MariaDB Replication with Docker Containers

Introduction:

This guide will give a tutorial on how to setup MariaDB replication between two Docker containers. The steps here could be used for MariaDB installations on bare metal or on virtual machines, but there are some considerations that have been made for Docker. This includes networking, storage of configuration files, and storage of the database itself.

Requirements:

- Docker -
  - Windows and OS X users can download the Docker Toolbox from: https://www.docker.com/products/docker-toolbox. This will install virtualbox, a virtual machine hosting a minimal linux installation that will actually run Docker, the Docker command line terminal, Docker compose, and kitematic: a GUI front end for managing Docker containers.
  - Linux users can run the following command to install the latest version of Docker:

```
mark@mark-ThinkPad-W530:~$ curl -fsSL https://get.docker.com/ | sh
```

- Docker Compose -
  - Docker Compose will already be installed in the Docker Toolbox that comes with Windows and OS X.
  - Linux users will need to install Docker Compose via their system's package manager after having Docker installed.

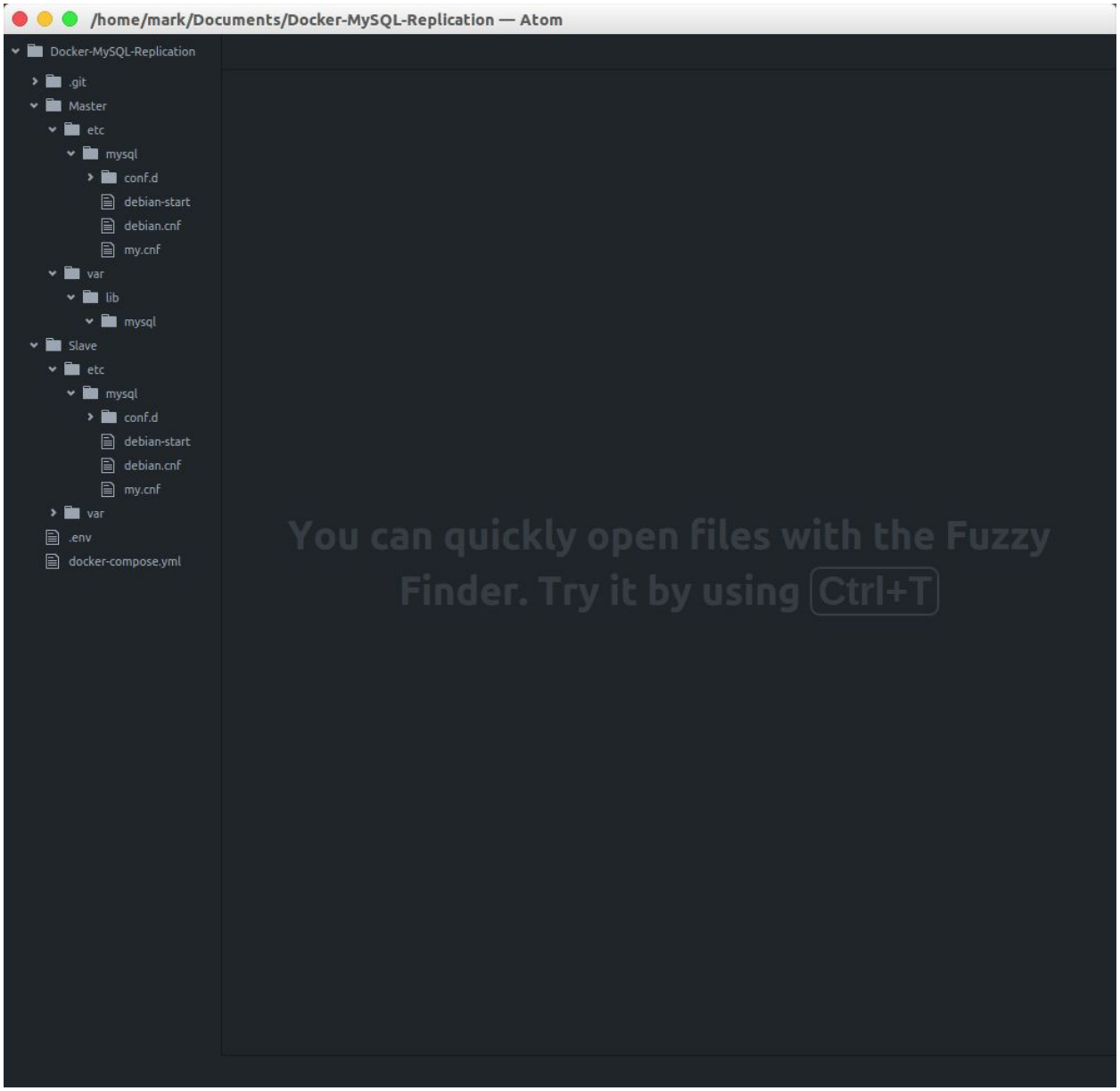
```
mark@mark-ThinkPad-W530:~$ sudo apt-get install docker-compose
```

Setup:

I decided to use Docker compose to spin up my containers, because it allows for granular control and definition of the containers through a .yml or "yaml" file. Compose is a tool for defining and running multi-container Docker applications. Then, using a single command, you create and start all the services from your yml file. Once both Docker and Docker Compose are installed, we are ready to get started. You will need to be connected to the internet to download the latest MariaDB Docker image. If you know you will be testing the replication somewhere without internet access, you can pull the image locally for use later.

```
mark@mark-ThinkPad-W530:~$ docker pull mariadb:latest
```

I recommend downloading this repo and using the atom text editor with this project. It allows for easy navigation of the text files you will be editing.



Networking:

The default method that Docker uses for networking is to join containers to a bridge on the local system called docker0. Docker containers possess standard eth0 interfaces with an IP address in a private network range that is currently not in use by the host, typically 172.x.x.x. This default bridge mode will be sufficient for the Docker containers to communicate with each other. There are methods for giving Docker

containers their own IP's on the network, but this proof of concept does not need those requirements.

Storage:

We want to have control over the configuration files that MariaDB uses as well as the database files themselves. We can do this by using the volumes feature of Docker, which will allow us to mount a folder from the local filesystem to a folder inside the filesystem of the Docker container. This way, we can edit the configuration files before even starting up the container, or do so while the container is running. There are two directories in the Docker containers that we want to mount to our local file system.

- /etc/mysql - Where configuration files are stored.
- /var/lib/mysql - Where the database files are stored.

This can be achieved by using the -v flag with the Docker run command, but we will be demonstrating the use of volumes in the Docker compose yml file.

Environment Variables:

We are using the official MariaDB Docker image from the Docker hub. (https://hub.docker.com/_/mariadb/) This image is created and supported by the MariaDB community, and provides some helpful instruction on how to use it. To use this image, it is necessary to define some environment variables before the container is started:

- MYSQL_ROOT_PASSWORD
- MYSQL_DATABASE
- MYSQL_USER, MYSQL_PASSWORD
- MYSQL_ALLOW_EMPTY_PASSWORD

We will be specifying environment variables in the yml file, but Docker does support the use of passing a file that contains all environment variables.

```
1  MYSQL_ROOT_PASSWORD:1dumbPassword
2  MYSQL_DATABASE:testdb
3  MYSQL_USER:mysqlusr
4  MYSQL_PASSWORD:Password11
5
```

Persistence:

A running Docker container can be saved or committed as it's own launchable image. To do this, you'll need to find the running Docker containers on the system.

```
mark@mark-ThinkPad-W530:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
69d608cd9b97        ubuntu:latest       "/bin/bash"         8 seconds ago       Up 7 seconds                            ecstatic_snyder
```

Notice the name Docker has given the container. Docker assigns each container a unique, yet psuedo-random name based on a list of fun verbs and nouns. You will be using this name to save the running Docker container.

```
mark@mark-ThinkPad-W530:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
69d608cd9b97        ubuntu:latest       "/bin/bash"         8 seconds ago       Up 7 seconds                            ecstatic_snyder
mark@mark-ThinkPad-W530:~$ docker commit ecstatic_snyder ubuntu_example
sha256:d188458691c255b4e457bc77ce9e7b6e5c2521412663 6b3836ff8bf5a4f74fb4
mark@mark-ThinkPad-W530:~$ docker images
REPOSITORY                     TAG                 IMAGE ID            CREATED             SIZE
ubuntu_example                 latest              d188458691c2        5 seconds ago       188 MB
dockermysqlreplication_master  latest              d3bcc8f3e16d        26 hours ago        347.1 MB
dockermysqlreplication_slave   latest              d3bcc8f3e16d        26 hours ago        347.1 MB
ubuntu                         latest              a1e4ed2ac65b        3 days ago          188 MB
mariadb                        latest              b7f6f1349c67        3 days ago          347.1 MB
```

I've renamed the running container from ecstatic_synder to ubuntu_example. I can now run ubuntu_example anytime I want and it will be an exact copy from the moment I committed it. This was achieved by using the command: docker commit.

Consider when it would be best to commit a Docker container. With our configuration, all possible changes to MariaDB and it's database will be stored on the host's filesystem, which could be used again and again to create more Docker containers. It would be unnecessary to commit the container, if there were no changes elsewhere.

For future reference, if you're not interested in saving the Docker image after you have run the container, use the -rm flag in docker run.

Master:

We finally get to start configuring our database instances for replication! The files in this repo were files I copied straight from the Docker container running MariaDB. Open the my.cnf file located in Master/etc/mysql. Insert these lines into the [mysqld] section:

```
[mysqld]
server_id=1
log-basename=master
log-bin
binlog-format=row
binlog-do-db=sakila
```

These settings specify the unique server id for MariaDB, the logging system, and the database to log. There is a sample database included with this project, and is found in the Master folder under: /var/lib/mysql. The sakila-scheme.sql will setup the structure

and tables for the database, and the sakila-data.sql will actually fill the database with information.

Slave:

Here is the configuration used for the slave container in /etc/mysql/my.cnf.

```
[mysqld]
server_id = 2
replicate-do-db=sakila
```

It's time to actually use docker compose. Let's take a look at the yml file.

```
master:
  image: mariadb:latest
  environment:
    MYSQL_ROOT_PASSWORD: 1dumbpassword
    MYSQL_DATABASE: sakila
    MYSQL_USER: mysqlusr
    MYSQL_PASSWORD: password11
  volumes:
    - ./Master/etc/mysql/:/etc/mysql
    - ./Master/var/lib/mysql/:/var/lib/mysql

slave:
  image: mariadb:latest
  environment:
    MYSQL_ROOT_PASSWORD: 1dumbpassword
    MYSQL_DATABASE: sakila
    MYSQL_USER: mysqlusr
    MYSQL_PASSWORD: password11
  volumes:
    - ./Slave/etc/mysql/:/etc/mysql
    - ./Slave/var/lib/mysql/:/var/lib/mysql
```

Notice there are two main sections, one for the master server and one for the slave. For the most part the configurations are identical, except for the volume mounts as we will be using different configuration files for each. This yml file specifies that we are using the MariaDB image from the Docker hub, with the 'latest' tag. We are specifying our environment variables manually in order to configure MariaDB properly. Finally there is a section for the volume mounts.

To start these containers, use this command:

```
mark@mark-ThinkPad-W530:~/Documents/Docker-MySQL-Replication2$ docker-compose up
```

Each container will initiate and you will see the console output from each of them. The MariaDB image is configured to launch a script at the root of the file system called docker-entrypoint.sh. You can see this by running docker-compose ps, or running docker inspect on the container name.

```
mark@mark-ThinkPad-W530:~/Documents/Docker-MySQL-Replication2$ docker-compose ps
              Name                            Command               State    Ports
-------------------------------------------------------------------------------------
dockermysqlreplication2_master_1   /docker-entrypoint.sh mysqld   Up      3306/tcp
dockermysqlreplication2_slave_1    /docker-entrypoint.sh mysqld   Up      3306/tcp
```

You can also see the ports that are configured to be exposed for MariaDB. This is configured in the MariaDB image, so we don't need to expose the ports manually with the yml file. Notice the docker-entrypoint.sh script. This script simply ensures the required environment variables are set and starts up MariaDB. Now it's time to run some commands in order to get replication to work. Login to the master container in a separate shell with the following command.

```
mark@mark-ThinkPad-W530:~/Documents/Docker-MySQL-Replication2$ docker-compose ps
              Name                            Command               State    Ports
-------------------------------------------------------------------------------------
dockermysqlreplication2_master_1   /docker-entrypoint.sh mysqld   Up      3306/tcp
dockermysqlreplication2_slave_1    /docker-entrypoint.sh mysqld   Up      3306/tcp
mark@mark-ThinkPad-W530:~/Documents/Docker-MySQL-Replication2$ docker exec -it dockermysqlreplication2_master_1 /bin/bash
root@733407f35219:/#
```

You now have a root prompt. Login to MariaDB.

```
root@733407f35219:/# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.1.13-MariaDB-1~jessie mariadb.org binary distribution

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Use the password you set as the root password for mysql in the environment variables. Run the following two commands to insert the sakila database into master.

- SOURCE /var/lib/mysql/sakila-schema.sql;
- SOURCE /var/lib/mysql/sakila-data.sql;

Then use these commands to enable replication on the master server.

- STOP SLAVE;
- GRANT REPLICATION SLAVE ON *.* TO ‘sk’@’%’ IDENTIFIED BY ‘ubuntu’;
- FLUSH PRIVILEGES;
- FLUSH TABLES WITH READ LOCK;
- UNLOCK TABLES;
- SHOW MASTER STATUS;

```
MariaDB [sakila]> SHOW MASTER STATUS;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| mysqld-bin.000005 |    23687 | sakila       |                  |
+--------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

This information will be useful later for the slave, so take note of it. Let’s export everything into a master database file that we will copy over to the slave. Use the following command to do this:

- mysqldump --all-databases --user=root --password --master-data > masterdatabase.sql

We are now going to use a nifty Docker command to copy the masterdatabase.sql file from the master container to the local file system.

```
docker cp dockermysqlreplication2_master_1:/masterdatabase.sql /tmp/masterdatabase.sql
docker cp /tmp/masterdatabase.sql dockermysqlreplication2_slave_1:/
```

Copying files between containers is currently not supported, so I simply copied the file to a temp folder, and from that temp folder to the slave container. Login to the slave container in another shell prompt using the same method from earlier.

```
mark@mark-ThinkPad-W530:~/Documents/Docker-MySQL-Replication2$ docker exec -it dockermysqlreplication2_slave_1 /bin/bash
root@1621dd1de168:/#
```

Import the master database.

```
root@1621dd1de168:/# mysql -u root -p < masterdatabase.sql
Enter password:
root@1621dd1de168:/#
```

Sweet! We are almost there. We need to know the IP address of the master container. This can be found via ‘ip addr’.

Take note of the IP address in the eth0 interface. Login to MariaDB on the slave container and run the following commands:

- STOP SLAVE;
- CHANGE MASTER TO MASTER_HOST='172.17.0.2', MASTER_USER='root', MASTER_PASSWORD=’1dumbpassword’, MASTER_LOG_FILE='mysqld-bin.000005', MASTER_LOG_POS=23687;

Be sure to change the IP address, the username, the password, the log file name, and the log file position to what they should be in your environment.

- SLAVE START;
- SHOW SLAVE STATUS\G;

This is what you are looking for:

```
MariaDB [(none)]> SHOW SLAVE STATUS\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 172.17.0.2
                  Master_User: root
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysqld-bin.000005
          Read_Master_Log_Pos: 23687
               Relay_Log_File: mysqld-relay-bin.000002
                Relay_Log_Pos: 538
        Relay_Master_Log_File: mysqld-bin.000005
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB: sakila
```

We now know the slave is connected and waiting for the master to send data that the database has changed. Let’s test it out. Pull up the console with the master container and use the following commands:

- Use sakila - Tells MariaDB what database we are using.
- SHOW TABLES; - Displays all table in the database.

```
mark@mark-ThinkPad-W530: ~/Documents/Docker-MySQL-Replication2
MariaDB [(none)]> use sakila
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [sakila]> SHOW TABLES;
+----------------------------+
| Tables_in_sakila           |
+----------------------------+
| actor                      |
| actor_info                 |
| address                    |
| category                   |
| city                       |
| country                    |
| customer                   |
| customer_list              |
| film                       |
| film_actor                 |
| film_category              |
| film_list                  |
| film_text                  |
| inventory                  |
| language                   |
| nicer_but_slower_film_list |
| payment                    |
| rental                     |
| sales_by_film_category     |
| sales_by_store             |
| staff                      |
| staff_list                 |
| store                      |
+----------------------------+
23 rows in set (0.00 sec)

MariaDB [sakila]>
```

Issue the following commands to create a new table with some data:

- Create table sample (c int);
- Insert into sample (c) values (1);
- Select * from sample;

```
MariaDB [sakila]> create table sample (c int);
Query OK, 0 rows affected (0.03 sec)

MariaDB [sakila]> insert into sample (c) values (1);
Query OK, 1 row affected (0.01 sec)

MariaDB [sakila]> select * from sample;
+------+
| c    |
+------+
|    1 |
+------+
1 row in set (0.00 sec)
```

We now have a new table with some really trivial sample data. Let’s check what tables the slave has.

```
MariaDB [(none)]> use sakila;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [sakila]> SHOW TABLES;
+----------------------------+
| Tables_in_sakila           |
+----------------------------+
| actor                      |
| actor_info                 |
| address                    |
| category                   |
| city                       |
| country                    |
| customer                   |
| customer_list              |
| film                       |
| film_actor                 |
| film_category              |
| film_list                  |
| film_text                  |
| inventory                  |
| language                   |
| nicer_but_slower_film_list |
| payment                    |
| rental                     |
| sales_by_film_category     |
| sales_by_store             |
| sample                     |
| staff                      |
| staff_list                 |
| store                      |
+----------------------------+
24 rows in set (0.00 sec)

MariaDB [sakila]> select * from sample;
+------+
| c    |
+------+
|    1 |
+------+
1 row in set (0.00 sec)
```

We have success. Database replication is fully working between two Docker containers! This process took me some trial and error, so here's some tidbits that may help you in case you get stuck.

FAQ:

1. Whenever I use a command, I receive a prompt that says TERM environment variable not set. What is this?
    a. I have gotten this error a few times when working with Linux. I'm not exactly sure what it is. Using 'echo $TERM' I can see it's set to the value 'dumb', even though the system just yelled at me for an unset variable. Use the command "export TERM=dumb" and that should solve your problem.
2. My volume mounts aren't working. How do I troubleshoot this?
    a. Make sure you're using an absolute path for the file system in the Docker container. Also double check for spelling. If you're using a relative path for your local file system, ensure you have a period (.) in front of your path. It's never a bad idea to remove the old images docker-compose creates and create them again. Use docker-compose rm to remove stopped containers.
3. How do I stop Docker containers?
    a. To stop docker containers running in docker-compose, simply press Ctrl+C and the containers will close. To close containers you have started with 'docker run', use 'docker stop container_name'.

References:

It's important to reference work that has considerable contributions from others. Here are the links I used to help create this project:

- http://www.unixmen.com/setup-mariadb-master-slave-replication-in-centos-7/
    - SK from UnixMen.com
- https://dev.mysql.com/doc/sakila/en/sakila-installation.html
    - Sakila sample database from Mike Hillyer at dev.mysql.com
- https://docs.docker.com/compose/compose-file/
    - Docker compose documentation
- https://hub.docker.com/_/mariadb/
    - MariaDB Docker hub image & documentation