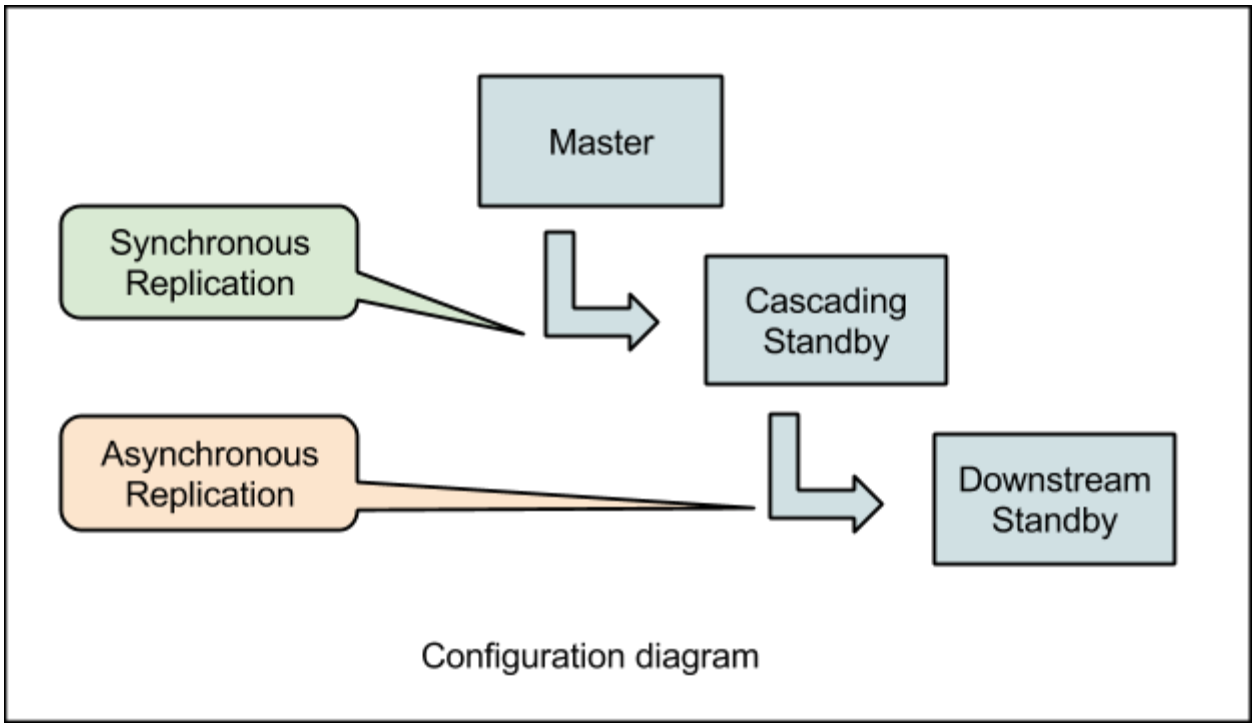
High Availability with PostgreSQL



Install PostgreSQL v9.3

1. Follow the instructions in http://www.postgresql.org/download/linux/ubuntu/ to add the PostgreSQL Apt Repository (for Ubuntu 13.04 replace YOUR_UBUNTU_VERSION_HERE with precise)

2. Install PostgreSQL
   `sudo apt-get install postgresql-9.3 postgresql-contrib-9.3`
3. After this, PostgreSQL creates, configures and starts a default database cluster (*) named “main” using the following locations:

Data directory	/var/lib/postgresql/9.3/main/
Configuration directory	/etc/postgresql/9.3/main/
Binaries directory	/usr/lib/postgresql/9.3/
Log file	/var/log/postgresql/postgresql-9.3-main.log

   (*) Don't be confused. In that case the word "cluster" is used to indicate a cluster of databases in the same computer (according to PostgreSQL documentation), not a set of connected computers that work together as a whole.

4. Set up passwordless access over ssh for the postgres user used in the `archive_command` (only if not already done). You must copy the generated public key to the others servers to allow access from any server to any other one. Before copy the public key you need to create a password for the user postgres for each host.
   `sudo su postgres`
   `ssh-keygen -t dsa`
   `ssh-copy-id -i ~/.ssh/id_dsa.pub <server_to_access_ip>`
   To verify if it works do the following (you shouldn’t be asked to enter a password):
   `ssh <server_to_access_ip>`
5. Repeat the previous steps for each server: Master, Cascading Standby and Downstream Standby.

Setting up a Master server

1. Change to the `postgres` user (by default the cluster's owner)
   `sudo su postgres`
2. Create the user used for the replication (see Script to configure a Master/Hot Standby server)
   `psql -c "CREATE USER replicator REPLICATION LOGIN ENCRYPTED PASSWORD 'secret'"`
3. Stop the cluster
   `pg_ctlcluster 9.3 main stop` (see how to show a db cluster information)
4. Edit the **/etc/postgresql/9.3/main/postgresql.conf** configuration file and change the following settings:

Name	Value

`listen_addresses`	`'*'`
`wal_level`	`hot_standby`
`archive_mode`	`on`
`archive_command`	`'rsync -av %p postgres@<standby_ip>:/var/lib/postgresql/wal_archive/%f'`
`max_wal_senders`	`3`
`wal_keep_segments`	`32`
`synchronous_standby_names`	`'*'`

TODO: add a brief explanation of each setting

5. Edit the **/etc/postgresql/9.3/main/pg_hba.conf** configuration file and allow connections from the standby server and Cyclos:

host	replication	replicator	<sub_net>/<stand_by_ip>	md5
host	all	<user>/all	<sub_net>/<cyclos_ip>	md5

6. Start the cluster
   `pg_ctlcluster 9.3 main start`

Setting up a Cascading (Upstream) Standby server

1. Follow the steps as for the Master (Setting up a Master server) except the last one (the Master initialization). Those settings in postgresql.conf will be ignored when running as a standby but will be needed in case of failover. For the `archive_command` **REPLACE** the <standby_ip> with the IP of the Downstream Standby.
2. Additionally, the following setting must be enabled in **/etc/postgresql/9.3/main/postgresql.conf**
   `hot_standby = on`
3. Create the WAL directory
   `mkdir /var/lib/postgresql/wal_archive`
   (the postgres user must have write permissions on this directory. See the archive_command configured in the Master)
4. Clean the data directory
   `rm -r /var/lib/postgresql/9.3/main/*`
5. Take a fresh backup from the Master server
   `pg_basebackup -D /var/lib/postgresql/9.3/main -Fp -P -v -Xs -U replicator -W -h <master_ip> [-p <master_port>]`
6. Create a file **/var/lib/postgresql/9.3/main/recovery.conf** and set the following:

Name	Value
standby_mode	'on'
primary_conninfo	'host=<master_ip> port=5432 application_name=standby_1 user=replicator password=secret'
restore_command	'cp /var/lib/postgresql/wal_archive/%f "%p"'
archive_cleanup_command	'/usr/lib/postgresql/9.3/bin/pg_archivecleanup /var/lib/postgresql/wal_archive/ %r'
recovery_target_timeline	'latest'

7. Start the cluster
   `pg_ctlcluster 9.3 main start`
8. Check the Master's log file to verify all is working as expected. You should see something similar to this:
   database system is ready to accept connections
   ...
   standby "standby_1" is now the synchronous standby with priority 1
9. Check the standby's log. You should see something like this:
   *database system is ready to accept **read only** connections*
   ...
   started streaming WAL from primary at ... on timeline 1

Setting up a Downstream Standby server

Setting up a downstream standby is basically the same as for the upstream standby (see Setting up a Cascading (Upstream) Standby server) with a minor changes.

1. Replace the <master_ip> with the upstream_ip
2. Choose a different name (e.g.: standby_2) for the `application_name` attribute of the `primary_conninfo` in the **/var/lib/postgresql/9.3/main/recovery.conf** file.
3. Start the cluster
   `pg_ctlcluster 9.3 main start`
4. Check the log file. You should see something like this:
   *database system is ready to accept **read only** connections*
   ...
   started streaming WAL from primary at ... on timeline 1

Script to configure a Master/Hot Standby server

If you prefer you can run (logged as the postgres user) the following script on each server to configure a Master/Standby node to apply all of those changes automatically:

```
#!/bin/bash

# ==================== show_help function ====================================
```

```
show_help() {
    echo
    echo "It configures a PostgreSQL v$PG_VERSION server to run as a Master/Hot Standby"
    echo
    echo "Usage: " $(basename $0)    "{clean} | {standby_server sub_net (used in
pg_hba.conf to allow access) [master_server standby_name]} "
    echo
    echo "          If a master_server is specified a standby_name must be specified too
and this means you are configuring a hot standby server."
    echo "          The 'clean' parameter will erase the information (all data will be
lost!) related to the '$CLUSTER_NAME' cluster and create a new one."
    echo "          If you are configuring a standby server and you don't have another
standby to cascade just put something like"
    echo "          standby_ip as the first parameter (this won't be used because a
standby works in recovery mode and that parameter is used in archive mode)."
}

# ==================== configure_master function ==============================
configure_master() {
    echo "Configuring Master server..."

    echo "Creating replicator user for replication/backup..."
    echo -n "Enter password for replicator user:"
    read -s REPLICATOR_PWD
    echo
    # ----------------------------------------------------------------------
    psql -c "CREATE USER replicator REPLICATION LOGIN ENCRYPTED PASSWORD
'$REPLICATOR_PWD'"

    # ----------------------------------------------------------------------
    echo "Stopping server..."
    pg_ctlcluster $PG_VERSION $CLUSTER_NAME stop

    # ----------------------------------------------------------------------
    echo "Modifying settings in $PG_CONFIG_FILE..."

    echo "
listen_addresses = '*'
wal_level = hot_standby
archive_mode = on
archive_command = 'rsync -av %p postgres@$STANDBY_SERVER:$WAL_FOLDER/%f'
max_wal_senders = 3
wal_keep_segments = 32
synchronous_standby_names = '*'
" >> $PG_CONFIG_FILE

    # ----------------------------------------------------------------------
    echo "Modifying settings in $HBA_CONFIG_FILE..."

    echo "
```

```
    host     all          all          $SUB_NET/24   md5
    host     replication  replicator   $SUB_NET/24   md5
    " >> $HBA_CONFIG_FILE
}

# ================ configure_standby function ================================
configure_standby() {
    echo
    echo "Configuring Standby server with name $STANDBY_NAME..."

    # ---------------------------------------------------------------------
    echo "Modifying settings in $PG_CONFIG_FILE..."
    echo "
hot_standby = on
" >> $PG_CONFIG_FILE

    # ---------------------------------------------------------------------
    echo "Creating WAL folder $WAL_FOLDER..."
    rm -r $WAL_FOLDER
    mkdir $WAL_FOLDER

    # ---------------------------------------------------------------------
    echo "Cleaning data directory..."
    rm -r /var/lib/postgresql/$PG_VERSION/$CLUSTER_NAME/*

    echo "Taking a fresh backup from $MASTER_SERVER..."
    pg_basebackup -D /var/lib/postgresql/$PG_VERSION/$CLUSTER_NAME -Fp -P -v -Xs -U
replicator -W -h $MASTER_SERVER

    echo "Creating recovery file..."
    echo "
standby_mode = 'on'
primary_conninfo = 'host=$MASTER_SERVER port=$PG_PORT application_name=$STANDBY_NAME
user=replicator password=$REPLICATOR_PWD'
restore_command = 'cp $WAL_FOLDER/%f \"%p\"'
archive_cleanup_command = '/usr/lib/postgresql/$PG_VERSION/bin/pg_archivecleanup
$WAL_FOLDER/ %r'
recovery_target_timeline = 'latest'
" > $RECOVERY_CONFIG_FILE
}

# ================ reset_cluster function ====================================
reset_cluster() {
    echo -n "Reset cluster $CLUSTER_NAME (y/n)?"
    read resp
    echo

    if [ "$resp" == "y" ]
      then
        echo "Stopping cluster..."
```

```
        pg_ctlcluster $PG_VERSION $CLUSTER_NAME stop -m immediate

        echo "Dropping cluster..."
        pg_dropcluster $PG_VERSION $CLUSTER_NAME

        pg_createcluster $PG_VERSION $CLUSTER_NAME

        echo "Starting cluster..."
        pg_ctlcluster $PG_VERSION $CLUSTER_NAME start

        pg_lsclusters
        else
        echo "Clean was canceled!"
    fi

    exit 0
}

# ================================================================================
# ============================ BEGIN =============================================
# ================================================================================

# ========================== VARIABLES INITIALIZATION ==========================
STANDBY_SERVER=$1
SUB_NET=$2
MASTER_SERVER=$3
STANDBY_NAME=$4
CLUSTER_NAME=main
PG_VERSION=9.3
PG_PORT=5432

PG_CONFIG_FILE=/etc/postgresql/$PG_VERSION/$CLUSTER_NAME/postgresql.conf
HBA_CONFIG_FILE=/etc/postgresql/$PG_VERSION/$CLUSTER_NAME/pg_hba.conf
RECOVERY_CONFIG_FILE=/var/lib/postgresql/$PG_VERSION/$CLUSTER_NAME/recovery.conf
WAL_FOLDER=/var/lib/postgresql/wal_archive

# ========================== PARAMETERS VERIFICATION ==========================
# expected args are 1, 2 or 4
if [[ $# -ne 1 && $# -ne 2 && $# -ne 4 ]]
  then
    show_help
    exit 0
fi

if [[ $# -eq 1 && "$1" -eq "clean" ]]
  then
    reset_cluster
    exit 0
elif [ $# -eq 1 ]
  then
```

```
    echo "Invalid parameter: '$1'. 'clean' was expected."
    exit 1
fi


# ========================= FUNCTIONS INVOCATIONS ==========================
configure_master

if [ -n "$MASTER_SERVER" ]
  then
    configure_standby
fi

echo "Starting server..."
pg_ctlcluster $PG_VERSION $CLUSTER_NAME start

echo "Done!"
exit 0
```

Sample usage
Supose you want to configure as follow (the example is using the host name but it can works with IP too):

Server type	Host name
Master	master.server
Cascading	cascading.server
Downstream	downstream.server

Then you must execute the script in this order (the script was saved in a file named configure):
The 192.168.0.0 is the subnet_mask of the network (by default it use the first 24 bits of the mask: subnet_mask/24).

On maser: **`configure cascading.server 192.168.0.0`**
On cascading: **`configure downstream.server 192.168.0.0 master.server standby_1`**
On downstream: **`configure to_define 192.168.0.0 cascading.server standby_2`**

Monitor

Additionally, you could execute this query on each server to verify the overall configuration:
`select application_name,client_addr,state,sync_priority,sync_state from pg_stat_replication;`

The expected results should be.

On Master:

```
 application_name |   state   | sync_priority | sync_state
------------------+-----------+---------------+------------
 standby_1        | streaming |             1 | sync
```

On Cascading Standby:

```
 application_name |   state   | sync_priority | sync_state
------------------+-----------+---------------+------------
 standby_2        | streaming |             0 | async
```

On Downstream standby:

```
 application_name | state | sync_priority | sync_state
------------------+-------+---------------+------------
(0 rows)
```

At every moment you can run the following script to verify how the replication is working (just copy save it as shell script and set the right values for the variables):

```
#!/bin/bash

USER=cyclos
DB=cyclos4
MASTER=stroit11.local
UPSTREAM_STANDBY=stroit18.local
DOWNSTREAM_STANDBY=stroit15.local

MASTER_QUERY="SELECT pg_xlog_location_diff(pg_current_xlog_location(), '0/0') AS offset;"
STANDBY_QUERY="SELECT pg_xlog_location_diff(pg_last_xlog_receive_location(), '0/0') AS
receive, pg_xlog_location_diff(pg_last_xlog_replay_location(), '0/0') AS replay;"

exec_query() {
       h=$1    #host
       q=$2

       echo $(psql -A -t -U $USER -h $h -c "$q" $DB)
}

show_standby() {
       standby=$1
       standby_receive=$2
       standby_replay=$3
       master_offset=$4

       receive_diff=$(($master_offset - $standby_receive))
       replay_diff=$(($master_offset - $standby_replay))
```

```
        echo "  Standby:  "$standby
        echo "  receive: "$standby_receive bytes". Behind: " $(($receive_diff / 1024)) KB
        echo "   replay: "$standby_replay bytes". Behind: " $(($replay_diff / 1024)) KB
}

show_replication_status() {
        master=$1    #mandatory
        standby1=$2    #mandatory
        standby2=$3    #optional

        TMP=$(exec_query $standby1 "$STANDBY_QUERY")
        if [ $# -eq 3 ]
        then
        TMP2=$(exec_query $standby2 "$STANDBY_QUERY")
        fi

        master_offset=$(exec_query $master "$MASTER_QUERY")

        IFS='|' read -ra STANDBY_VALUES <<< "$TMP"
        standby1_receive=${STANDBY_VALUES[0]}
        standby1_replay=${STANDBY_VALUES[1]}

        if [ $# -eq 3 ]
        then
        IFS='|' read -ra STANDBY_VALUES <<< "$TMP2"
        standby2_receive=${STANDBY_VALUES[0]}
        standby2_replay=${STANDBY_VALUES[1]}

        receive2_diff=$(($master_offset - $standby2_receive))
        replay2_diff=$(($master_offset - $standby2_replay))
        fi



        echo "------------------------ REPLICATION STATUS -------------------------"
        echo "Master: "$master
        echo "  current: "$master_offset bytes
        echo
        show_standby $standby1 $standby1_receive $standby1_replay $master_offset
        if [ $# -eq 3 ]
        then
        echo
        show_standby $standby2 $standby2_receive $standby2_replay $master_offset
        fi
        echo "---------------------------------------------------------------------"
}

show_replication_status $MASTER $UPSTREAM_STANDBY $DOWNSTREAM_STANDBY
#show_replication_status $MASTER $UPSTREAM_STANDBY
```

Failover

Now, it is time to discuss how to recover/reconfigure the system if any of the running servers fails.

The Master fails

In this case the only thing we must do is promote the Cascading Standby server as the new master. Logged in the standby server this can be done with the following:

`pg_ctlcluster 9.3 main promote`

if you look at its log file you should find something like this:

```
received promote request
...
standby "standby_2" is now the synchronous standby with priority 1
```

In the log of the Downstream Standby server you should find this:

```
new target timeline is 2
...
restarted WAL streaming at 0/12000000 on timeline 2
```

After this, we have the servers running with synchronous replication enabled.
Of course, we should take action to leave this “degenerate state” as soon as possible. To achieve this, basically, there are two alternatives, to know:

1. Set up a new Downstream Standby connected to the (new) Cascading Standby.
2. Recreate the old master and switch the current master to recovery mode (again set up to function as a Cascading Standby) renaming the **/var/lib/postgresql/9.3/main/recovery.done** file to **recovery.conf** and restarting the server.

The Cascading Standby fails

This consequences of this fail are quite important because a commit of a write transaction on the master will wait until confirmation is received from the (crashed) Cascade Standby causing the master to stop to work.
To solve this we must reconnect the Downstream Standby directly to the the Master server editing the **/var/lib/postgresql/9.3/main/recovery.conf** file in the downstream server and put the Master’ IP. Also we must change the `archive_command` setting in the **/etc/postgresql/9.3/main/postgresql.conf file** of the Master to send the WAL segments to its new upstream server.
Restart the standby and just reload (restart is not required in this case) the Master:

`pg_ctlcluster 9.3 main restart` (on standby)
`pg_ctlcluster 9.3 main reload` (on master. Current connections are not closed!)

Check the Master’s log and you should see the following:

```
parameter "archive_command" changed to
...
```

*standby "**standby_2**" is now the synchronous standby with priority 1*

The Downstream Standby fails
In this case at first time there is nothing to do because the synchronous replication will continue working. Again, we should try to restore the original state setting up a new downstream standby as soon as possible.

Cluster Database Administration

To drop an existing cluster use:
pg_dropcluster --stop 9.3 main

To create a new cluster and start it immediately (by default the cluster is configured to be started/stopped automatically in the init script):
pg_createcluster --start 9.3 main

To start/stop an existing cluster use:
pg_ctlcluster 9.3 main start/stop

To show information about all database clusters in a host:
pg_lsclusters

Assuming this scenario:

Master = 192.168.1.11
Cascade = 192.168.1.12
Downstream = 192.168.1.13

Run this command on each machine:
uname -n

This will return the resolved hostname for each machine.
Put on each machine at /etc/hosts the IP's and its corresponding resolved name

192.168.1.11 master_hostname
192.168.1.12 cascade_hostname
192.168.1.13 downstream_hostname

It will be the same for each machine

At master and cascade we will use heartbeat for monitoring

Run on master and cascade machine

```
apt-get install heartbeat --no-install-recommends
```

In both servers, create a file named /etc/ha.d/ha.cf like this:

```
logfacility local0  # used to tell heartbeat which log facility to utilize for logging
keepalive 2         # interval between heartbeat packets currently every 2 secs you
could also use 2000ms
deadtime 5          # timeout before the other server takes over
ping 192.168.1.12   # physical address of the other server. Set the correct ip
udpport 694         # port to listen in on for broadcasts made by heartbeat
bcast eth0          # device to use for broadcasts
node masterHostname # dns name of the main server - should be the same as returned by
`uname -n`
node cascadeHostname# dns name of the failover server - should be the same as returned
by `uname -n`
auto_failback off
```

Then, also in both servers, create a file named /etc/ha.d/haresources like this:

master_hostname IPaddr::192.168.1.15/24/eth0 hapostgres

Finally, you need, in both servers, the /etc/ha.d/authkeys file, which must be equals in both servers:

```
auth 1
1 md5 1234
```

Set the correct permission at authkeys file:

```
chmod 600 /etc/ha.d/authkeys
```

We will use the 192.168.1.15 for database conection at our application.
And in case of failure the hapostgres service will be launched, so we need to create it:

At master machine create this file with the content below

```
nano /etc/init.d/hapostgres

      case $1 in
      start)
            ;;
      stop)
            ;;
      restart)
            ;;
      esac
      exit 0
```

Set the exec permission

chmod +x /etc/init.d/hapostgres

At the cascade machine create the same file but with this content:

```
nano /etc/init.d/hapostgres
      case $1 in
      start)
      sudo -u postgres  pg_ctlcluster 9.3 main promote
            ;;
      stop)
            ;;
      restart)
            ;;
      esac
      exit 0
```

Set the exec permission

```
chmod +x /etc/init.d/hapostgres
```

So you can start the heartbeat process on the master machine and after on the cascade:

```
service heartbeat start
```

If it's ok you should see something like this:

```
Starting High-Availability services: INFO:  Resource is stopped
Done.
```

Now you will have another ip address (192.168.1.15) on the master machine.

In case of failure this IP will be moved to the cascade machine and the /etc/init.d/hapostgres will be executed promoting the cascade machine as the new master.

To monitor the cascade we will use an ping script because we don't need another virtual ip or something else.

As in case of a failure in the cascade, we need to make a reload at the master server we will need an passwordless ssh for the postgres user from the downstream to the master server.

On the downstream server do this:

```
cp /var/lib/postgresql/9.3/main/recovery.conf
/var/lib/postgresql/9.3/main/recovery.conf.ori
cp /var/lib/postgresql/9.3/main/recovery.conf
/var/lib/postgresql/9.3/main/recovery.cascade
```

Put the ip or the hostname of the masterserver on the file recovery.cascade at primary_conninfo

On the master machine do:

```
cp /etc/postgresql/9.3/main/postgresql.conf /etc/postgresql/9.3/main/postgresql.conf.ori
cp /etc/postgresql/9.3/main/postgresql.conf /etc/postgresql/9.3/main/postgresql.cascade
```

Put the ip of the downstream on the archive_command at the file postgresql.cascade

At the master server we will create a new script: (make sure that the postgress user can run this script) /script/failover_cascade.sh
With this content:

```
cp /etc/postgresql/9.3/main/postgresql.cascade /etc/postgresql/9.3/main/postgresql.conf
pg_ctlcluster 9.3 main reload
```

set the exec permission for these file

chmod +x /script/failover_cascade.sh

At the downstream server we will create this 2 scripts:

/script/failover_cascade.sh With this content:

```
cp /var/lib/postgresql/9.3/main/recovery.cascade  /var/lib/postgresql/9.3/main/recovery.conf
pg_ctlcluster 9.3 main restart
ssh postgres@192.168.1.11 /script/failover_cascade.sh
```

set the exec permission for these file
chmod +x /script/failover_cascade.sh

Create these script:
/script/ping_monitor.sh with the content:

```
#!/bin/bash
set -x
/bin/ping -c1 192.168.1.12
if [ $? = 0 ];
then
echo "Resposta Ok!"
else
/script/failover_cascade.sh
fi
```

set the exec permission for these file
chmod +x /script/ping_monitor.sh

put the ping_monitor.sh to run every 2 or 5 seconds at the cron of the postgres user on the downstream server.

So this will be check if the ip of the cascade are live, and in case of failure they will run the /script/failover_cascade.sh

The /script/failover_cascade.sh will change the required files and will run the /script/failover_cascade.sh at the master machine.

After this the dowsntream server will be the new cascade server.