
Master Slave Replication on PostgreSQL-9.4

Introduction

PostgreSQL, or postgres, is a popular database management system that can organize and manage the data associated with websites or applications. Replication is a means of copying database information to a second system in order to create high availability and redundancy.

There are many ways to set up replication on a postgres system. In this tutorial, we will cover how to configure replication using a hot standby, which has the advantage of being relatively simple to configure.

To do this, we will need two Ubuntu 14.04 VPS instances. One will serve as the master database server and the other will function as a slave, which will replicate.

Install PostgreSQL Software

The steps in this section should be performed on **both** the master and slave servers.

The postgres software is available in Ubuntu's default repositories. Install the appropriate packages with these commands.

```
# sudo add-apt-repository "deb https://apt.postgresql.org/pub/repos/apt/ trusty-pgdg main"  
# sudo apt-get update  
# sudo apt-get install postgresql-9.4
```

PostgreSQL creates a user called "postgres" in order to handle its initial databases. We will configure ssh access between our servers to make transferring files easier.

We will need to set a password for the postgres user so that we can transfer the key files initially. If you desire, you can remove the password at a later time:

```
# sudo passwd postgres
```

Switch over to the postgres user like this:

```
# sudo su - postgres
```

Generate an ssh key for the postgres user:

```
# ssh-keygen
```

Press "ENTER" to all of the prompts that follow.

Transfer the keys to the other server by typing:

```
# ssh-copy-id IP_address_of_opposite_server
```

You should now be able to ssh freely between your two servers as the postgres user.

Configure the Master Server

We will begin by configuring our master server. All of these commands should be executed with the postgres user.

First, we will create a user called "rep" that can be used solely for replication:

```
# psql -c "CREATE USER rep REPLICATION LOGIN CONNECTION LIMIT 1 ENCRYPTED  
PASSWORD 'yourpassword';"
```

Change the password to whatever you'd like to use.

Next, we will move to the postgres configuration directory:

```
# cd /etc/postgresql/9.4/main
```

We will modify the access file with the user we just created:

```
# nano pg_hba.conf
```

At any place **not** at the bottom of the file, add a line to let the new user get access to this server:

```
host replication rep IP_address_of_slave/32 md5  
or  
host replication rep all md5  
IP_address_of_slave/32 >> access from this particular ip address  
  
all >> access from all ip
```

Save and close the file.

```

root@ubuntu-server: /home/ubuntu
root@ubuntu-server: /home/ubuntu
root@ubuntu-client: /home/ubuntu
bizruntime@biz: ~
GNU nano 2.2.6 File: /etc/postgresql/9.3/main/pg_hba.conf Modified

# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all peer
# IPv4 local connections:

host replication rep all md5
host all all 127.0.0.1/32 md5

# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local replication postgres peer
#host replication postgres 127.0.0.1/32 md5
#host replication postgres ::1/128 md5

```

Next, we will open the main postgres configuration file:

nano postgresql.conf

Find these parameters. Uncomment them if they are commented, and modify the values according to what we have listed below:

listen_addresses = ''*
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on

Save and close the file.

```

root@ubuntu-server: /home/ubuntu
root@ubuntu-server: /home/ubuntu
root@ubuntu-client: /home/ubuntu
bizruntime@biz: ~
GNU nano 2.2.6 File: /etc/postgresql/9.3/main/postgresql.conf

listen_addresses = '*'
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on

# -----
# PostgreSQL configuration file
# -----
#
# This file consists of lines of the form:
#
#   name = value
#
# (The "=" is optional.)  Whitespace may be used.  Comments are introduced with
# "#" anywhere on a line.  The complete list of parameter names and allowed
# values can be found in the PostgreSQL documentation.
#
# The commented-out settings shown in this file represent the default values.
Read 599 lines
^G Get Help      ^O WriteOut     ^R Read File    ^V Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit          ^J Justify      ^W Where Is     ^N Next Page    ^U UnCut Text  ^T To Spell

```

Restart the master server to implement your changes:

```
# service postgresql restart
```

Configure the Slave Server

Begin on the slave server by shutting down the postgres database software:

```
# service postgresql stop
```

We will be making some similar configuration changes to postgres files, so change to the configuration directory:

```
# cd /etc/postgresql/9.4/main
```

Adjust the access file to allow the other server to connect to this. This is in case we need to turn the slave into the master later on down the road.

```
# nano pg_hba.conf
```

Again, add this line somewhere not at the end of the file:

```
host replication rep IP_address_of_master/32 md5
```

or

```
host replication rep all md5
```

IP_address_of_slave/32 >> access from this particular ip address

all >> access from all ip

```
root@ubuntu-client: /home/ubuntu
root@ubuntu-server: /home/ubuntu
bizruntime@biz: ~
GNU nano 2.2.6 File: /etc/postgresql/9.3/main/pg_hba.conf Modified
local all postgres peer
# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all peer
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host replication rep all md5
# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local replication postgres peer
#host replication postgres 127.0.0.1/32 md5
#host replication postgres ::1/128 md5
^G Get Help ^O WriteOut ^R Read File ^V Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^N Next Page ^U UnCut Text ^T To Spell
```

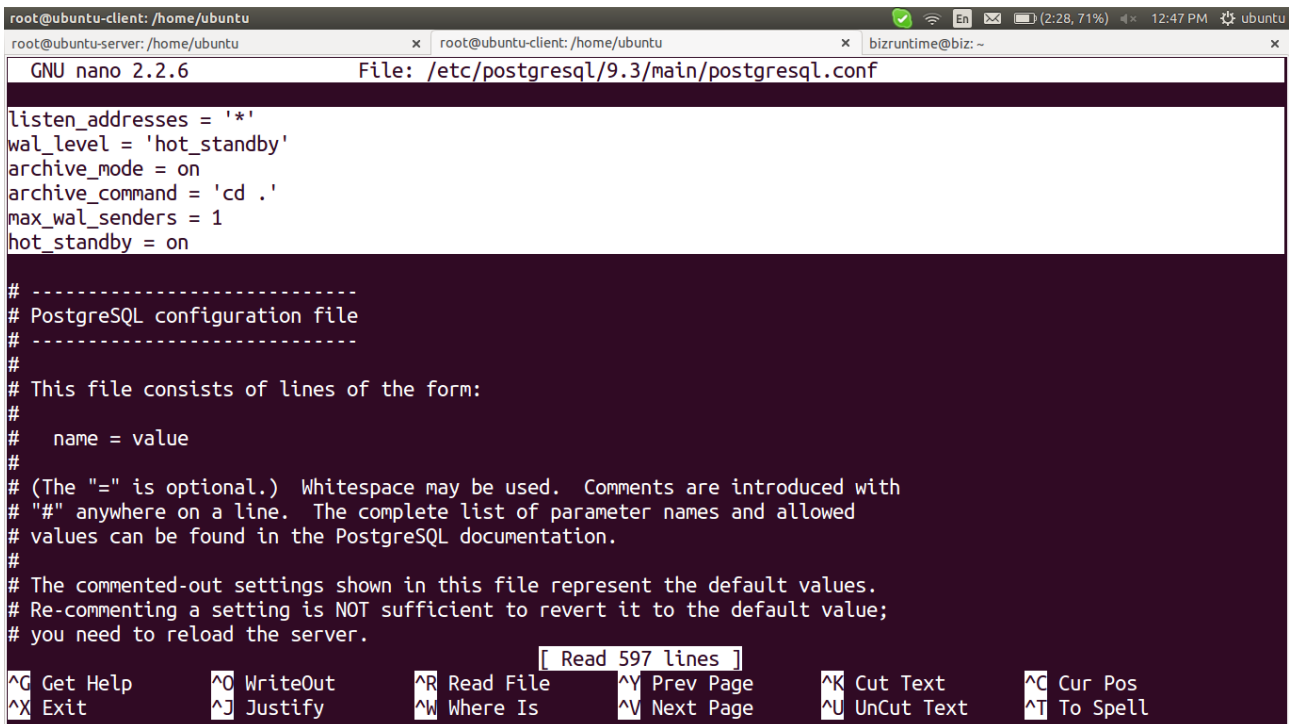
Save and close the file.

Next, open the postgres configuration file:

nano postgresql.conf

You can use the same configuration options you set for the master server, modifying only the IP address to reflect the slave server's address:

listen_addresses = ''*
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on



```
root@ubuntu-client: /home/ubuntu
root@ubuntu-server: /home/ubuntu
bizruntime@biz: ~
GNU nano 2.2.6 File: /etc/postgresql/9.3/main/postgresql.conf

listen_addresses = '*'
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on

# -----
# PostgreSQL configuration file
# -----
#
# This file consists of lines of the form:
#
#   name = value
#
# (The "=" is optional.)  Whitespace may be used.  Comments are introduced with
# "#" anywhere on a line.  The complete list of parameter names and allowed
# values can be found in the PostgreSQL documentation.
#
# The commented-out settings shown in this file represent the default values.
# Re-commenting a setting is NOT sufficient to revert it to the default value;
# you need to reload the server.

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

Save and close the file.

Replicating the Initial database:

Before the slave can replicate the master, we need to give it the initial database to build off of. This is because it reads the logs off of the master server and applies the changes to its own database. We need that database to match the master database.

On the master server, we can use an internal postgres backup start command to create a backup label command. We then will transfer the database data to our slave and then issue an internal backup stop command to clean up:

```
# psql -c "select pg_start_backup('initial_backup');"
# rsync -cva --inplace --exclude=*pg_xlog* /var/lib/postgresql/9.4/main/
slave_IP_address:/var/lib/postgresql/9.4/main/
# psql -c "select pg_stop_backup();"
```

The rsync command may have had an error on modifying the certificate files, but this is okay for our use. The master's data should now be on the slave.

We now have to configure a recovery file on our slave. On the slave navigate to the data directory:

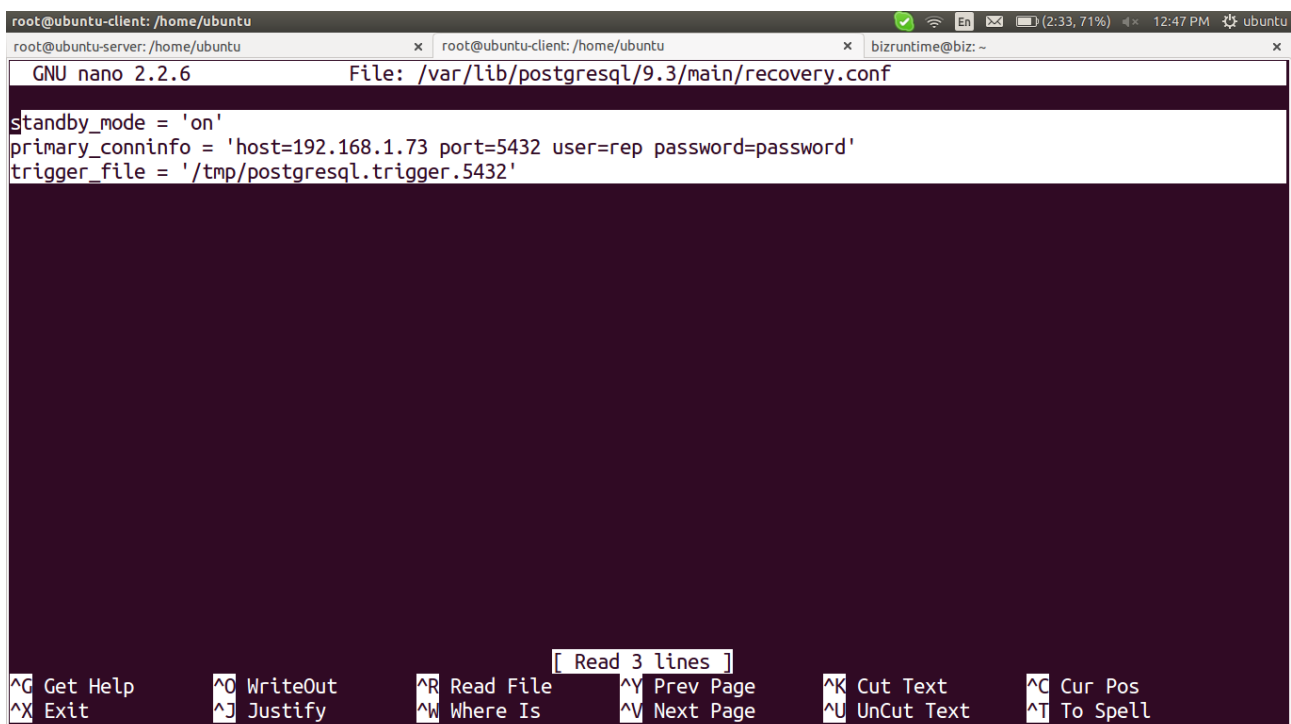
```
# cd /var/lib/postgresql/9.4/main
```

Here, we need to create a recovery file called recovery.conf:

```
# nano recovery.conf
```

Fill in the following information. Make sure to change the IP address of your master server and the password for the rep user you created:

```
standby_mode = 'on'
primary_conninfo = 'host=master_IP_address port=5432 user=rep password=yourpassword'
trigger_file = '/tmp/postgresql.trigger.5432'
```



```
root@ubuntu-client: /home/ubuntu
root@ubuntu-server: /home/ubuntu
root@ubuntu-client: /home/ubuntu
bizruntime@biz: ~
GNU nano 2.2.6 File: /var/lib/postgresql/9.3/main/recovery.conf
standby_mode = 'on'
primary_conninfo = 'host=192.168.1.73 port=5432 user=rep password=password'
trigger_file = '/tmp/postgresql.trigger.5432'
[ Read 3 lines ]
^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is      ^V Next Page     ^U UnCut Text   ^T To Spell
```

The last line in the file, `trigger_file`, is one of the most interesting parts of the entire configuration. If you create a file at that location on your slave machine, your slave will reconfigure itself to act as a master.

This will break your current replication, especially if the master server is still running, but is what you would need to do if your master server goes down. This will allow the slave to begin accepting writes. You can then fix the master server and turn that into the slave.

You should now have the pieces in place to start your slave server. Type:

```
# service postgresql start
```

You'll want to check the logs to see if there are any problems. They are located on both machines here:

```
# less /var/log/postgresql/postgresql-9.4-main.log
```

You should see that it is successfully connecting to the master server.

```
*****
```

Test the Replication

```
*****
```

We will see first-hand if our servers are replicating correctly by making some changes on the master server and then querying the slave.

On the master server, as the postgres user, log into the postgres system by typing:

```
# psql
```

Your prompt will change to indicate that you are now communicating with the database software.

We will create a test table to create some changes:

```
> CREATE TABLE rep_test (test varchar(40));
```

Now, we can insert some values into the table with the following commands:

```
> INSERT INTO rep_test VALUES ('data one');
> INSERT INTO rep_test VALUES ('some more words');
> INSERT INTO rep_test VALUES ('lalala');
> INSERT INTO rep_test VALUES ('hello there');
> INSERT INTO rep_test VALUES ('blahblah');
```

You can now exit out of this interface by typing:

```
> \q
```

Now, on the slave, enter the database interface in the same way:

```
# psql
```

Now, we can see if the data we entered in the master database has been replicated on the slave:

```
> SELECT * FROM rep_test;
```

test

data one
some more words
lalala
hello there
blahblah
(5 rows)

Excellent! Our data has been written to both the master and slave servers.

Let's see if we can insert more data into the table on our slave:

> INSERT INTO rep_test VALUES ('oops');

ERROR: cannot execute INSERT in a read-only transaction

As you can see, we are unable to insert data into the slave. This is because the data is only being transferred in one direction. In order to keep the databases consistent, postgres must make the slave read-only.

Link ::

installation with postgresql 9.3

<https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-on-postgresql-on-an-ubuntu-12-04-vps>

installation with postgresql 9.4

<http://www.dbrnd.com/2016/04/postgresql-how-to-install-postgresql-9-4-on-ubuntu-14-04/>
