MYSQL Database Replication Setup

Why replicate ? You can back up your MySQL database using the built-in dump tool, but the second you do that the backup is out of date. If your database is large then while the backup is taking place all of it's tables are locked, stopping any service from reading or writing to that database. In a 24/7 worldwide service this backup time will mean downtime for someone, somewhere in the world.

A simple solution is database replication, which keeps a real-time copy of the database on a remote server. With this in place, if something happens to the primary database, it will be much easier to get your database back up and running with current information. Backups can take place on the replica without downtime of the main database and once the backup has completed the replica will catch up with any changes that occurred on the Master during it's absence automatically.

This is what you'll need to set up MySQL database replication:

- At least Two correctly configured MySQL servers
- Root access and access to the database administrator on both servers

The setup will use two machines: the Master and the Slave. We'll start with the setup of the Master.

Before I start this guide I would like to bring up the fact that every OS has a different place for the main config file for MySQL that you'll need to edit in this guide.

In Ubuntu it's located in `/etc/mysql/my.cnf`, in Centos/RHEL it's in `/etc/my.cnf` and on OS X it does not exist and one must be copied to `/etc/my.cnf` from the MySQL install directory.

Configure the Master

Stop any process that may update the database your going to replicate. This is required as we will need a fixed point in time backup so that we can clone the database to the Slaves. For example if this is the JAMF JSS Server database stop tomcat7.

The next step is to enable MySQL for networking. Not all OS's need to do this. Open the file /etc/mysql/my.conf, look for the following lines, and uncomment them if they exist:

```
#skip-networking
#bind-address = 127.0.0.1
```

If the lines do not exist, you many need to add them (minus the "#").

Within the same file, we have to make MySQL aware of the database that will be replicated. You can do this with these lines:

```
log-bin = /var/log/mysql/mysql-bin.log
binlog-do-db=DATABASE_TO_BE_REPLICATED
server-id=1
```

`DATABASE_TO_BE_REPLICATED` is the actual name of the database you want to replicate.

The above lines tell MySQL the following:

Line 1: Configures the log file to be used. This uses the Binary Log System and as such these will get large. Look at `PURGE BINARY LOGS` later in this document.

Line 2: Configures the database to be logged and as such replicated. Without this ALL databases on the server will have all of there actions logged and result in MASSIVE log files and data that must NEVER be replicated being sent to the Slave server, such as Schema database entries. If this is not set you can use `replicate-do-db` on the Slave server to specify which Database to replicate. This is ok but as the Binary Log is logging ALL actions those logs will grow quickly.

Line 3: Configures the machine with a unique ID. Each MySQL server needs a unique ID. Zero is not a valid ID

After you add these lines, save the my.cnf file and restart MySQL. On Centos/RHEL and Ubuntu you can use the command `/etc/inid.t/mysqld restart` or you can restart the MySQL daemon is using the service command. Centos/RHEL is `service mysqld restart`. Ubuntu is `service mysql restart`

OS X is a bit different, as always. 10.7 and 10.8 server don't come with MySQL in the OS and it needs to be installed from the MySQL distribution. This comes with a Startup Item to auto start the MySQL process. This is a very outdated way of starting processes in OS X.

With this configuration complete, it's time to set up a user for replication privileges. Log on to the Master and from the command line issue the command `mysql -u root -p`

Enter the password for the MySQL admin password. Upon successful authentication, you will be at the MySQL prompt, where you will enter the command `GRANT REPLICATION SLAVE ON *.* TO 'USER'@'%' IDENTIFIED BY 'PASSWORD';`

`USER` is the user who will have replication privileges, and the `PASSWORD` is that user's MySQL password. The `USER` will be made if not present.

With that command issued, you must flush the database privileges with the command `FLUSH PRIVILEGES;`

Make sure MySQL can see the Master with the command `SHOW MASTER STATUS;`

This command should list information (including the all-important position number, which will be required for the Slave setup) for the database to be replicated. You need to write down that information, which will look something like this:

```
File: mysql-bin.00002
Position: 230
Binlog_Do_DB: database_to_be_replicated
Binlog_Ignore_DB:
1 row in set (0.00 sec)
```

The next step is to retrieve the tables and data from the database to be replicated; in order to do this, the database must be temporarily locked. When the database is locked, it cannot be used, so do this at a time when the database won't be needed.

To lock the database, issue the command `FLUSH TABLES WITH READ LOCK;`

You must dump the database that will be copied to the Slave. There used to be a command `LOAD DATA FROM MASTER`, but that command has been deprecated as it could kill a server if the database was massive. So, to get the data you can do it one of two ways.

Using MySQLDump

The First way is to do a dump with the command `mysqldump -u root -p DATABASE_TO_BE_REPLICATED > DATABASE_TO_BE_REPLICATED.sql` where `DATABASE_TO_BE_REPLICATED` is the name of the actual database

being replicated. You'll be prompted for the MySQL admin password; enter that password, and the dump will process.

Now unlock the tables with the command `UNLOCK TABLES;` and then quit from MySQL.

Copy that database dump file to the Slave and then restore the database (on the SLAVE) with the command `mysql -u root -p DATABASE_TO_BE_REPLICATED < DATABASE_TO_BE_REPLICATED.sql` where `DATABASE_TO_BE_REPLICATED` is the name of the database being replicated.

Using Raw Data Files

To create a raw data snapshot of **`MyISAM`** tables you can use standard copy tools such as **`cp`** or **`copy`**, a remote copy tool such as **`scp`** or **`rsync`**, an archiving tool such as **`zip`** or **`tar`**, or a file system snapshot tool such as **`dump`**, providing that your MySQL data files exist on a single file system. If you are replicating only certain databases then make sure you copy only those files that related to those tables. (For **`InnoDB`**, all tables in all databases are stored in the shared tablespace files, unless you have the innodb_file_per_table option enabled.)
You may want to specifically exclude the following files from your archive:

- Files relating to the **`mysql`** database.
- The master.info file.
- The master's binary log files.
- Any relay log files.

To get the most consistent results with a raw data snapshot you should shut down the master server during the process, as follows:

1. Note the Masters Details with `SHOW MASTER STATUS;`
2. Lock the databases using `FLUSH TABLES WITH READ LOCK;`
3. Quit MySQL and Shut down the master server with `mysqladmin shutdown` at the command line
4. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:
   ```
   tar cf /tmp/db.tar ./data
   zip -r /tmp/db.zip ./data
   rsync --recursive ./data /tmp/dbdata
   ```
5. Restart the master server.

If you are not using **InnoDB** tables, you can get a snapshot of the system from a master without shutting down the server as described in the following steps:

1. Note the Masters Details with `SHOW MASTER STATUS;`
2. Lock the databases using `FLUSH TABLES WITH READ LOCK;`
3. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:
   ```
   tar cf /tmp/db.tar ./data
   zip -r /tmp/db.zip ./data
   rsync --recursive ./data /tmp/dbdata
   ```
4. In the client where you acquired the read lock, release the lock using `UNLOCK TABLES;`

Once you have created the archive or copy of the database, you will need to copy the files to each slave before starting the slave replication process.

Configure the Slave(s)

In order to configure the Slave, open the my.cnf file, add the following, and save and close that file:

```
server-id=2
master-host=IP_ADDRESS_OF_MASTER
master-user=USER
master-password=USER_PASSWORD
master-connect-retry=60
replicate-do-db=DATABASE
```

Where:

- `server-id` is a Unique id number for this instance of MySQL in your replication setup
- `IP_ADDRESS_OF_MASTER` is the actual IP address of the Master server.
- `USER` is the database administrator with replication privileges.
- `USER_PASSWORD` is the password associated with the `USER`.
- `DATABASE` is the name to be replicated.

Depending on your MySQL version, adding the `master-` and the `replicate-do-db` configuration to the `my.cnf` may result in MySQL failing to start. If you find your having this problem remove those lines and just leave the `server-id` in the `my.cnf` file. The Slave configuration will be setup in the master.info file by running the `CHANGE MASTER TO` command in the next step.

NOTE: MySQL 5.6 includes a couple new options that allow you to store replication master and relay information in tables instead of in the respective master.info and relay-log.info files that have been used historically. This appears to be partly under the guise of increased security, particularly in the case of the master info "repository", if the "Note" we get from CHANGE MASTER TO is to be believed. However, it's important to note that using --master-info-repository=TABLE really offers no security benefit of any kind.

This next step requires information returned from the `SHOW MASTER STATUS;` command that was run on the Master. Stop the Slave by issuing these commands:

```
mysql -u root -p
SLAVE STOP;
```

Then run the following command:

```
CHANGE MASTER TO MASTER_HOST='IP_ADDRESS_OF_MASTER',
MASTER_USER='USER', MASTER_PASSWORD='USER_PASSWORD',
MASTER_LOG_FILE='mysql-bin.007', MASTER_LOG_POS=NUMBER;
```

Where the following applies:

- `IP_ADDRESS_OF_MASTER` is the actual IP address of the Master
- `USER` is the MySQL replication `USER` made earlier
- `USER_PASSWORD` is the password for the replication `USER`
- `NUMBER` is the Position Number reported from the `SHOW MASTER STATUS` command.

Restart the Slave by issuing the command `SLAVE START;` and then exit the MySQL prompt with the command `quit`.

You can follow these steps to make sure everything is working:
Issue the command on the slave:

```
mysql -u root -p
SHOW SLAVE STATUS\G
```

You should see something contaning this:

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

If the status of both the above are Yes, you now have a working, replicated MySQL database. Repeat for each slave you wish to setup.

Admin Commands for Replication

You can see details regarding a slave by login into MySQL on a slave and issuing :

```
SHOW SLAVE STATUS\G
```

This will display all the status you should need. Note the \G at the end of the command. This is not a typo buy forces a carriage return instead of a semi colon so make the output readable. Try with the normal semi colon to see the difference!

To pause replication, stop or restart a Slave processing the Masters binary log you can login into MySQL on a slave and issue:

```
STOP SLAVE;
START SLAVE;
```

To stop a Slave being a Slave and to make it a standalone login into MySQL on a slave and use:

```
RESET SLAVE;
```

To see the Master servers status login into MySQL on the Master and issue:

```
SHOW MASTER STATUS;
```

Log Files

Replication uses the Binary Log system in MySQL. The binary log is a set of files that contain information about data modifications made by the MySQL server. The log consists of a set of binary log files, plus an index file. These logs can get very large and may need manual purging from time to time, as it is not automatically done by default. This can be setup as detailed later on.

The `PURGE BINARY LOGS` statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. BINARY and MASTER are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

```
PURGE BINARY LOGS TO 'mysql-bin.010';
PURGE BINARY LOGS BEFORE '2008-04-02 22:46:26';
```

The `BEFORE` variant's argument should evaluate to a `DATETIME` value (a value in `'YYYY-MM-DD hh:mm:ss'` format).
This statement is safe to run while slaves are replicating. You need not stop them. If you have an active slave that currently is reading one of the log files you are trying to delete, this statement does nothing and fails with an error. However, if a slave is not connected and you happen to purge one of the log files it has yet to read, the slave will be

unable to replicate after it reconnects.

To safely purge binary log files, follow this procedure:

- On each slave server, use `SHOW SLAVE STATUS`; to check which log file it is reading.
- Obtain a listing of the binary log files on the master server with `SHOW BINARY LOGS`;
- Determine the earliest log file among all the slaves. This is the target file. If all the slaves are up to date, this is the last log file on the list.
- Make a backup of all the log files you are about to delete. (This step is optional, but always advisable.)
- Purge all log files up to but not including the target file.

You are advised to set the `expire_logs_days` system variable to expire binary log files automatically after a given number of days. You should set the variable no lower than the maximum number of days your slaves might lag behind the master. The default is 0, which means "no automatic removal". You can also set a maximum file size for the log files and they will be rotated when the file reaches that size. The Default is 1GB.

It's best to add this to the my.cnf config file by adding the line:

```
expire_logs_days = 14
max_binlog_size = 524288000
```

You can check this by using:

```
show variables like "expire%";
```

To delete all binary logs older than 7 days:

```
PURGE BINARY LOGS BEFORE DATE_SUB( NOW( ), INTERVAL 7
DAY);
```

To automate the log purging, setup a cron job or a LaunchD daemon to run the following

```
mysql -uroot -ppasswd -e "PURGE BINARY LOGS BEFORE
DATE_SUB( NOW( ), INTERVAL 7 DAY);"
```