By Falko Timme
Published: 2008-11-18 18:09

Setting Up Master-Master Replication On Four Nodes With MySQL 5 On Debian Etch

Version 1.0
Author: Falko Timme <ft [at] falkotimme [dot] com>
Last edited 11/14/2008

This tutorial explains how you can set up MySQL master-master replication on four MySQL nodes (running on Debian Etch). The difference to a two node master-master replication (which is explained **here**) is that if you have more than two nodes, the replication goes in a circle, i.e., with four nodes, the replication goes from node1 to node2, from node2 to node3, from node3 to node4, and from node4 to node1.

Since version 5, MySQL comes with built-in support for master-master replication, solving the problem that can happen with self-generated keys. In former MySQL versions, the problem with master-master replication was that conflicts arose immediately if node A and node B both inserted an auto-incrementing key on the same table. The advantages of master-master replication over the traditional master-slave replication are that you don't have to modify your applications to make write accesses only to the master, and that it is easier to provide high-availability because if the master fails, you still have the other master.

I do not issue any guarantee that this will work for you!

1 Preliminary Note

In this tutorial I will show how to replicate the database *exampledb* on four MySQL nodes:

- *server1.example.com*: IP address *192.168.0.100*
- *server2.example.com*: IP address *192.168.0.101*
- *server3.example.com*: IP address *192.168.0.102*
- *server4.example.com*: IP address *192.168.0.103*

Each node is a master and a slave at the same time. All four systems are running Debian Etch; however, the configuration should apply to almost all distributions with little or no modifications.

Replication will work in a circle, i.e., the replication goes from *server1* to *server2*, from *server2* to *server3*, from *server3* to *server4*, and from *server4* back to *server1*:

... --> server1 --> server2 --> server3 --> server4 --> server1 --> ...

2 Installing MySQL 5.0

If MySQL 5.0 isn't already installed on *server1* to *server4*, install it now:

server1/server2/server3/server4:

```
apt-get install mysql-server-5.0 mysql-client-5.0
```

To make sure that the replication can work, we must make MySQL listen on all interfaces, therefore we comment out the line *bind-address = 127.0.0.1* in */etc/mysql/my.cnf*:

server1/server2/server3/server4:

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

```
[...]
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
#bind-address           = 127.0.0.1
[...]
```

Restart MySQL afterwards:

server1/server2/server3/server4:

```
/etc/init.d/mysql restart
```

Then check with

server1/server2/server3/server4:

```
netstat -tap | grep mysql
```

that MySQL is really listening on all interfaces:

```
server1:~# netstat -tap | grep mysql
tcp        0      0 *:mysql                 *:*                     LISTEN     2671/mysqld
server1:~#
```

Afterwards, set a MySQL password for the user `root@localhost`:

server1/server2/server3/server4:

```
mysqladmin -u root password yourrootsqlpassword
```

Next we create MySQL passwords for `root@server1.example.com`, `root@server2.example.com`, `root@server3.example.com`, and `root@server4.example.com`:

server1:

```
mysqladmin -h server1.example.com -u root password yourrootsqlpassword
```

server2:

```
mysqladmin -h server2.example.com -u root password yourrootsqlpassword
```

server3:

```
mysqladmin -h server3.example.com -u root password yourrootsqlpassword
```

server4:

```
mysqladmin -h server4.example.com -u root password yourrootsqlpassword
```

Now we set up a replication user `slaveuser_for_s2` that can be used by `server2` to access the MySQL database on `server1`:

server1:

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

On the MySQL shell, run the following commands:

```
GRANT REPLICATION SLAVE ON *.* TO 'slaveuser_for_s2'@'%' IDENTIFIED BY 'slave_user_for_server2_password';

FLUSH PRIVILEGES;

quit;
```

Then we set up a replication user `slaveuser_for_s3` that can be used by `server3` to access the MySQL database on `server2`...

server2:

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

```
GRANT REPLICATION SLAVE ON *.* TO 'slaveuser_for_s3'@'%' IDENTIFIED BY 'slave_user_for_server3_password';

FLUSH PRIVILEGES;

quit;
```

... and a replication user `slaveuser_for_s4` that can be used by `server4` to access the MySQL database on `server3`...

server3:

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

```
GRANT REPLICATION SLAVE ON *.* TO 'slaveuser_for_s4'@'%' IDENTIFIED BY 'slave_user_for_server4_password';

FLUSH PRIVILEGES;

quit;
```

... and finally a replication user `slaveuser_for_s1` that can be used by `server1` to access the MySQL database on `server4`:

server4:

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

```
GRANT REPLICATION SLAVE ON *.* TO 'slaveuser_for_s1'@'%' IDENTIFIED BY 'slave_user_for_server1_password';

FLUSH PRIVILEGES;

quit;
```

3 Some Notes

In the following I will assume that the database `exampledb` is **already existing** on `server1`, and that there are tables with records in it.

Before we start setting up the replication, we create an **empty** database `exampledb` on `server2`, `server3`, and `server4`:

server2/server3/server4:

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

```
CREATE DATABASE exampledb;

quit;
```

4 Setting Up Replication

Now we set up master-master replication in `/etc/mysql/my.cnf`. The crucial configuration options for master-master replication are `auto_increment_increment` and `auto_increment_offset`:

- `auto_increment_increment` controls the increment between successive AUTO_INCREMENT values.
- `auto_increment_offset` determines the starting point for AUTO_INCREMENT column values.

Let's assume we have N MySQL nodes (N=4 in this example), then `auto_increment_increment` has the value N on all nodes, and each node must have a different value for `auto_increment_offset` (1, 2, ..., N).

We also need to configure `log-slave-updates` because otherwise replication will work only, for example, from `server1` to `server2`, but not to `server3` and `server4`.

Now let's configure our four MySQL nodes:

server1:

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

Search for the section that starts with `[mysqld]`, and put the following options into it (commenting out all existing **conflicting** options):

```
[...]
[mysqld]
server-id = 1
replicate-same-server-id = 0
auto-increment-increment = 4
auto-increment-offset = 1

master-host = 192.168.0.103
master-user = slaveuser_for_s1
master-password = slave_user_for_server1_password
master-connect-retry = 60
replicate-do-db = exampledb

log-bin = /var/log/mysql/mysql-bin.log
binlog-do-db = exampledb
log-slave-updates

relay-log = /var/lib/mysql/slave-relay.log
relay-log-index = /var/lib/mysql/slave-relay-log.index
```

```
expire_logs_days        = 10
max_binlog_size         = 500M
[...]
```

Then restart MySQL:

```
/etc/init.d/mysql restart
```

Now do the same on `server2`...

server2:

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

```
[...]
[mysqld]
server-id = 2
replicate-same-server-id = 0
auto-increment-increment = 4
auto-increment-offset = 2

master-host = 192.168.0.100
master-user = slaveuser_for_s2
master-password = slave_user_for_server2_password
master-connect-retry = 60
replicate-do-db = exampledb

log-bin= /var/log/mysql/mysql-bin.log
binlog-do-db = exampledb
```

```
log-slave-updates

relay-log = /var/lib/mysql/slave-relay.log
relay-log-index = /var/lib/mysql/slave-relay-log.index

expire_logs_days        = 10
max_binlog_size         = 500M
[...]
```

```
/etc/init.d/mysql restart
```

...server3...

server3:

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

```
[...]
[mysqld]
server-id = 3
replicate-same-server-id = 0
auto-increment-increment = 4
auto-increment-offset = 3

master-host = 192.168.0.101
master-user = slaveuser_for_s3
master-password = slave_user_for_server3_password
master-connect-retry = 60
replicate-do-db = exampledb
```

```
log-bin= /var/log/mysql/mysql-bin.log
binlog-do-db = exampledb
log-slave-updates

relay-log = /var/lib/mysql/slave-relay.log
relay-log-index = /var/lib/mysql/slave-relay-log.index

expire_logs_days        = 10
max_binlog_size         = 500M
[...]
```

```
/etc/init.d/mysql restart
```

... and `server4`:

server4:

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

```
[...]
[mysqld]
server-id = 4
replicate-same-server-id = 0
auto-increment-increment = 4
auto-increment-offset = 4

master-host = 192.168.0.102
master-user = slaveuser_for_s4
```

```
master-password = slave_user_for_server4_password
master-connect-retry = 60
replicate-do-db = exampledb

log-bin= /var/log/mysql/mysql-bin.log
binlog-do-db = exampledb
log-slave-updates

relay-log = /var/lib/mysql/slave-relay.log
relay-log-index = /var/lib/mysql/slave-relay-log.index

expire_logs_days        = 10
max_binlog_size         = 500M
[...]
```

```
/etc/init.d/mysql restart
```

Before we continue, we must make sure that no slave processes are running on `server1` to `server4`:

server1/server2/server3/server4:

```
/usr/bin/mysqladmin --user=root --password=yourrootsqlpassword stop-slave
```

Next we lock the `exampledb` database on `server1`, find out about the master status of `server1`, create an SQL dump of `exampledb` (that we will import into `exampledb` on `server2`, `server3`, and `server4` so that all four databases contain the same data), and unlock the database so that it can be used again:

server1:

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

On the MySQL shell, run the following commands:

server1:

```
USE exampledb;

FLUSH TABLES WITH READ LOCK;

SHOW MASTER STATUS;
```

The last command should show something like this (please write it down, we'll need it later on):

```
mysql> SHOW MASTER STATUS;
+------------------+----------+-------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+-------------+------------------+
| mysql-bin.000010 |       98 | exampledb    |                  |
+------------------+----------+-------------+------------------+
1 row in set (0.00 sec)

mysql>
```

Now don't leave the MySQL shell, because if you leave it, the database lock will be removed, and this is not what we want right now because we must create a database dump now. While the MySQL shell is still open, we open a **second** command line window where we create the SQL dump `snapshot.sql` and transfer it to `server2`, `server3`, and `server4` (using scp):

server1:

```
cd /tmp

mysqldump -u root -pyourrootsqlpassword --opt exampledb > snapshot.sql
```

```
scp snapshot.sql root@192.168.0.101:/tmp
```

```
scp snapshot.sql root@192.168.0.102:/tmp
```

```
scp snapshot.sql root@192.168.0.103:/tmp
```

Afterwards, you can close the second command line window. On the first command line window, we can now unlock the database and leave the MySQL shell:

server1:

```
UNLOCK TABLES;

quit;
```

4.1 Setting Up Replication On server2

(This chapter is for `server2` only!)

On `server2`, we can now import the SQL dump `snapshot.sql` like this:

server2:

```
/usr/bin/mysqladmin --user=root --password=yourrootsqlpassword stop-slave

cd /tmp

mysql -u root -pyourrootsqlpassword exampledb < snapshot.sql
```

Afterwards, we must find out about the master status of `server2` as well and write it down:

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

```
USE exampledb;

FLUSH TABLES WITH READ LOCK;
```

```
SHOW MASTER STATUS;
```

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000010 |     1067 | exampledb    |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)

mysql>
```

Then unlock the tables:

```
UNLOCK TABLES;
```

and run the following command to make `server2` a slave of `server1` (it is important that you replace the values in the following command with the values you got from the SHOW MASTER STATUS; command that we ran on server1!):

```
CHANGE MASTER TO MASTER_HOST='192.168.0.100', MASTER_USER='slaveuser_for_s2', MASTER_PASSWORD='slave_user_for_server2_password', MASTER_LOG_FILE='mysql-bin.000010', MASTER_LOG_POS=98;
```

Finally start the slave:

```
START SLAVE;
```

Then check the slave status:

```
SHOW SLAVE STATUS \G
```

It is important that both `Slave_IO_Running` and `Slave_SQL_Running` have the value `Yes` in the output (otherwise something went wrong, and you should check your setup again and take a look at `/var/log/syslog` to find out about any errors):

```
mysql> SHOW SLAVE STATUS G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.0.100
                Master_User: slaveuser_for_s2
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000010
        Read_Master_Log_Pos: 98
             Relay_Log_File: slave-relay.000002
              Relay_Log_Pos: 235
      Relay_Master_Log_File: mysql-bin.000010
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB: exampledb
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
```

```
                 Last_Errno: 0
                 Last_Error:
               Skip_Counter: 0
        Exec_Master_Log_Pos: 98
            Relay_Log_Space: 235
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 0
1 row in set (0.01 sec)

mysql>
```

Afterwards, you can leave the MySQL shell on *server2*:

```
quit
```

Now the replication from *server1* to *server2* is set up. Next we must configure replication from *server2* to *server3*.

4.2 Setting Up Replication On server3

(This chapter is for *server3* only!)

On *server3*, we can now import the SQL dump *snapshot.sql* like this:

server3:

```
/usr/bin/mysqladmin --user=root --password=yourrootsqlpassword stop-slave

cd /tmp

mysql -u root -pyourrootsqlpassword exampledb < snapshot.sql
```

Afterwards, we must find out about the master status of `server3` as well and write it down:

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

```
USE exampledb;

FLUSH TABLES WITH READ LOCK;
```

```
SHOW MASTER STATUS;
```

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000010 |     1067 | exampledb    |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)

mysql>
```

Then unlock the tables:

```
UNLOCK TABLES;
```

and run the following command to make `server3` a slave of `server2` (it is important that you replace the values in the following command with the values you got from the SHOW MASTER STATUS; command that we ran on server2!):

```
CHANGE MASTER TO MASTER_HOST='192.168.0.101', MASTER_USER='slaveuser_for_s3', MASTER_PASSWORD='slave_user_for_server3_password', MASTER_LOG_FILE='mysql-bin.000010', MASTER_LOG_POS=1067;
```

Finally start the slave:

```
START SLAVE;
```

Then check the slave status:

```
SHOW SLAVE STATUS \G
```

It is important that both `Slave_IO_Running` and `Slave_SQL_Running` have the value `Yes` in the output (otherwise something went wrong, and you should check your setup again and take a look at `/var/log/syslog` to find out about any errors):

```
mysql> SHOW SLAVE STATUS G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.0.101
                Master_User: slaveuser_for_s3
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000010
        Read_Master_Log_Pos: 1067
             Relay_Log_File: slave-relay.000002
              Relay_Log_Pos: 235
      Relay_Master_Log_File: mysql-bin.000010
           Slave_IO_Running: Yes
```

```
          Slave_SQL_Running: Yes
            Replicate_Do_DB: exampledb
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                 Last_Errno: 0
                 Last_Error:
               Skip_Counter: 0
        Exec_Master_Log_Pos: 1067
            Relay_Log_Space: 235
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 0
1 row in set (0.00 sec)

mysql>
```

Afterwards, you can leave the MySQL shell on `server3`:

```
quit
```

Now the replication from `server2` to `server3` is set up. Next we must configure replication from `server3` to `server4`.

4.3 Setting Up Replication On server4

(This chapter is for *server4* only!)

On *server4*, we can now import the SQL dump *snapshot.sql* like this:

server4:

```
/usr/bin/mysqladmin --user=root --password=yourrootsqlpassword stop-slave

cd /tmp

mysql -u root -pyourrootsqlpassword exampledb < snapshot.sql
```

Afterwards, we must find out about the master status of *server4* as well and write it down:

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

```
USE exampledb;

FLUSH TABLES WITH READ LOCK;
```

```
SHOW MASTER STATUS;
```

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000010 |     1067 | exampledb    |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)

mysql>
```

Then unlock the tables:

```
UNLOCK TABLES;
```

and run the following command to make `server4` a slave of `server3` (it is important that you replace the values in the following command with the values you got from the SHOW MASTER STATUS; command that we ran on server3!):

```
CHANGE MASTER TO MASTER_HOST='192.168.0.102', MASTER_USER='slaveuser_for_s4', MASTER_PASSWORD='slave_user_for_server4_password', MASTER_LOG_FILE='mysql-bin.000010', MASTER_LOG_POS=1067;
```

Finally start the slave:

```
START SLAVE;
```

Then check the slave status:

```
SHOW SLAVE STATUS \G
```

It is important that both `Slave_IO_Running` and `Slave_SQL_Running` have the value `Yes` in the output (otherwise something went wrong, and you should check your setup again and take a look at `/var/log/syslog` to find out about any errors):

```
mysql> SHOW SLAVE STATUS G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
```

```
                Master_Host: 192.168.0.102
                Master_User: slaveuser_for_s4
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000010
        Read_Master_Log_Pos: 1067
             Relay_Log_File: slave-relay.000002
              Relay_Log_Pos: 235
      Relay_Master_Log_File: mysql-bin.000010
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB: exampledb
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                 Last_Errno: 0
                 Last_Error:
               Skip_Counter: 0
        Exec_Master_Log_Pos: 1067
            Relay_Log_Space: 235
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 0
1 row in set (0.00 sec)
```

```
mysql>
```

Afterwards, you can leave the MySQL shell on `server3`:

```
quit
```

Now the replication from `server3` to `server4` is set up. Finally we must configure replication from `server4` to `server1 to close the replication circle.`

4.4 Setting Up Replication On server1

(This chapter is for `server1` only!)

To do this, we stop the slave on `server1` and make it a slave of `server4`:

server1:

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

```
STOP SLAVE;
```

Make sure that you use the values of the SHOW MASTER STATUS; command that you ran on server4 in the following command:

```
CHANGE MASTER TO MASTER_HOST='192.168.0.103', MASTER_USER='slaveuser_for_s1', MASTER_PASSWORD='slave_user_for_server1_password',
MASTER_LOG_FILE='mysql-bin.000010', MASTER_LOG_POS=1067;
```

Then start the slave on `server1`:

```
START SLAVE;
```

Then check the slave status:

```
SHOW SLAVE STATUS \G
```

It is important that both `Slave_IO_Running` and `Slave_SQL_Running` have the value `Yes` in the output (otherwise something went wrong, and you should check your setup again and take a look at `/var/log/syslog` to find out about any errors):

```
mysql> SHOW SLAVE STATUS G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.0.103
                Master_User: slaveuser_for_s1
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000010
        Read_Master_Log_Pos: 1067
             Relay_Log_File: slave-relay.000002
              Relay_Log_Pos: 235
      Relay_Master_Log_File: mysql-bin.000010
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB: exampledb
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                 Last_Errno: 0
                 Last_Error:
               Skip_Counter: 0
        Exec_Master_Log_Pos: 1067
            Relay_Log_Space: 235
            Until_Condition: None
             Until_Log_File:
              Until_Log_Pos: 0
         Master_SSL_Allowed: No
         Master_SSL_CA_File:
         Master_SSL_CA_Path:
            Master_SSL_Cert:
          Master_SSL_Cipher:
             Master_SSL_Key:
      Seconds_Behind_Master: 0
1 row in set (0.00 sec)

mysql>
```

Afterwards you can leave the MySQL shell:

```
quit
```

If nothing went wrong, MySQL master-master replication should now be working. If it isn't, please check `/var/log/syslog` for MySQL errors.

5 Links

- MySQL: **http://www.mysql.com**
- Debian: **http://www.debian.org**