir;"
```

After you enter the MySQL password, the output should be similar to.

Step 2: Copy MySQL Data Directory to a New Location

To avoid data corruption, stop the service if it is currently running before proceeding. Use the **systemd** well-known commands to do so:

```
------------- On SystemD -------------
# systemctl stop mariadb
# systemctl is-active mariadb

------------- On SysVInit -------------
# service mysqld stop
# service mysqld status

OR

# service mysql stop
# service mysql status
```

If the service has been brought down, the output of the last command should be as follows:

Then copy recursively the contents of `/var/lib/mysql` to `/mnt/mysql-data` preserving original permissions and timestamps:

```
# cp -R -p /var/lib/mysql/* /mnt/mysql-data
```

Step 3: Configure a New MySQL Data Directory

Edit the configuration file (`my.cnf`) to indicate the new data directory (`/mnt/mysql-data` in this case).

```
# vi /etc/my.cnf
OR
# vi /etc/mysql/my.cnf
```

Locate the `[mysqld]` and `[client]` sections and make the following changes:

```
Under [mysqld]:
datadir=/mnt/mysql-data
socket=/mnt/mysql-data/mysql.sock

Under [client]:
port=3306
socket=/mnt/mysql-data/mysql.sock
```

Save the changes and then proceed with the next step.

Step 4: Start the MariaDB Service

```
# systemctl start mariadb
# systemctl is-active mariadb
```

Now, use the same command as in **Step 1** to verify the location of the new data directory:

```
# mysql -u root -p -e "SELECT @@datadir;"
```

Step 5: Create MySQL Database to Confirm Data Directory

Login to MariaDB, create a new database and then check `/mnt/mysql-data`:

```
# mysql -u root -p -e "CREATE DATABASE tecmint;"
```

Congratulations! You have successfully changed the data directory for MySQL or MariaDB.

Documentation derived from: How to Change a Default MySQL/MariaDB Data Directory in Linux (tecmint.com)

[How-To] Migrate MariaDB Database from local to remote MariaDB Server

Purpose

Show how to migrate MariaDB database from local to remote MariaDB server

Prerequisites

List of prerequisites:

- Root user or sudo user
- MariaDB Servers
- MariaDB Admin user

Instructions

Step 1: Backup your WordPress Database

1. **Local Database Backup:**
    - Log in to your current WordPress site's server.
    - Use a tool like `mysqldump` to create a backup of your local WordPress database. Run the following command:

    -
      ```
      mysqldump -u [username] -p[password] [database_name] > local_backup.sql
      ```

    Replace `[username]`, `[password]`, and `[database_name]` with your MariaDB username, password, and WordPress database name, respectively.
2. **Transfer Backup to Remote Server:**
    - Copy the `local_backup.sql` file to your dedicated MariaDB server. You can use secure copy (SCP) or any other method you prefer.

Step 2: Create a Database on Remote Server

1. **Access Remote MariaDB:**
   - Log in to your dedicated MariaDB server.
2. **Create a New Database:**
   - In the MariaDB shell, create a new database for your WordPress site:

   -
     ```
     CREATE DATABASE new_database;
     ```
3. **Create a Database User:**
   - Create a user and grant necessary privileges:

   -
     ```
     CREATE USER 'new_user'@'%' IDENTIFIED BY 'password'; GRANT ALL PRIVILEGES ON new_database.* TO 'new_user'@'%'; FLUSH PRIVILEGES;
     ```

   Replace `'new_user'` and `'password'` with your preferred username and password.

Step 3: Import Database to Remote Server

1. **Navigate to the Backup Location:**
   - On your dedicated MariaDB server, navigate to the directory where you transferred the `local_backup.sql` file.
2. **Import Database:**
   - Use the following command to import the database:

   -
     ```
     mysql -u [username] -p[password] [database_name] < local_backup.sql
     ```

   Replace `[username]`, `[password]`, and `[database_name]` with the new database username, password, and name.

Step 4: Update WordPress Configuration

1. **Edit `wp-config.php`:**
   - On your WordPress site's server, locate the `wp-config.php` file.
   - Update the database connection details:

   -
     ```
     define('DB_NAME', 'new_database');
     define('DB_USER', 'new_user');
     define('DB_PASSWORD', 'password');
     define('DB_HOST', 'remote_server_ip');
     ```

Replace `'new_database'`, `'new_user'`, `'password'`, and `'remote_server_ip'` with your database name, username, password, and the IP address of your dedicated MariaDB server.

Step 5: Test and Update URLs

1. **Test the Site:**
   - Visit your WordPress site to ensure everything is working correctly.
2. **Update URLs (if needed):**
   - If your site URLs have changed, update them using a search and replace tool or a plugin like "Better Search Replace."

Step 6: Finalize

1. **Remove Local Database:**
   - Once you've confirmed that your site is working well on the remote database, you can remove the local database.
2. **Review and Optimize:**
   - Take a moment to review your site and ensure that all functionality is working as expected. Additionally, you may want to optimize your database tables.

By following these steps, you should successfully migrate your WordPress database from a local MariaDB server to a remote one. Always make sure to have backups before making any significant changes to your website.

[How-To] Configure Galera Cluster on MariaDB Servers

Purpose

The purpose of this How-To is to explain the process in detail of how to take 3 MariaDB servers and cluster them together in a Multi-Master Galera cluster. For best results, it is recommended to have a minimum of 3 MariaDB servers to achieve quorum and proper redundancy. If you only cluster 2 MariaDB servers together and 1 becomes unavailable, it renders the remaining server unusable. For production environments, minimum of 3 servers and recommended 5 or more.

Prerequisites

List of prerequisites:

- Sudo user
- Minimum 3 Ubuntu 24.04 LTS VMs with MariaDB installed and configured properly (See below for instructions):
  - https://wiki.stretchpowered.com/books/mariadb/page/how-to-install-mariadb-on-ubuntu-2404-lts

Instructions

Step 1: Stop the MariaDB Service on all Servers

First step is to stop the MariaDB service during configuration of Galera as to not get weird results with setup. Run this on all MariaDB servers that will be in the cluster:

```
sudo systemctl stop mariadb
```

Step 2: Make a Backup of the MariaDB Config File on all Servers

Next important step is to make a copy of the MariaDB config file on all affected servers in case you need to go backwards. You can also achieve this with snapshots in your hypervisor:

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

Step 3: Prepare the MariaDB Config File for Galera Cluster Settings

This step is where we actually configure the cluster. This will need to be configured on all affected MariaDB servers but is going to look slightly different on each one of them. So lets edit the config file. On each server you can access the config file by using the following command:

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

Inside the config file, the following should be done on each server, adjusting ips and names of servers to be each one:

```
[mysqld]

# Basic MariaDB Settings
bind-address = 0.0.0.0
default_storage_engine = InnoDB
binlog_format = ROW
innodb_autoinc_lock_mode = 2
# query_cache_type=0
# query_cache_size=0
# log_slave_updates

# Galera Cluster Settings
wsrep_on = ON
wsrep_provider = /usr/lib/galera/libgalera_smm.so
wsrep_cluster_name = "my_galera_cluster"
wsrep_cluster_address = "gcomm://<IP-OF-DB-NODE1>,<IP-OF-DB-NODE2>,<IP-OF-DB-NODE3>"

# Node-specific Settings
wsrep_node_name = "<NAME-OF-THIS-NODE>"
wsrep_node_address = "<IP-OF-THIS-NODE>"

# SST (State Snapshot Transfer) method
wsrep_sst_method = rsync
wsrep_sst_auth = "sst_user:secure_password" # Configure in later step
```

With this config saved, you should be prepared for the Galera cluster.

Step 4: Configure SST User

We need to take a moment to configure the user that we specified for SST in the config for Galera. Run the following commands to do so:

```
sudo mariadb -u root -p
```

```
CREATE USER 'sst_user'@'%' IDENTIFIED BY 'secure_password';
```

```
GRANT RELOAD, LOCK TABLES, PROCESS, REPLICATION CLIENT ON *.* TO 'sst_user'@'%';
```

```
FLUSH PRIVILEGES;
```

```
QUIT;
```

This should be now operation for SST. Continue on to firewall steps.

Step 5: Add UFW Firewall Rules for Galera

Now that our Galera cluster is prepped, we need to open some ports on the VMs UFW firewall to allow for the Galera cluster traffic. Run the following on each MariaDB server in the Galera cluster:

```
sudo ufw allow 4444/tcp
```

```
sudo ufw allow 4567/tcp
```

```
sudo ufw allow 4568/tcp
```

```
sudo ufw reload
```

Step 6: Start the Galera Cluster

Our servers are prepped and ready to go. Time to start the cluster up. On your first node, run the following (This is not the name of your cluster, it is a built in command so don't edit this):

```
sudo galera_new_cluster
```

Now, check the cluster status by logging into MariaDB and checking the `wsrep_cluster_size` value:

```
sudo mariadb -u root -p
```

```
SHOW STATUS LIKE 'wsrep_cluster_size';
```

It should show 1, as we've only started the first. Now, on all remain nodes, run the following to start MariaDB normally:

```
sudo systemctl start mariadb
```

Now, on any of the nodes you have, run the following commands to check cluster size:

```
sudo mariadb -u root -p
```

```
SHOW STATUS LIKE 'wsrep_cluster_size';
```

If you have three nodes, it should return a 3.

Step 7: Verify Functionality and Replication

Everything should be working now so we can test by creating a test database on any of the servers. Remember, Galera is designed to be a Multi-Master clustering solution for MariaDB and allows all nodes to simultaneously be read-write active. So, on any one of your nodes, run the following:

```
sudo mariadb -u root -p
```

```
CREATE DATABASE testdb;
```

```
SHOW DATABASES;
```

You should see your DB. Now, pick another node in the cluster and run the following:

```
sudo mariadb -u root -p
```

```
SHOW DATABASES;
```

You should see your new DB have replicated to the other node. This wraps up this How-To. The next step is to get a Virtual IP that can be used to access your DBs. This can be done with keepalived or haproxy.

[How-To] Configure Virtual IP for MariaDB servers with Keepalived

Purpose

This How-To serves as documentation on how to add keepalived to your MariaDB servers so they can all share a virtual IP or a VIP.

Prerequisites

List of prerequisites:

- Sudo user
- Galera cluster of MariaDB servers on Ubuntu 24.04 LTS

Instructions

Step 1: Update and Install Software

Run commands below to update os and install keepalived:

```
sudo apt update
```

```
sudo apt install -y keepalived
```

Step 2: Configure Keepalived

With keepalived freshly installed, we can open the configuration file on each server with this command:

```
sudo nano /etc/keepalived/keepalived.conf
```

Here, you'll want to configure the master like this:

```
vrrp_instance VI_1 {
    state MASTER
    interface eth0         # Replace with your actual network interface
    virtual_router_id 51
    priority 100           # Higher priority on the master
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass securepassword
    }

    virtual_ipaddress {
        192.168.1.100      # Replace with your VIP
    }

    track_script {
        chk_mariadb
    }
}

vrrp_script chk_mariadb {
    script "/usr/local/bin/check_mariadb.sh"
    interval 2
    weight -20
}
```

Then, you'll want to configure the backups like this:

```
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 90            # Lower priority on the backup
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass securepassword
    }

    virtual_ipaddress {
        192.168.1.100
    }

    track_script {
        chk_mariadb
    }
}

vrrp_script chk_mariadb {
    script "/usr/local/bin/check_mariadb.sh"
    interval 2
    weight -20
}
```

Step 3: Enable and Start Keepalived

Now we're ready to start the service. Run the following to enable it at startup and then start the service once now:

```
sudo systemctl enable keepalived
```

```
sudo systemctl start keepalived
```

Check the status of the service to verify its running:

```
sudo systemctl status keepalived
```

Step 4: Test the VIP

Finally, test that the VIP is there:

```
ip addr show
```