

\*\*\*\*\*

# *MySQL Cluster with Failover*

\*\*\*\*\*

- 1) master-slave replication*
- 2) ssh access as root from and to the mha manager, mysql-master and mysql-slaves*
- 3) mha ( Master High Availability ) configuration*

\*\*\*\*\*

## *1) Master Slave Replication in MySQL*

\*\*\*\*\*

*192.168.1.234 :: MHA Manager*  
*192.168.1.45 :: Master Database*  
*192.168.1.47 :: Slave*  
*192.168.1.235 :: Slave*  
*Database database user :: root*  
*Database password :: ubuntu*

## *About MySQL replication*

*Here i configure master-slave for two different scenario*  
*1 ) For a particular database replication*  
*2 ) For a all database replication*

MySQL replication is a process that allows you to easily maintain multiple copies of a MySQL data by having them copied automatically from a master to a slave database. This can be helpful for many reasons including facilitating a backup for the data, a way to analyze it without using the main database, or simply as a means to scale out.

This tutorial will cover a very simple example of mysql replication—one master will send information to a single slave. For the process to work you will need two IP addresses: one of the master server and one of the slave.

This tutorial will use the following IP addresses:

*192.168.1.45 :: Master Database*  
*192.168.1.47 :: Slave*  
*192.168.1.235 :: Slave*  
*Database database user :: root*  
*Database password :: ubuntu*

## Setup

This article assumes that you have user with sudo privileges and have MySQL installed. If you do not have mysql, you can install it with this command:

```
# sudo apt-get install mysql-server mysql-client
```

\*\*\*\*\*

## Step One—Configure the Master Database

\*\*\*\*\*

Open up the mysql configuration file on the master server.

```
# sudo nano /etc/mysql/my.cnf
```

Once inside that file, we need to make a few changes.

The first step is to find the section that looks like this, binding the server to the local host:

```
bind-address          = 127.0.0.1
```

comment this line

```
#bind-address        = 127.0.0.1
```

The next configuration change refers to the server-id, located in the [mysqld] section. You can choose any number for this spot (it may just be easier to start with 1), but the number must be unique and cannot match any other server-id in your replication group. I'm going to go ahead and call this one 1.

Make sure this line is uncommented.

```
server-id            = 1
```

Move on to the log\_bin line. This is where the real details of the replication are kept. The slave is going to copy all of the changes that are registered in the log. For this step we simply need to uncomment the line that refers to log\_bin:

```
log_bin              = /var/log/mysql/mysql-bin.log
```

Finally, we need to designate the database that will be replicated on the slave server. You can include more than one database by repeating this line for all of the databases you will need

**For a particular database ::**

```
binlog_do_db         = newdatabase
```

**For all database ::**

```
#binlog_do_db        = newdatabase
```

```
root@ubuntu-server: /home/ubuntu
GNU nano 2.2.6                               File: /etc/mysql/my.cnf                               Modified
log_error = /var/log/mysql/error.log
#
# Here you can see queries with especially long duration
#log_slow_queries      = /var/log/mysql/mysql-slow.log
#long_query_time = 2
#log-queries-not-using-indexes
#
# The following can be used as easy to replay backup logs or for replication.
# note: if you are setting up a replication slave, see README.Debian about
# other settings you may need to change.
#
server-id              = 1
log_bin                = /var/log/mysql/mysql-bin.log
expire_logs_days       = 10
max_binlog_size         = 100M
#binlog_do_db           = include_database_name
#binlog_ignore_db       = include_database_name
#
# * InnoDB
#
# InnoDB is enabled by default with a 10MB datafile in /var/lib/mysql/.
# Read the manual for more InnoDB related options. There are many!
#
# * Security Features
#
# Read the manual, too, if you want chroot!
# chroot = /var/lib/mysql/
#
^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^M Where Is      ^V Next Page     ^U UnCut Text    ^T To Spell
```

After you make all of the changes, go ahead and save and exit out of the configuration file.

Refresh MySQL.

**# *sudo service mysql restart***

The next steps will take place in the MySQL shell, itself.

Open up the MySQL shell.

**# *mysql -u root -p***

We need to grant privileges to the slave. You can use this line to name your slave and set up their password. The command should be in this format:

**> *GRANT REPLICATION SLAVE ON \*.\* TO 'slave\_user'@'%' IDENTIFIED BY 'password';***

Follow up with:

**> *FLUSH PRIVILEGES;***

***For particular database ::***

The next part is a bit finicky. To accomplish the task you will need to open a *new window or tab* in addition to the one that you are already using a few steps down the line.

In your *current tab* switch to “newdatabase”.

**> *Create database newdatabase;***

**> *USE newdatabase;***

Following that, lock the database to prevent any new changes:

**> *FLUSH TABLES WITH READ LOCK;***

Then type in:

**> *SHOW MASTER STATUS;***

You will see a table that should look something like this:

```
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000001 | 107 | newdatabase | |
+-----+-----+-----+-----+
```

**1 row in set (0.00 sec)**

This is the position from which the slave database will start replicating. Record these numbers, they will come in useful later.

If you make any new changes in the same window, the database will automatically unlock. For this reason, you should open the new tab or window and continue with the next steps there.

Proceeding the with the database still locked, export your database using mysqldump in the new window (make sure you are typing this command in the bash shell, not in MySQL).

```
# mysqldump -u root -p --opt newdatabase > newdatabase.sql
```

Now, returning to your your original window, unlock the databases (making them writeable again). Finish up by exiting the shell.

```
> UNLOCK TABLES;
```

```
> QUIT;
```

Now you are all done with the configuration of the the master database.

### For all databases ::

The next part is a bit finicky. To accomplish the task you will need to open a *new window or tab* in addition to the one that you are already using a few steps down the line.

Following that, lock the database to prevent any new changes:

```
> FLUSH TABLES WITH READ LOCK;
```

Then type in:

```
> SHOW MASTER STATUS;
```

You'll see something like this:

```
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000037 | 14462 | | |
+-----+-----+-----+-----+
```

**1 row in set (0.00 sec)**

My server has been logging for some time, hence the .000037 at the end of the log fie. Make a note of these coordinates. If all goes well you won't need them – but if you run into trouble, these will come in handy.

If you make any new changes in the same window, the database will automatically unlock. For this reason, you should open the new tab or window and continue with the next steps there.

```
# mysqldump --all-databases --user=root --password --master-data > everything.sql
```

Now, returning to your your original window, unlock the databases (making them writeable again). Finish up by exiting the shell.

```
> UNLOCK TABLES;
```

**> QUIT;**

Now you are all done with the configuration of the master database.

\*\*\*\*\*

## *Step Two—Configure the Slave Database*

\*\*\*\*\*

Once you have configured the master database. You can put it aside for a while, and we will now begin to configure the slave database.

Log into your slave server, open up the MySQL shell and create the new database that you will be replicating from the master (then exit):

*For particular database ::*

**> CREATE DATABASE newdatabase;**

**> EXIT;**

*For all databases ::*

no need to create databases

Import the database that you previously exported from the master database.

*For particular database ::*

**# mysql -u root -p newdatabase < /path/to/newdatabase.sql**

*For all databases ::*

**# mysql -u root -p < everything.sql**

Now we need to configure the slave configuration in the same way as we did the master:

**# sudo nano /etc/mysql/my.cnf**

Once inside that file, we need to make a few changes.

The first step is to find the section that looks like this, binding the server to the local host:

**bind-address = 127.0.0.1**

comment this line

**#bind-address = 127.0.0.1**

We have to make sure that we have a few things set up in this configuration. The first is the

server-id. This number, as mentioned before needs to be unique. Since it is set on the default (still 1), be sure to change it's something different.

**server-id** = 2

Following that, make sure that your have the following three criteria appropriately filled out:

**relay-log** = /var/log/mysql/mysql-relay-bin.log

**log\_bin** = /var/log/mysql/mysql-bin.log

**For particular database ::**

**binlog\_do\_db** = newdatabase

**For all database ::**

**#binlog\_do\_db** = newdatabase

You will need to add in the relay-log line: it is not there by default. Once you have made all of the necessary changes, save and exit out of the slave configuration file.

Restart MySQL once again:

**# sudo service mysql restart**

The next step is to enable the replication from within the MySQL shell.

Open up the the MySQL shell once again and type in the following details, replacing the values to match your information:

This command accomplishes several things at the same time:

1. It designates the current server as the slave of our master server.
2. It provides the server the correct login credentials
3. Last of all, it lets the slave server know where to start replicating from; the master log file and log position come from the numbers we wrote down previously.

With that—you have configured a master and slave server.

Activate the slave server:

**> START SLAVE;**

You be able to see the details of the slave replication by typing in this command. The \G rearranges the text to make it more readable.

**> SHOW SLAVE STATUS\G;**

If there is an issue in connecting, you can try starting slave with a command to skip over it:

**> SET GLOBAL SQL\_SLAVE\_SKIP\_COUNTER = 1; SLAVE START;**

\*\*\*\*\*

### *Testing ::*

#### *For particular database ::*

create some tables inside that particular database in master and check whether it is replicate on slave side or not

> *CREATE TABLE testing( ID INT NOT NULL);*

#### *For all database ::*

create some database in master and check whether it is replicate on slave side or not

> *create database new;*

\*\*\*\*\*

\*\*\*\*\*

### *links ::*

For particular database ::

<https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-in-mysql>

For all databases ::

<http://wpguru.co.uk/2013/05/how-to-setup-mysql-masterslave-replication-with-existing-data/>

\*\*\*\*\*

\*\*\*\*\*

## 2) ssh access as root

\*\*\*\*\*

on mha manager as root

1) create key with ssh-keygen

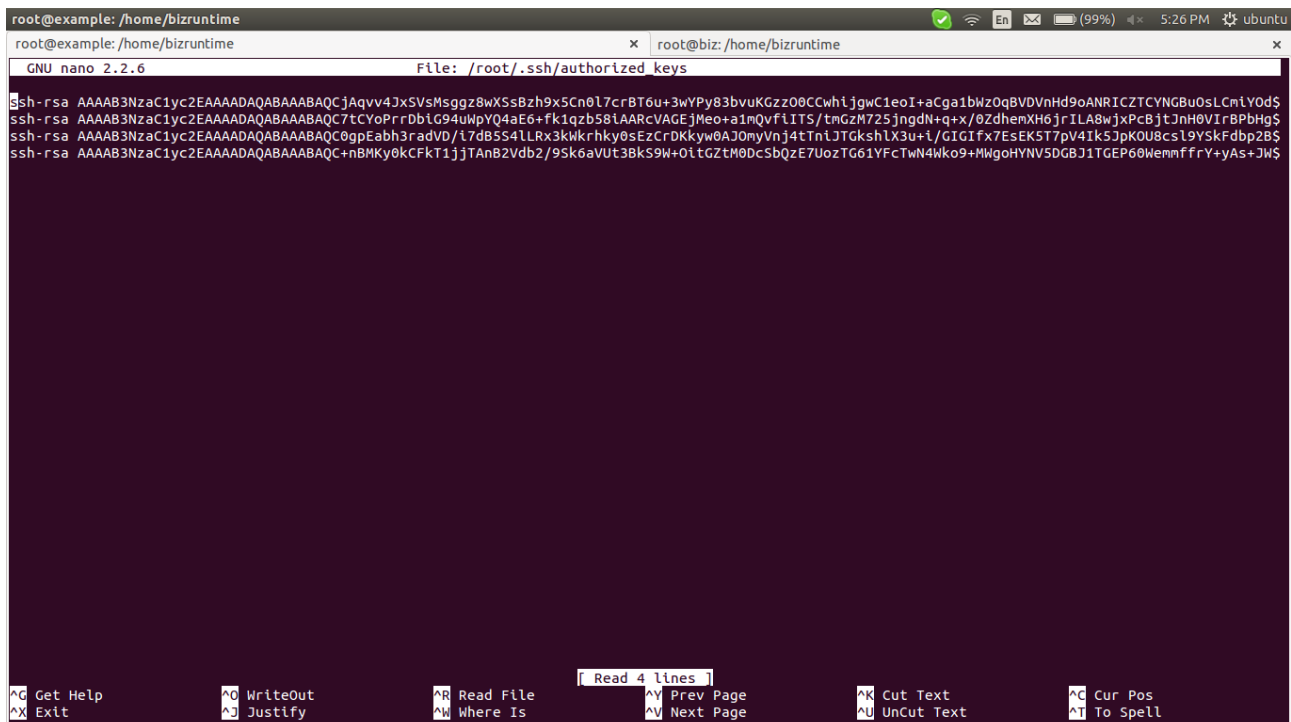
```
# ssh-keygen
# cd /root/.ssh/
# cat id_rsa.pub
```

2) go to mysql-master and copy this and paste it in nano /root/.ssh/authorized\_keys

```
# cat id_rsa.pub >> authorized_keys
```

3) copy the id\_rsa.pub from all other mysql-slaves and paste in this same file called authorized\_keys

```
# cat authorized_keys
```



```
root@example: /home/bizruntime
root@example: /home/bizruntime
GNU nano 2.2.6 File: /root/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAjAqv4JxSVsMsgg28wXSsBzh9x5Cn0L7crBT6u+3wYPy83bvukGzz08CCwhijgwC1eoI+aCga1bwz0qBVDVnHd9oANRICZTCYNGBu0sLCnIY0d$
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAj7tCYoPrRDbiG94uWpY04E6+fk1qzb58IAARcVAGEjMeo+a1mQvfiITS/tngZM725jngdN+q+x/0ZdhemXH6jrILA8wjxPcBjtJnH0VIRBPbHg$
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAj0pEabh3radVD/i7dB5S4LLRx3kWrky0sEzCrDKkyw0AJomyVnj4tTnIjTGkshLX3u+I/GIGIfx7EsEK5T7pV4Ik5JpK0U8cs19YskFdbp2B$
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCA+nBMKy0kCFkT1jjTAnB2Vdb2/9Sk6aVUt3BkS9W+0ltGZtM0DcSbQzE7UozTG61YFcTwN4Wko9+MWgoHYNV5DGBJ1TGEp60WemmfFrY+yAs+JWS$

^G Get Help      ^O WriteOut      ^R Read File     Read 4 lines
^X Exit          ^J Justify       ^W Where Is      ^V Prev Page
                  ^Y Next Page     ^C Cut Text       ^U UnCut Text
                  ^T To Spell
```

this file containing all ssh public keys of mysql-master, mysql-slaves and mha manager

6) ensure that this file copied to,

- a) all mysql-slaves
- b) mysql-master
- c) mha manager



\*\*\*\*\*

### 3) MHA Configuration

\*\*\*\*\*

A primary objective of MHA is automating master failover and slave promotion within short (usually 10-30 seconds) downtime, without suffering from replication consistency problems, without performance penalty, without complexity, and without changing existing deployments. It runs on top of existing MySQL installations and uses standard MySQL replication both asynchronous and semi-synchronous.

The main features are:

- Automatic failover
- Short downtime
- MySQL-Replication consistency
- Easy installation
- No change to existing deployments

MHA is delivered in two packages, MHA Node and MHA Manager. The node package will be installed on both servers, and the manager package only on the slave server. (If you install the manager on the master server, if the master goes down MHA will be completely useless...yep).

Install MHA Node and additional libraries on both servers: (complete those steps in SSH on master AND slave servers):

go to <https://downloads.mariadb.com/MHA/>

and download the node “mha4mysql-node\_0.54-0\_all.deb” for all machine.

*# apt-get install libdbd-mysql-perl*

*# dpkg -i mha4mysql-node\_0.54-0\_all.deb*

Install MHA Manager and additional libraries on MHA Manager

go to <https://downloads.mariadb.com/MHA/>

*and download the node “mha4mysql-manager\_0.55-0\_all.deb” for all machine.*

*# apt-get install libdbd-mysql-perl*

*# apt-get install libconfig-tiny-perl*

*# apt-get install liblog-dispatch-perl*

*# apt-get install -y libswitch-perl*

*# apt-get install libparallel-forkmanager-perl*

*# dpkg -i mha4mysql-manager\_0.55-0\_all.deb*

Create a configuration file on this server to setup MHA Manager:

*# nano /etc/mha/app1.cnf*

*[server default]*

*# mysql user and password*

*user=mhauser*

*password=password*

*ssh\_user=root*

*# working directory on the manager*

*manager\_workdir=/var/log/masterha/app1*

*# working directory on MySQL servers*

*remote\_workdir=/var/log/masterha/app1*

*master\_ip\_failover\_script=/etc/mha/master\_ip\_failover*

*[server1]*

*hostname=192.168.1.45*

*[server2]*

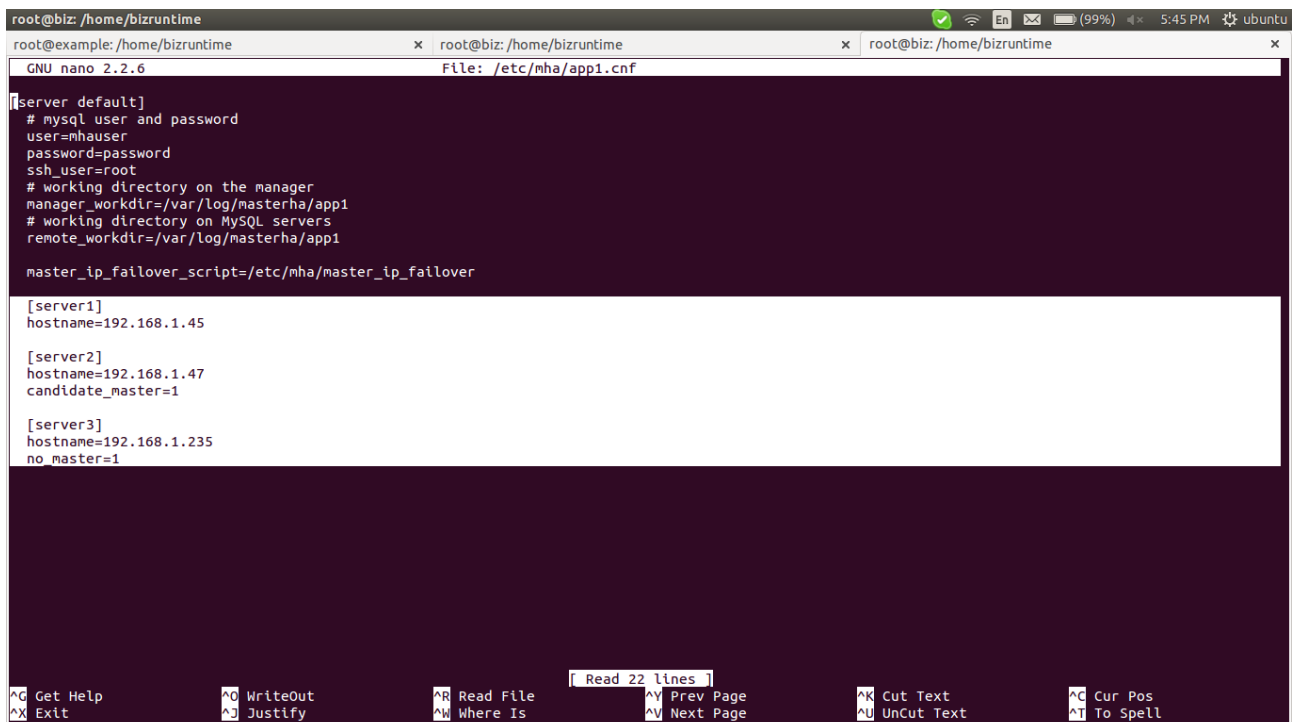
*hostname=192.168.1.47*

*candidate\_master=1*

*[server3]*

*hostname=192.168.1.235*

*no\_master=1*



```
root@biz: /home/bizruntime
root@example: /home/bizruntime x root@biz: /home/bizruntime x root@biz: /home/bizruntime x
GNU nano 2.2.6 File: /etc/mha/app1.cnf

[server default]
# mysql user and password
user=mhauser
password=password
ssh_user=root
# working directory on the manager
manager_workdir=/var/log/masterha/app1
# working directory on MySQL servers
remote_workdir=/var/log/masterha/app1

master_ip_failover_script=/etc/mha/master_ip_failover

[server1]
hostname=192.168.1.45

[server2]
hostname=192.168.1.47
candidate_master=1

[server3]
hostname=192.168.1.235
no_master=1

^G Get Help      ^O WriteOut      ^R Read File     Read 22 lines
^X Exit          ^J Justify       ^W Where Is      ^Y Prev Page
                  ^K Cut Text      ^U Uncut Text    ^C Cur Pos
                  ^T To Spell
```

Be sure that both hostnames can be resolved by both servers; add entries in /etc/hosts file if needed or put IP addresses instead of hostnames.

The "master \_ ip \_ failover" script will be called by MHA when the master server goes down, to switch the VIP from the dead server to the new promoted master.  
Grant access to "mhauser" on both servers: (this user is just used by MHA):

```
# mysql -u root -p
> grant all on *.* to 'mhauser'@'%' identified by 'password';
> flush privileges;
```

Create the "master \_ ip \_ failover" script:

```
# nano /etc/mha/master_ip_failover
```

And fill it in with the code below: (This script will switch the VIP from the dead server to the new promoted master server. Please be sure to change the 2 addresses):

```
#!/usr/bin/env perl
```

```
use strict;
use warnings FATAL => 'all';
```

```
use Getopt::Long;
```

```
use Net::Ping;
use Switch;
```

```
my ($command, $ssh_user, $orig_master_host, $orig_master_ip, $orig_master_port,
    $new_master_host, $new_master_ip, $new_master_port, $new_master_user,
    $new_master_password);
```

```
GetOptions(
    'command=s'          => \$command,
    'ssh_user=s'         => \$ssh_user,
    'orig_master_host=s' => \$orig_master_host,
    'orig_master_ip=s'  => \$orig_master_ip,
    'orig_master_port=i' => \$orig_master_port,
    'new_master_host=s'  => \$new_master_host,
    'new_master_ip=s'   => \$new_master_ip,
    'new_master_port=i' => \$new_master_port,
    'new_master_user=s'  => \$new_master_user,
    'new_master_password=s' => \$new_master_password,
);
```

```
my $vip = '192.168.1.45'; # Virtual IP
my $master_srv = '192.168.1.45';
```

```

my $timeout = 5;
my $key = "1";
my $ssh_start_vip = "sudo /sbin/ifconfig eth0:$key $vip";
my $ssh_stop_vip = "sudo /sbin/ifconfig eth0:$key down";

exit &main();

sub main {

print "\n\nIN SCRIPT TEST====$ssh_stop_vip==$ssh_start_vip====\n\n";

if ( $command eq "stop" || $command eq "stopssh" ) {

    # $orig_master_host, $orig_master_ip, $orig_master_port are passed.
    # If you manage master ip address at global catalog database,
    # invalidate orig_master_ip here.
    my $exit_code = 1;
    eval {
        print "Disabling the VIP on old master if the server is still UP: $orig_master_host\n";
        my $p=Net::Ping->new('icmp');
        &stop_vip() if $p->ping($master_srv, $timeout);
        $p->close();
        $exit_code = 0;
    };
    if ($?) {
        warn "Got Error: $@\n";
        exit $exit_code;
    }
    exit $exit_code;
}
elsif ( $command eq "start" ) {

    # all arguments are passed.
    # If you manage master ip address at global catalog database,
    # activate new_master_ip here.
    # You can also grant write access (create user, set read_only=0, etc) here.
    my $exit_code = 10;
    eval {
        print "Enabling the VIP - $vip on the new master - $new_master_host\n";
        &start_vip();
        $exit_code = 0;
    };
    if ($?) {
        warn $@;
        exit $exit_code;
    }
    exit $exit_code;
}
elsif ( $command eq "status" ) {
    print "Checking the Status of the script.. OK\n";
    #`ssh $ssh_user@$new_master_host \" $ssh_start_vip \";`
    exit 0;
}

```

```

}
else {
    &usage();
    exit 1;
}
}

# A simple system call that enable the VIP on the new master
sub start_vip() {
# `ssh $ssh_user@$new_master_host \" $ssh_start_vip \"`;
`ifconfig eth0:$key $vip`;
}

# A simple system call that disable the VIP on the old_master
sub stop_vip() {
`ssh $ssh_user@$orig_master_host \" $ssh_stop_vip \"`;
}

sub usage {
print
"Usage: master_ip_failover --command=start|stop|stopssh|status --orig_master_host=host
--orig_master_ip=ip --orig_master_port=port --new_master_host=host --new_master_ip=ip
--new_master_port=port\n";
}

```

Make this script executable:

*# chmod +x /etc/mha/master\_ip\_failover*

```

root@biz: /home/bizruntime
root@example: /home/bizruntime
root@biz: /home/bizruntime
GNU nano 2.2.6 File: /etc/mha/master_ip_failover

#!/usr/bin/env perl

use strict;
use warnings FATAL => 'all';
use Getopt::Long;

use Net::Ping;
use Switch;

my ($command, $ssh_user, $orig_master_host, $orig_master_ip, $orig_master_port, $new_master_host, $new_master_ip, $new_master_port, $new_master_user, $new_master_password);

GetOptions(
    'command=s' => \$command,
    'ssh_user=s' => \$ssh_user,
    'orig_master_host=s' => \$orig_master_host,
    'orig_master_ip=s' => \$orig_master_ip,
    'orig_master_port=i' => \$orig_master_port,
    'new_master_host=s' => \$new_master_host,
    'new_master_ip=s' => \$new_master_ip,
    'new_master_port=i' => \$new_master_port,
    'new_master_user=s' => \$new_master_user,
    'new_master_password=s' => \$new_master_password,
);

my $vip = '192.168.1.45'; # Virtual IP
my $master_srv = '192.168.1.45';
my $timeout = 5;
my $key = "1";
my $ssh_start_vip = "sudo /sbin/ifconfig eth0:$key $vip";
my $ssh_stop_vip = "sudo /sbin/ifconfig eth0:$key down";

exit &main();

sub main {
    print "\n\nIN SCRIPT TEST====$ssh_stop_vip==$ssh_start_vip====\n\n";

    if ( $command eq "stop" || $command eq "stopssh" ) {

```

MHA Manager internally invokes programs included in the MHA Node package via SSH. MHA Node programs also send differential relay log files to other non-latest slaves via SSH (scp).

To make these procedures non-interactive, it is necessary to setup SSH public key authentication. MHA Manager provides a simple check program "masterha \_ check \_ ssh" to verify non-interactive SSH connections can be established each other.

To test if everything is OK with SSH configuration, type the following command in the slave (MHA Manager) SSH shell:

```
# masterha_check_ssh --conf=/etc/mha/app1.cnf
```

Please be sure that you have the "MySQL Replication Health is OK." message at the end before moving further. If not, you will need to fix errors shown in the logs.

As you can see at the end of the logs, MHA automatically detects which server is the master, and which one is the slave.

Now that you configured MySQL replication, installed both MHA Node and MHA Manager, and configured SSH public key authentication, next step is starting MHA Manager.

MHA Manager can be started by "masterha \_ manager" command on the MHA Manager server. In our case, we will start it in background as below:

```
# nohup masterha_manager --conf=/etc/mha/app1.cnf < /dev/null >
/var/log/masterha/app1/app1.log 2>&1 &
```

You can now check MHA status at any time with the command:

```
# masterha_check_status --conf=/etc/mha/app1.cnf
```

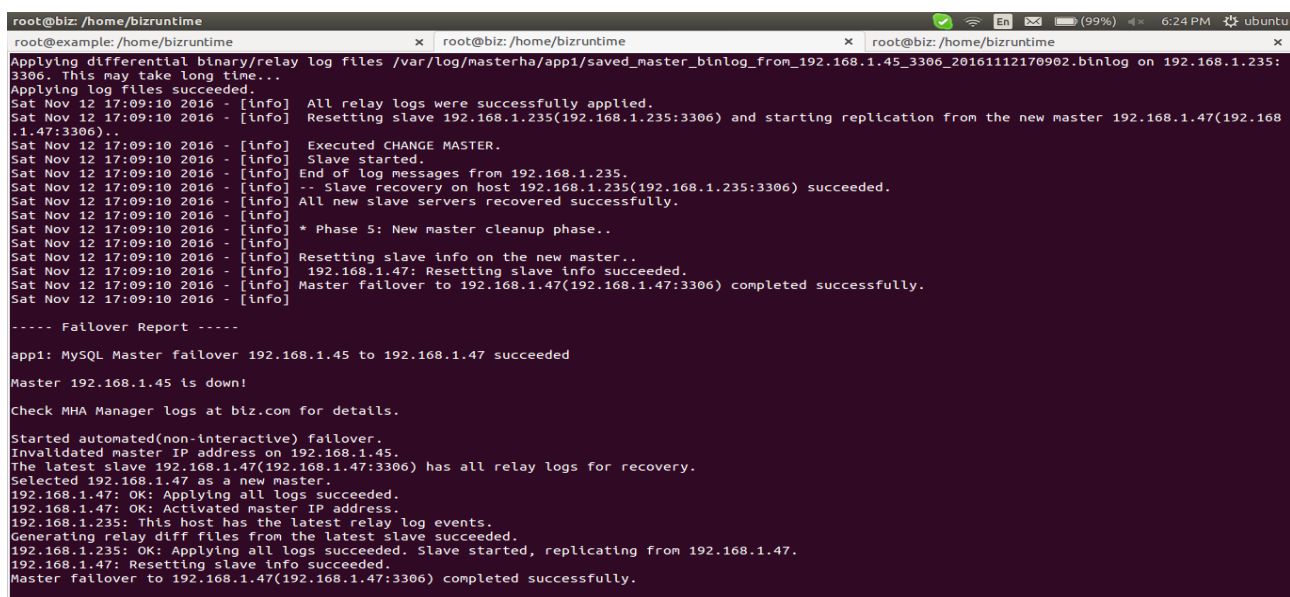
Now that everything is correctly configured, we will test the most important part, the master automatic failover.

Type the following command in the MySQL master SSH shell (or halt/reboot the server):

```
# /etc/init.d/mysql stop
```

Then, on the MySQL slave, (which is the MHA Manager server), check the MHA logs by typing:

```
# tail -f /var/log/masterha/app1/app1.log
```



```
root@biz: /home/bizruntime
root@example: /home/bizruntime x root@biz: /home/bizruntime x root@biz: /home/bizruntime x
Applying differential binary/relay log files /var/log/masterha/app1/saved_master_binlog_from_192.168.1.45_3306_20161112170902.binlog on 192.168.1.235:
3306. This may take long time...
Applying log files succeeded.
Sat Nov 12 17:09:10 2016 - [info] All relay logs were successfully applied.
Sat Nov 12 17:09:10 2016 - [info] Resetting slave 192.168.1.235(192.168.1.235:3306) and starting replication from the new master 192.168.1.47(192.168
.1.47:3306)..
Sat Nov 12 17:09:10 2016 - [info] Executed CHANGE MASTER.
Sat Nov 12 17:09:10 2016 - [info] Slave started.
Sat Nov 12 17:09:10 2016 - [info] End of log messages from 192.168.1.235.
Sat Nov 12 17:09:10 2016 - [info] -- Slave recovery on host 192.168.1.235(192.168.1.235:3306) succeeded.
Sat Nov 12 17:09:10 2016 - [info] All new slave servers recovered successfully.
Sat Nov 12 17:09:10 2016 - [info]
Sat Nov 12 17:09:10 2016 - [info] * Phase 5: New master cleanup phase..
Sat Nov 12 17:09:10 2016 - [info]
Sat Nov 12 17:09:10 2016 - [info]
Sat Nov 12 17:09:10 2016 - [info] Resetting slave info on the new master..
Sat Nov 12 17:09:10 2016 - [info] 192.168.1.47: Resetting slave info succeeded.
Sat Nov 12 17:09:10 2016 - [info] Master failover to 192.168.1.47(192.168.1.47:3306) completed successfully.
Sat Nov 12 17:09:10 2016 - [info]

----- Failover Report -----
app1: MySQL Master failover 192.168.1.45 to 192.168.1.47 succeeded
Master 192.168.1.45 is down!
Check MHA Manager logs at biz.com for details.
Started automated(non-interactive) failover.
Invalidated master IP address on 192.168.1.45.
The latest slave 192.168.1.47(192.168.1.47:3306) has all relay logs for recovery.
Selected 192.168.1.47 as a new master.
192.168.1.47: OK: Applying all logs succeeded.
192.168.1.47: OK: Activated master IP address.
192.168.1.235: This host has the latest relay log events.
Generating relay diff files from the latest slave succeeded.
192.168.1.235: OK: Applying all logs succeeded. Slave started, replicating from 192.168.1.47.
192.168.1.47: Resetting slave info succeeded.
Master failover to 192.168.1.47(192.168.1.47:3306) completed successfully.
```

now the configuration is changed as follow,

*192.168.1.234 :: MHA Manager*  
*192.168.1.45 :: Master Database (down)*  
*192.168.1.47 :: Master Database (up)*  
*192.168.1.235 :: Slave*

now the “192.168.1.235” is a slave server for “192.168.1.47” not “192.168.1.45”  
When the failover is complete, the MHA script will stop, and you will need to manually perform the steps below to reconfigure the cluster:

Reboot 192.168.1.45 (old master).

Make a clean dump of the database (with lock) 192.168.1.47 (new master):

```
# mysqldump --all-databases --user=root --password --master-data > everything.sql
```

```
# scp everything.sql 192.168.1.45:/home/ubuntu/
```

on that mysql-old master (192.168.1.45) run this command:

```
# mysql -u root -p < everything.sql
```

on new-master (192.168.1.47)

```
> mysql -u root -p  
> show master status;
```

if it is vip then;

Enable VIP on old server:

```
# ifconfig eth0:1 192.168.10.238
```

Disable VIP on new server:

```
# ifconfig eth0:1 down
```

Disable locks on BDD2 and reconfigure replication (you will need to adjust the log file and position according to those shown on the master server):

on mysql-slaves ;

For particular database ::

```
> CHANGE MASTER TO MASTER_HOST='192.168.1.45', MASTER_USER='slave_user',  
MASTER_PASSWORD='password', MASTER_LOG_FILE='mysql-bin.000001',  
MASTER_LOG_POS= 107;
```

For all database ::

```
> change master to MASTER_HOST='192.168.1.45', MASTER_USER='slave_user',  
MASTER_PASSWORD='password';
```

```
> start slave;  
> show slave status\G
```

Remove MHA logs files on MHA Manager:

```
# rm /var/log/masterha/app1/app1.failover.complete  
# rm /var/log/masterha/app1/app1.failover.error
```

Restart MHA on MHA Manager:

```
# masterha_check_repl --conf=/etc/mha/app1.cnf  
nohup masterha_manager --conf=/etc/mha/app1.cnf < /dev/null >  
/var/log/masterha/app1/app1.log 2>&1 &
```



\*\*\*\*\*

### **Testing ::**

create some database in master and check whether it is replicate on slave side or not {after failover}  
(create database in new-master and check it reflect on remaining slaves)

> *create database new;*

\*\*\*\*\*

\*\*\*\*\*

### **links ::**

#### **failover configuration ::**

<http://www.arborissoft.com/how-to-configure-mysql-masterslave-replication-with-mha-automatic-failover/>

<http://dbpandit.com/mysql-failover-with-mha-master-high-availability/>

<https://ruiaylin.github.io/2015/03/03/master-ha-install-conf/>

#### **MHA installation ::**

<https://downloads.mariadb.com/MHA/>

#### **failover configuration with haproxy ::**

<http://www.linux-admins.net/2014/10/deploying-highly-available-mysql-with.html>

#### **failover overview ::**

<http://www.howto-expert.com/how-to-create-a-server-failover-solution/>

<http://www.slideshare.net/matsunobu/automated-master-failover>

#### **fsailover concept in mysql-cluster (different) ::**

<http://www.alexwilliams.ca/blog/2009/08/10/using-haproxy-for-mysql-failover-and-redundancy/index.html#HAProxyConfig>

\*\*\*\*\*