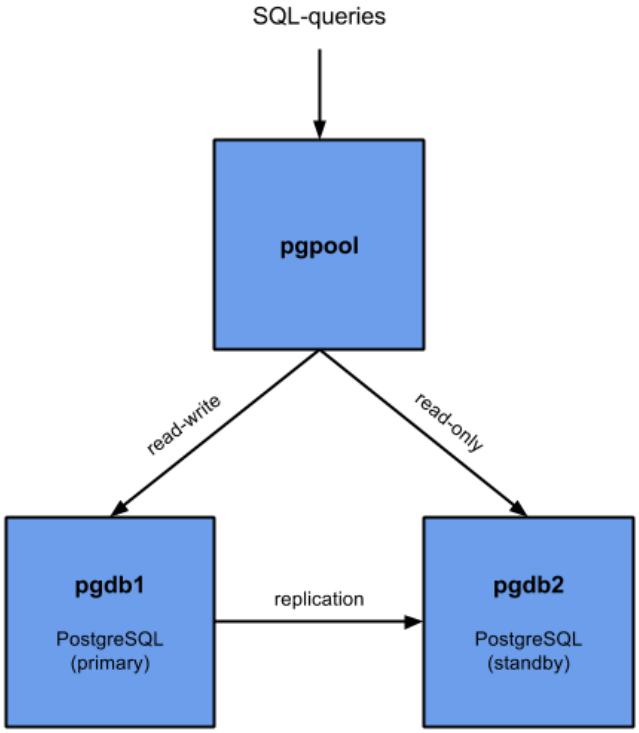
Table of Contents

PostgreSQL Cluster 1
- Setup the Primary PostgreSQL node 1
- Setup the Standby PostgreSQL node 4
- Test Replication 6
- Setup Pgpool-II 10
- Recommendation 13
- Test Failover 17
- Recover after failover to a master-slave situation 19
- Recover to the original situation 22
- Recover to the original situation 23
- Troubleshooting 25

PostgreSQL Cluster

We will setup two nodes as redundant *PostgreSQL DB* (a primary and a standby), and one machine which will do the pooling and distribution. The pool is a single point of failure and ideally should also be redundant (which is why it is installed on the OpenNMS servers). For our case, the most important is that all data stays available in case of a failure.



As you can see in the above scheme, *SQL* queries will be distributed to the primary and standby, based on the type of query. Data written to the master/primary should get replicated to the standby.

In case of a failure of the primary, the standby should replace the primary and become read-write. This will make sure that the database stays available for all applications. In the meanwhile, you can investigate what happened to the old primary and, after a check up, start it as the new standby.

The *PostgreSQL* Servers are going to use *repmgr*, to reduce the complexity of promoting a standby server to be the new primary, and for recovering a failed primary to be a new standby.

Because only one *OpenNMS* server will be running at a time thanks to the *RedHat Clustering Services*, the active *OpenNMS* server will have the role of the *Pgpool-II* server; in other words:

```
pgpool server == opennms server
```

Setup the Primary PostgreSQL node

The first step is to install the *PostgreSQL* database for the master and configure it correctly for replication.

Let's start with initializing *PostgreSQL*:

On *RHEL/CentOS 6*:

```
[root@pgdbsrv01 ~]# service postgresql-9.4 initdb
Initializing database:                                     [  OK  ]
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv01 ~]# /usr/pgsql-9.4/bin/postgresql94-setup initdb
Initializing database ... OK
```

Configure database access by editing `/var/lib/pgsql/9.4/data/pg_hba.conf`:

```
Unresolved directive in postgresql-cluster.adoc - include::../files/postgresql-cluster/pg_hba.conf[]
```

As you can see, md5 has been added to all combinations of user/database/server, to ensure that a password will be required to connect to any database.

Configure *PostgreSQL* itself and streaming replication by editing `/var/lib/pgsql/9.4/data/postgresql.conf`, and updating the following attributes:

```
Unresolved directive in postgresql-cluster.adoc - include::../files/postgresql-cluster/postgresql.conf[]
```

It's a good idea to merge the changed parameters with the existing, default, contents of `postgresql.conf` since it contains a lot of useful comments (annotated). I've just posted the effective settings here to keep it clear.

To get good values for your buffers and memory reservations, you can use this as a source:

http://pgtune.leopard.in.ua/

Here is an example:

```
max_connections = 200
shared_buffers = 512MB
effective_cache_size = 1536MB
work_mem = 2621kB
maintenance_work_mem = 128MB
```

Warning: The above settings should not be used with the *Vagrant VMs*, as these VMs only have 1GB of *RAM*. That was just an example for a physical server with enough *RAM*. Leave the default settings for the *Vagrant VMs*.

IMPORTANT	For *PostgreSQL 9.0* to *9.3*, `max_replication_slots` cannot be used. In case it is required to use an older version, you should replace `max_replication_slots` with: `wal_keep_segments = 5000`

The reason of using *9.4* with replication slots is precisely because the above setting must be big in 9.3 (or older) to avoid issues with replication, but that forces to have a huge filesystem for /var/lib/pgsql. Either way, with replication slots, a standby server must not be down for a long time, otherwise, this might eat all the disk space allocated for `/var/lib/psql`.

By default, the configuration directory for *repmgr* is located at `/etc/repmgr/9.4/`. It is recommended to make a copy of `repmgr.conf` (for documentation purposes), and replace the content of it with the following:

```
Unresolved directive in postgresql-cluster.adoc - include::../files/postgresql-cluster/repmgr.conf[]
```

IMPORTANT For *PostgreSQL 9.0* to *9.3*, `use_replication_slots` must be `0` (i.e. not enabled).

Don't forget to set the owner of the configuration file to *postgres*:

```
[root@pgdbsrv01 ~]$ chown postgres:postgres /etc/repmgr/9.4/repmgr.conf
```

Enable and Start *PostgreSQL* on the master:

On *CentOS/RHEL 6*:

```
[root@pgdbsrv01 ~]# chkconfig postgresql-9.4 on
[root@pgdbsrv01 ~]# service postgresql-9.4 start
Starting postgresql-9.4 service:                           [  OK  ]
```

On *CentOS/RHEL 7*:

```
[root@pgdbsrv01 ~]# systemctl enable postgresql-9.4
ln -s '/usr/lib/systemd/system/postgresql-9.4.service' '/etc/systemd/system/multi-
user.target.wants/postgresql-9.4.service'
[root@pgdbsrv01 ~]# systemctl start postgresql-9.4
```

Create the required users for replication, *repmgr*, *pgpool-II*, *opennms*, and the *repmgr* database:

```
[root@pgdbsrv01 ~]$ sudo su - postgres
-bash-4.2$ psql
psql (9.4.4)
Type "help" for help.

postgres=# CREATE ROLE pgpool SUPERUSER CREATEDB CREATEROLE INHERIT REPLICATION LOGIN ENCRYPTED PASSWORD
'pgpool';
CREATE ROLE
postgres=# CREATE USER repmgr SUPERUSER REPLICATION LOGIN ENCRYPTED PASSWORD 'repmgr';
CREATE ROLE
postgres=# CREATE DATABASE repmgr OWNER repmgr;
CREATE DATABASE
postgres=# CREATE USER opennms SUPERUSER CREATEDB ENCRYPTED PASSWORD 'opennms';
CREATE ROLE
postgres=# ALTER USER postgres WITH ENCRYPTED PASSWORD 'postgres';
ALTER ROLE
postgres=# \q
-bash-4.1$ exit
logout
```

Remember to set appropriate passwords (the word between single quotes after the keyword PASSWORD). For testing purposes, I've used the name of the user for its password to simplify the explanation and configuration examples.

The user *postgres* will be used for replication (as shown in the `pg_hba.conf` file), and for creating the *OpenNMS* database, so it requires a password.

In case you want to have a different user for replication, besides creating the user on the database as shown before, the

`pg_hba.conf` file must be properly updated, as well as the `.pgpass` (more on this below).

Because *md5* has been configured on `pg_hba.conf`, it is important to create a `.pgpass` for the *postgres* user, with the password for the *repmgr* user in the database for all databases, and the password for the user created for replication in the database (i.e. the *postgres* user for this particular deployment):

```
[root@pgdbsrv01 ~]$ su - postgres
-bash-4.2$ echo "*:*:*:repmgr:repmgr" > ~/.pgpass
-bash-4.2$ echo "*:*:replication:postgres:postgres" >> ~/.pgpass
-bash-4.2$ chmod 600 ~/.pgpass
-bash-4.2$ cat ~/.pgpass
*:*:repmgr:repmgr:repmgr
-bash-4.1$ exit
logout
[root@pgdbsrv01 ~]$ restorecon -R /var/lib/pgsql/
```

The last 3 keys of each `.pgpass` entry are respectively: the database name, the user and the password, so remember to use the appropriate users and passwords. The permissions on this file must be `0600` and it is important to update the *SELinux* attributes.

IMPORTANT	If you forgot to execute the `restorecon` command and *SELinux* is enforcing, the `.pgpass` file wont work in some cases. Of course, this is not the case if *SELinux* is permissive or disabled.

Refer to the following link for more details about `.pgpass` files:

http://www.postgresql.org/docs/9.4/static/libpq-pgpass.html

The last step is to register *pgdbsrv01* as master node for *repmgr*:

```
[root@pgdbsrv01 ~]$ sudo su - postgres
-bash-4.1$ /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf --verbose master register
[2015-07-13 10:13:09] [NOTICE] opening configuration file: /etc/repmgr/9.4/repmgr.conf
[2015-07-13 10:13:09] [INFO] connecting to master database
[2015-07-13 10:13:09] [INFO] connected to master, checking its state
[2015-07-13 10:13:09] [INFO] master register: creating database objects inside the repmgr_opennms_cluster
schema
[2015-07-13 10:13:09] [NOTICE] master node correctly registered for cluster opennms_cluster with id 1
(conninfo: host=pgdbsrv01 user=repmgr dbname=repmgr)
```

If the `.pgpass` or the *DNS* is not well configured, the above command could fail or request a password for the *repmgr* user.

Setup the Standby PostgreSQL node

Setting up the standby node doesn't require a lot of configuration. On the first sync with the primary, most of the configuration is taken from here.

It is important to copy the `.pgpass` file from the primary server:

```
[root@pgdbsrv02 ~]# su - postgres
-bash-4.1$ scp pgdbsrv01:~/.pgpass ~/.pgpass
.pgpass        100%   54     0.1KB/s   00:00
-bash-4.1$ exit
logout
[root@pgdbsrv02 ~]$ restorecon -R /var/lib/pgsql/
```

Configure *repmgr* similar to the primary by creating `/etc/repmgr/9.4/repmgr.conf` with the following content:

```
cluster=opennms_cluster
node=2
node_name=pgdbsrv02
conninfo='host=pgdbsrv02 user=repmgr dbname=repmgr'
use_replication_slots=1
loglevel=INFO
pg_bindir=/usr/pgsql-9.4/bin/
pg_basebackup_options='--xlog-method=stream'
master_response_timeout=30
reconnect_attempts=3
reconnect_interval=10
failover=manual
promote_command='/usr/pgsql-9.4/bin/repmgr standby promote -f /etc/repmgr/9.4/repmgr.conf'
follow_command='/usr/pgsql-9.4/bin/repmgr standby follow -f /etc/repmgr/9.4/repmgr.conf'
```

IMPORTANT For *PostgreSQL 9.0* to *9.3*, `use_replication_slots` must be `0` (i.e. not enabled).

Also notice that the first name (the *cluster ID*) must be the same on all members of the cluster. Only the lines 2 to 4 are going to be different on each cluster node.

Don't forget to set the owner of the configuration file to *postgres*:

```
[root@pgdbsrv02 ~]$ chown postgres:postgres /etc/repmgr/9.4/repmgr.conf
```

Sync the configuration and contents of the primary with the standby:

```
[root@pgdbsrv02 ~]$ sudo su - postgres
-bash-4.1$ /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf --verbose -D /var/lib/pgsql/9.4/data
-d repmgr -p 5432 -U repmgr -R postgres standby clone pgdbsrv01
[2015-07-13 10:46:54] [NOTICE] opening configuration file: /etc/repmgr/9.4/repmgr.conf
[2015-07-13 10:46:54] [NOTICE] destination directory '/var/lib/pgsql/9.4/data' provided
[2015-07-13 10:46:54] [INFO] connecting to upstream node
[2015-07-13 10:46:54] [INFO] connected to upstream node, checking its state
[2015-07-13 10:46:54] [INFO] Successfully connected to upstream node. Current installation size is 26 MB
[2015-07-13 10:46:54] [NOTICE] starting backup...
[2015-07-13 10:46:54] [INFO] checking and correcting permissions on existing directory
/var/lib/pgsql/9.4/data ...
[2015-07-13 10:46:54] [INFO] executing: '/usr/pgsql-9.4/bin/pg_basebackup -l "repmgr base backup"  -h
pgdbsrv01 -p 5432 -U repmgr -D /var/lib/pgsql/9.4/data --xlog-method=stream'
[2015-07-13 10:46:59] [NOTICE] standby clone (using pg_basebackup) complete
[2015-07-13 10:46:59] [NOTICE] HINT: you can now start your PostgreSQL server
[2015-07-13 10:46:59] [NOTICE] for example : pg_ctl -D /var/lib/pgsql/9.4/data start
-bash-4.1$ exit
logout
```

IMPORTANT	If the command execution ask for the password configured for the *repmgr* user, you might forgot to execute the `restorecon` command, to avoid *SELinux* issues when it is enforcing.

In case you experience problems with the synchronization and you need to repeat this step, first delete the contents of the *PostgreSQL* datadir (`rm -rf /var/lib/pgsql/9.4/data/*`) and try again.

On *RHEL/CentOS 6*:

```
[root@pgdbsrv02 ~]# chkconfig postgresql-9.4 on
[root@pgdbsrv02 ~]# service postgresql-9.4 start
Starting postgresql-9.4 service:                           [  OK  ]
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv02 ~]# systemctl enable postgresql-9.4
ln -s '/usr/lib/systemd/system/postgresql-9.4.service' '/etc/systemd/system/multi-
user.target.wants/postgresql-9.4.service'
[root@pgdbsrv02 ~]# systemctl start postgresql-9.4
```

As the last step in the standby node setup, register the standby in *repmgr*:

```
[root@pgdbsrv02 ~]$ su - postgres
-bash-4.1$ /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf --verbose standby register
[2015-07-13 10:49:24] [NOTICE] opening configuration file: /etc/repmgr/9.4/repmgr.conf
[2015-07-13 10:49:24] [INFO] connecting to standby database
[2015-07-13 10:49:24] [INFO] connecting to master database
[2015-07-13 10:49:24] [INFO] finding node list for cluster 'opennms_cluster'
[2015-07-13 10:49:24] [INFO] checking role of cluster node '1'
[2015-07-13 10:49:24] [INFO] registering the standby
[2015-07-13 10:49:24] [INFO] standby registration complete
[2015-07-13 10:49:24] [NOTICE] standby node correctly registered for cluster opennms_cluster with id 2
(conninfo: host=pgdbsrv02 user=repmgr dbname=repmgr)
-bash-4.1$ exit
logout
```

Test Replication

First, let's check that all the servers are running the proper role. The primary/master server should not have a `recovery.con` file, but that should exist on each standby server. This file is automatically generated by *repmgr* as part of the execution of the `standby clone` command.

Another way to verify which server is the master, execute the following *SQL* query:

```
SELECT pg_is_in_recovery();
```

If the value is true, you're on a slave (or standby server); if false, you're on the master/primary server:

On *pgdbsrv01*:

```
[root@pgdbsrv01 ~]# su - postgres
-bash-4.1$ psql
psql (9.4.4)
Type "help" for help.

postgres=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 f
(1 row)

postgres=# \q
```

Or,

```
[root@pgdbsrv01 ~]# su - postgres -c 'psql -c "SELECT pg_is_in_recovery();"'
 pg_is_in_recovery
-------------------
 f
(1 row)
```

On *pgdbsrv02*:

```
[root@pgdbsrv02 ~]# su - postgres
-bash-4.1$ psql
psql (9.4.4)
Type "help" for help.

postgres=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 row)

postgres=# \q
```

You can also check the *repmgr* tables to verify if the nodes have been successfully registered:

```
|---
[root@pgdbsrv01 ~]# su - postgres
-bash-4.1$ psql -x -U repmgr -h pgdbsrv01 -d repmgr -c "SELECT * FROM repmgr_opennms_cluster.repl_nodes;"
-[ RECORD 1 ]----+----------------------------------------
id               | 1
type             | master
upstream_node_id |
cluster          | opennms_cluster
name             | pgdbsrv01
conninfo         | host=pgdbsrv01 user=repmgr dbname=repmgr
slot_name        | repmgr_slot_1
priority         | 100
active           | t
-[ RECORD 2 ]----+----------------------------------------
id               | 2
type             | standby
upstream_node_id | 1
cluster          | opennms_cluster
name             | pgdbsrv02
conninfo         | host=pgdbsrv02 user=repmgr dbname=repmgr
slot_name        | repmgr_slot_2
priority         | 100
active           | t
|---
```

NOTE that the query has been executed against the primary server.

It is important to have in mind that the name of the view starts with the word "repmgr", then the underscore character ("_"), and then the name of the cluster (in this case, `opennms_cluster`, as it was defined on `repmgr.conf`); in other words: `repmgr_opennms_cluster`. This is the main reason why you should never use special characters when choosing the name of the cluster.

Another way to have a quick status of the cluster is:

```
[root@pgdbsrv01 ~]# su - postgres -c "/usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf cluster
show"
[2015-07-13 10:56:35] [INFO] connecting to database
Role      | Connection String
* master  | host=pgdbsrv01 user=repmgr dbname=repmgr
  standby | host=pgdbsrv02 user=repmgr dbname=repmgr
```

The above command can be executed on any *PostgreSQL* server.

Finally, you can use the following command on the primary server to see the statistics from the replication.

```
|---
[root@pgdbsrv01 ~]# su - postgres -c "psql -x -c 'select * from pg_stat_replication;'"
-[ RECORD 1 ]----+------------------------------
pid              | 3391
usesysid         | 16385
usename          | repmgr
application_name | pgdbsrv02
client_addr      | 192.168.205.154
client_hostname  | pgdbsrv02.local
client_port      | 49663
backend_start    | 2015-07-13 11:34:19.452167-04
backend_xmin     |
state            | streaming
sent_location    | 0/C001B78
write_location   | 0/C001B78
flush_location   | 0/C001B78
replay_location  | 0/C001B78
sync_priority    | 0
sync_state       | async
|---
```

Also you should see that the replication processes are running on the standby server:

```
[root@pgdbsrv02 ~]# ps aux | egrep 'wal\sreceiver'
postgres   3377  0.0  0.3 345876  3176 ?        Ss   11:34   0:01 postgres: wal receiver process
streaming 0/C001B78
```

If the output of the above commands is empty, check the credentials on the `.pgpass` file, the *DNS*, the connection between the database servers, the internal firewall, and `pg_hba.conf`.

At this point, updates on the master should be replicated to the slave so it's a good time to see if that really works as we would expect by creating a new database on the primary node.

Before we create a sample database on the primary, let's list the databases from the standby-node:

```
|---
[root@pgdbsrv01 ~]# su - postgres -c "psql -c '\list'"
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 repmgr    | repmgr   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(4 rows)
```

Create the database on the primary:

[root@pgdbsrv01 ~]# su - postgres -c "psql -c 'CREATE DATABASE test'" CREATE DATABASE |---

Same command as we did before the *DB-create*: list the current databases on the standby:

```
|---
[root@pgdbsrv02 ~]# su - postgres -c "psql -c '\list'"
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 repmgr    | repmgr   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 test      | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(5 rows)
|---
```

As you can see, the new database, test, got replicated to the standby.

Another test which you could do, is to check if the standby is read-only:

```
[root@pgdbsrv02 ~]# su - postgres -c "psql -c 'CREATE DATABASE test2'"
ERROR:  cannot execute CREATE DATABASE in a read-only transaction
```

The above result is good, since we don't want somebody to update our standby, everything should pass via the primary and get replicated to the standby.

Finally, remove the test database:

```
[root@pgdbsrv01 ~]# su - postgres -c "psql -c 'DROP DATABASE test'"
DROP DATABASE
```

If you check the standby server again, you should see that the test *DB*: is not there, as expected.

Setup Pgpool-II

The next chapter is to setup *pgpool-II* in order to distribute queries to the primary and standby, and to make sure that we failover to the standby in case the primary isn't available anymore.

Configure *pgpool-II* by copying the sample configuration file:

```
[root@onmssrv01 ~]$ cd /etc/pgpool-II/
[root@onmssrv01 pgpool-II]$ cp pgpool.conf.sample-stream pgpool.conf
```

Edit the following settings in the copied sample file `/etc/pgpool-II/pgpool.conf`, do not replace the contents of the file but edit everything listed under here:

```
listen_addresses = '*'
port = 9999
backend_hostname0 = 'pgdbsrv01'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/var/lib/pgsql/9.4/data'
backend_flag0 = 'ALLOW_TO_FAILOVER'
backend_hostname1 = 'pgdbsrv02'
backend_port1 = 5432
backend_weight1 = 1
backend_data_directory1 = '/var/lib/pgsql/9.4/data'
backend_flag1 = 'ALLOW_TO_FAILOVER'
enable_pool_hba = on
pool_passwd = 'pool_passwd'
authentication_timeout = 60
num_init_children = 60
max_pool = 1
child_max_connections = 0
pid_file_name = '/var/run/pgpool/pgpool.pid'
logdir = '/var/log/pgpool'
connection_cache = on
replication_mode = off
load_balance_mode = on
master_slave_mode = on
master_slave_sub_mode = 'stream'
sr_check_user = 'pgpool'
sr_check_password = 'pgpool'
health_check_period = 10
health_check_user = 'pgpool'
health_check_password = 'pgpool'
failover_command = '/etc/pgpool-II/failover.sh %h %H'
recovery_user = 'pgpool'
recovery_password = 'pgpool'
```

IMPORTANT	Replace the sample password of the *pgpool* user accordingly (not required for the *Vagrant lab*).
IMPORTANT	For *RHEL/CentOS 6*, the `pid_file_name` must be `/var/run/pgpool/pgpool.pid`. For *RHEL/CentOS 7* this is enforced through `pgpool.service` (the *systemd* initialization script). That way, the content of `pgpool.conf` is the same for both operating systems.

Having the password on a configuration file is not optimal, but unfortunately, this is something that is still required for the *pgpool-II* version installed on the systems (which is the latest one by the time this document was written).

Also, be sure that `num_init_children` is consistent with the required number of connections in *OpenNMS*, and the configured `max_connections` in `postgresql.conf`.

To reduce possible security holes, change the permissions of the file to be `0600`, and make the user *postgres* to own the file:

```
[root@onmssrv01 ~]$ chmod 0600 /etc/pgpool-II/pgpool.conf
[root@onmssrv01 ~]$ chown postgres:postgres /etc/pgpool-II/pgpool.conf
```

The selected port is the default (i.e. 9999). It is possible to change it to *PostgreSQL's* default (i.e. 5432), but I've chosen 9999 to remember that we're connecting to *pgpool-II* and not directly to the *PostgreSQL* servers.

The online recovery feature won't be used. It is recommended that the server administrator manually execute the recovery procedure on the broken *DB* server, to convert it into a new slave server.

On the above settings, we specified that on failover, the script `failover.sh` should be executed, so we'll need to create this file in `/etc/pgpool-II/failover.sh`:

```
Unresolved directive in postgresql-cluster.adoc - include::../files/postgresql-cluster/failover.sh[]
```

Remember to fix the permissions on the failover script:

```
[root@onmssrv01 ~]$ chmod 0700 /etc/pgpool-II/failover.sh
[root@onmssrv01 ~]$ chown postgres:postgres /etc/pgpool-II/failover.sh
```

We also specified in the configuration that we want to use `pool_hba.conf`, so we need to add the credentials to the `/etc/pgpool-II/pool_hba.conf` file.

Due to a known problem in *pgpool-II*, it is not possible to mix authentication methods on `pool_hba.conf`. For this reason, and because *md5* has been configured on all the *PostgreSQL* servers, the content of the `pool_hba.conf` file must be:

```
# "local" is for Unix domain socket connections only
local   all         all                               md5
# IPv4 local connections:
host    all         all         127.0.0.1/32          md5
host    all         all         ::1/128               md5
host    all         all         0.0.0.0/0             md5
```

Remember to fix the permissions on the failover script:

```
[root@onmssrv01 ~]$ chmod 0600 /etc/pgpool-II/pool_hba.conf
[root@onmssrv01 ~]$ chown postgres:postgres /etc/pgpool-II/pool_hba.conf
```

Every user that needs to connect via *pgpool-II* needs to be added in `pool_passwd`. First we need to create this file and let it be owned by *postgres*:

```
[root@onmssrv01 ~]# touch /etc/pgpool-II/pool_passwd
[root@onmssrv01 ~]# chmod 600 /etc/pgpool-II/pool_passwd
[root@onmssrv01 ~]# chown postgres:postgres /etc/pgpool-II/pool_passwd
```

Next, we can add users to the file. Let's do that for the *pgpool* user which we created earlier:

```
[root@onmssrv01 ~]# su - postgres -c "pg_md5 -m -u pgpool pgpool"
```

You'll have to repeat this step for every user that needs access to the *DB* (including the replication user if you have one), in other words:

```
[root@onmssrv01 ~]# su - postgres -c "pg_md5 -m -u repmgr repmgr"
[root@onmssrv01 ~]# su - postgres -c "pg_md5 -m -u opennms opennms"
[root@onmssrv01 ~]# su - postgres -c "pg_md5 -m -u postgres postgres"
```

The passwords will be stored as *MD5* strings, and must be the same passwords configured on the databases.

IMPORTANT	I've used the sample passwords for the *Vagrant lab*, so remember to use the appropriate passwords.

Last step is to allow connection via *PCP* to `manage postgres`. This requires a similar approach as with `pool_passwd`. For simplicity, I've chosen the *postgres* user (with the same password). It can be a brand new user if required. This will be the user required to execute *PCP* commands.

Generate the *MD5* form for the chosen password (i.e. the word `postgres`):

```
[root@onmssrv01 ~]# pg_md5 postgres
e8a48653851e28c69d0506508fb27fc5
```

Now, add the user with the above password to `pcp.conf`:

```
[root@onmssrv01 ~]# echo "postgres:e8a48653851e28c69d0506508fb27fc5" >> /etc/pgpool-II/pcp.conf
```

Remember to fix the permissions on the failover script:

```
[root@onmssrv01 ~]$ chmod 0600 /etc/pgpool-II/pcp.conf
[root@onmssrv01 ~]$ chown postgres:postgres /etc/pgpool-II/pcp.conf
```

At the end, you should have something like the following at `/etc/pgpool-II/`:

```
 -rwx------. 1 postgres postgres   278 Jul 13 12:11 failover.sh
 -rw-------. 1 postgres postgres   901 Jul 13 12:20 pcp.conf
 -rw-------. 1 postgres postgres 32939 Jul 13 12:08 pgpool.conf
 -rw-------. 1 postgres postgres  3312 Jul 13 12:15 pool_hba.conf
 -rw-------. 1 postgres postgres   175 Jul 13 12:16 pool_passwd
```

You can either backup, move, delete or leave untouched the sample files.

Make sure the log directories for *pgpool-II* exist and have proper permissions:

```
[root@onmssrv01 ~]$ mkdir /var/log/pgpool
[root@onmssrv01 ~]$ chown postgres:postgres /var/log/pgpool
[root@onmssrv02 ~]$ mkdir /var/log/pgpool
[root@onmssrv02 ~]$ chown postgres:postgres /var/log/pgpool
```

Once you have all the files properly configured, use *rsync* to copy over all the files to the other *Pgpool-II* server (i.e. *onmssrv02*):

```
[root@onmssrv01 ~]# rsync -avr --delete /etc/pgpool-II/ onmssrv02:/etc/pgpool-II/
```

The *pgpool-II* service is ready to be started.

Recommendation

The *pgpool-II* configuration files are common to both *OpenNMS* servers. For this reason, they must be stored on a shared server and expose them through *NFS* (or a similar technology, as mentioned before). It is important to know that there are

some configuration changes on the *NFS* server that are strictly required in order to respect the ownership of the files when it is not root.

As described on an earlier section, on our current testing deployment, the domain of all the servers is `local`. For this reason, the `/etc/idmapd.conf` file on all the machines (including the *NFS* server) must be updated accordingly. In this case:

```
[General]
#Verbosity = 0
# The following should be set to the local NFSv4 domain name
# The default is the host's DNS domain name.
Domain = local
[source, bash]
```

It might be possible that other changes are required in order to make it work on certain environments. Also the *postgres* user must be created on the *NFS* server (more on this later).

If this is not configured properly, the *pgpool-II* files will appear owned by the nobody user which is not correct, and will prevent the proper work of pgpool-II. When this happens, you should see an error similar to the following on the logs:

```
Jul  7 22:26:39 onmssrv01 nfsidmap[102421]: nss_getpwnam: name 'nobody' does not map into domain 'local'
```

The current workaround is to make the changes on one server and then *rsync* the directory to the other server, as explained before.

Test Pgpool-II

If all went well, at this point, you should have a working installation of *pgpool-II* that will execute queries on the replicated *PostgreSQL* nodes.

First, verify with the `psql` command that you can reach the databases using the *pgpool-II* user:

```
[root@onmssrv01 ~]# psql -U pgpool -h pgdbsrv01 -d postgres
Password for user pgpool:
psql (9.4.4)
Type "help" for help.

postgres=# \q
[root@onmssrv01 ~]# psql -U pgpool -h pgdbsrv02 -d postgres
Password for user pgpool:
psql (9.4.4)
Type "help" for help.

postgres=# \q
```

If you can't connect to the databases, check *iptables*, the user credentials and `pg_hba.conf` on the database servers and the *pgpool-II* configuration files.

Remember to test the connection from the other *OpenNMS* server (i.e. *onmssrv02*).

In order to test *pgpool-II*, we need to start the service. Execute the following commands from one *OpenNMS* server only, as the *pgpool-II* should run on one machine.

On *RHEL/CentOS 6*:

```
[root@onmssrv01 ~]# service pgpool start
```

On *RHEL/CentOS 7*:

```
[root@onmssrv01 ~]# systemctl start pgpool
```

Check the logs to verify *pgpool-II* was started correctly:

On *RHEL/CentOS 6*:

```
[root@onmssrv01 ~]# cat /var/log/pgpool.log
2015-07-13 12:33:05: pid 2145: LOG:  Backend status file /var/log/pgpool_status does not exist
2015-07-13 12:33:05: pid 2145: LOG:  Setting up socket for 0.0.0.0:9999
2015-07-13 12:33:05: pid 2145: LOG:  Setting up socket for :::9999
2015-07-13 12:33:05: pid 2145: LOG:  pgpool-II successfully started. version 3.4.2 (tataraboshi)
2015-07-13 12:33:05: pid 2145: LOG:  find_primary_node: checking backend no 0
2015-07-13 12:33:05: pid 2145: LOG:  find_primary_node: primary node id is 0
```

On *RHEL/CentOS 7*:

```
[root@onmssrv01 ~]# systemctl status pgpool
pgpool.service - pgpool-II
   Loaded: loaded (/usr/lib/systemd/system/pgpool.service; disabled)
   Active: active (running) since Wed 2015-07-29 18:38:18 UTC; 2min 41s ago
 Main PID: 18369 (pgpool)
   CGroup: /system.slice/pgpool.service
           ├─18369 /usr/bin/pgpool -f /etc/pgpool-II/pgpool.conf -n
           ├─18437 pgpool: wait for connection request
           ├─18438 pgpool: wait for connection request

           ├─18722 pgpool: PCP: wait for connection request
           └─18723 pgpool: worker process

Jul 29 18:55:02 onmssrv01.local systemd[1]: Starting pgpool-II...
Jul 29 18:55:02 onmssrv01.local systemd[1]: Started pgpool-II.
Jul 29 18:55:02 onmssrv01.local pgpool[18661]: 2015-07-29 18:55:02: pid 18661: LOG:  reading status file:
1 th backend is set to down status
Jul 29 18:55:02 onmssrv01.local pgpool[18661]: 2015-07-29 18:55:02: pid 18661: LOG:  Setting up socket for
0.0.0.0:9999
Jul 29 18:55:02 onmssrv01.local pgpool[18661]: 2015-07-29 18:55:02: pid 18661: LOG:  Setting up socket for
:::9999
Jul 29 18:55:02 onmssrv01.local pgpool[18661]: 2015-07-29 18:55:02: pid 18661: LOG:  pgpool-II
successfully started. version 3.4.3 (tataraboshi)
Jul 29 18:55:02 onmssrv01.local pgpool[18661]: 2015-07-29 18:55:02: pid 18661: LOG:  find_primary_node:
checking backend no 0
Jul 29 18:55:02 onmssrv01.local pgpool[18661]: 2015-07-29 18:55:02: pid 18661: LOG:  find_primary_node:
primary node id is 0
```

Or,

```
[root@onmssrv01 ~]# journalctl | grep pgpool | tail
Jul 29 18:58:41 onmssrv01.local systemd[1]: Stopped pgpool-II.
Jul 29 18:58:46 onmssrv01.local systemd[1]: Starting pgpool-II...
Jul 29 18:58:46 onmssrv01.local systemd[1]: Started pgpool-II.
Jul 29 18:58:46 onmssrv01.local pgpool[18740]: 2015-07-29 18:58:46: pid 18740: LOG:  Setting up socket for
0.0.0.0:9999
Jul 29 18:58:46 onmssrv01.local pgpool[18740]: 2015-07-29 18:58:46: pid 18740: LOG:  Setting up socket for
:::9999
Jul 29 18:58:46 onmssrv01.local pgpool[18740]: 2015-07-29 18:58:46: pid 18740: LOG:  pgpool-II
successfully started. version 3.4.3 (tataraboshi)
Jul 29 18:58:46 onmssrv01.local pgpool[18740]: 2015-07-29 18:58:46: pid 18740: LOG:  find_primary_node:
checking backend no 0
Jul 29 18:58:46 onmssrv01.local pgpool[18740]: 2015-07-29 18:58:46: pid 18740: LOG:  find_primary_node:
primary node id is 0
```

Let's test if everything is working by executing a query on the databases via *pgpool-II*:

```
|---
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "\l"
Password for user pgpool:
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 repmgr    | repmgr   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(4 rows)
|---
```

This should also work from another host that has network access to the *pgpool-II* machine.

Pgpool-II has a range of `show`-commands available that allow you to get the status of *pgpool-II* itself. They're executed as normal *SQL* but will be intercepted by *pgpool-II*.

To get the status of the nodes connected to *pgpool-II*:

```
|---
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "show pool_nodes"
Password for user pgpool:
 node_id | hostname  | port | status | lb_weight |  role
---------+-----------+------+--------+-----------+---------
 0       | pgdbsrv01 | 5432 | 2      | 0.500000  | primary
 1       | pgdbsrv02 | 5432 | 2      | 0.500000  | standby
(2 rows)
|---
```

If for some reason, the status is not 2 or you see problems on the logs, double check the configuration of pgpool in pgpool.conf (specially the section related with the *PostgreSQL* servers). It might be possible that you must re-attach the failed server:

```
[root@onmssrv01 ~]# pcp_attach_node 0 localhost 9898 postgres postgres X
```

Replace X with the `node_id` from the `show pool_nodes` command.

Finally, stop the *pgpool-II* service and repeat the connection test from *onmssrv02*.

On *RHEL/CentOS 6*:

```
[root@onmssrv01 ~]# service pgpool stop
```

On *RHEL/CentOS 7*:

```
[root@onmssrv01 ~]# systemctl stop pgpool
```

Test Failover

In order to test the failover with *pgpool-II*, the application must be running on one of the *OpenNMS* servers; so make sure it is running before continue.

We can easily test the failover scenario by bringing down the primary database node and see what happens:

On *RHEL/CentOS 6*:

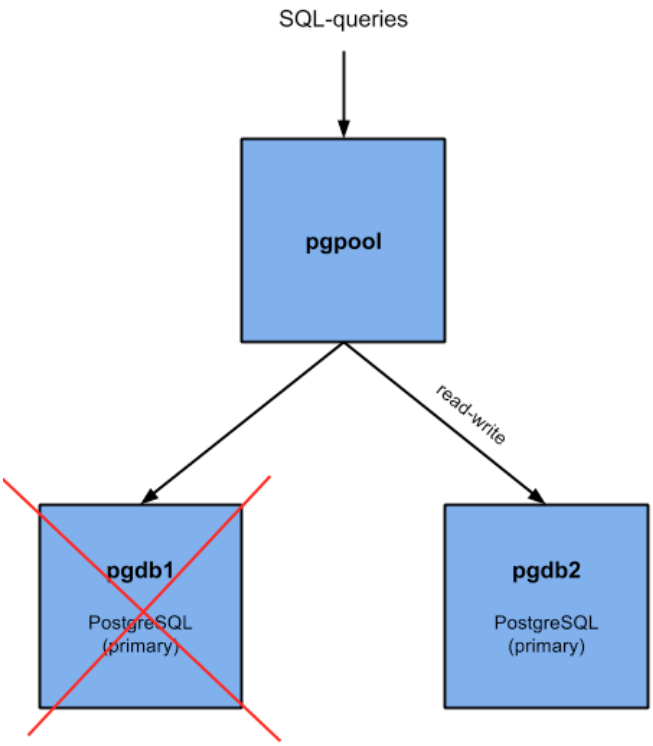
```
[root@pgdbsrv01 ~]# service postgresql-9.4 stop
```

On RHEL/CentOS 7:

```
[root@pgdbsrv01 ~]# systemctl stop postgresql-9.4
```

That will stop the *PostgreSQL* on the primary/master server. That means, *pgpool-II* should automatically execute the `failover.sh` script, that will promote the standby server as the new master.

In other words, our setup becomes like this:



Result on *pgpool-II*:

```
|---
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "show pool_nodes"
Password for user pgpool:
 node_id | hostname  | port | status | lb_weight |  role
---------+-----------+------+--------+-----------+---------
 0       | pgdbsrv01 | 5432 | 3      | 0.500000  | standby
 1       | pgdbsrv02 | 5432 | 2      | 0.500000  | primary
(2 rows)
|---
```

You can see that the former slave (*pgdbsrv02*) has now became the primary, and the former primary (*pgdbsrv01*) has now become a standby but has an offline status (*3*).

Result in *repmgr*:

```
[root@pgdbsrv02 ~]# su - postgres -c " /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf cluster
show"
[2015-07-13 13:58:27] [INFO] connecting to database
Role      | Connection String
[2015-07-13 13:58:27] [ERROR] connection to database failed: could not connect to server: Connection
refused
Is the server running on host "pgdbsrv01" (192.168.205.163) and accepting
TCP/IP connections on port 5432?

  FAILED  | host=pgdbsrv01 user=repmgr dbname=repmgr
* master  | host=pgdbsrv02 user=repmgr dbname=repmgr
```

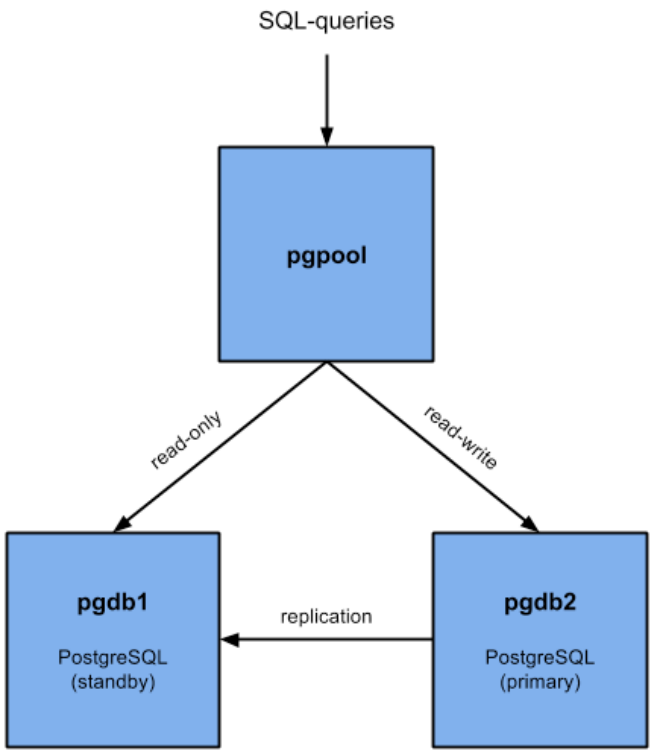
The failover script should add some useful information to `/var/log/pgpool/pgpool_failover.log`, when a failover happens:

```
[root@onmssrv01 ~]# cat /var/log/pgpool/pgpool_failover.log
Wed Jul 29 19:02:49 UTC 2015
Failed node: pgdbsrv01, Promoting pgdbsrv02 ...
+ /usr/bin/ssh -T -l postgres pgdbsrv02 '/usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf standby
promote 2>/dev/null 1>/dev/null <&-'
+ exit 0
```

As you can see, the database stays available via *pgpool-II* and the former standby became read-write.

Recover after failover to a master-slave situation

After the failover, you end up in a working, non-redundant situation. Everything keeps working for users of the database but it would be good to go back to a primary-standby configuration. The goal is to get to this setup:



The first step in this process is to investigate why the primary DB became unavailable and resolve that problem. Once you're sure that the host (*pgdbsrv01* in our example) is healthy again, we can start using that host as the new standby-server.

First, make sure that the database on *pgdbsrv01* isn't started:

On *RHEL/CentOS 6*:

```
[root@pgdbsrv01 ~]# service postgresql-9.4 stop
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv01 ~]# systemctl stop postgresql-9.4
```

Then, we'll sync the data of *pgdbsrv02* (the current primary) to the previously failed node (*pgdbsrv01*). In the meanwhile, the database has got some updates and changes so we need to make sure that those get replicated to *pgdbsrv011* before we

start it as a standby.

There are two way to perform the sync operation:

1. Perform a full synchronization, in other words, recreate the whole `/var/lib/pgsql/9.4/data` directory from scratch:

```
[root@pgdbsrv01 ~]# su - postgres
-bash-4.1$ rm -rf 9.4/data/
-bash-4.1$ /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf --verbose -D /var/lib/pgsql/9.4/data
-d repmgr -p 5432 -U repmgr -R postgres standby clone pgdbsrv02
[2015-07-13 17:26:19] [NOTICE] opening configuration file: /etc/repmgr/9.4/repmgr.conf
[2015-07-13 17:26:19] [NOTICE] destination directory '/var/lib/pgsql/9.4/data' provided
[2015-07-13 17:26:19] [INFO] connecting to upstream node
[2015-07-13 17:26:19] [INFO] connected to upstream node, checking its state
[2015-07-13 17:26:19] [INFO] Successfully connected to upstream node. Current installation size is 39 MB
[2015-07-13 17:26:19] [NOTICE] starting backup...
[2015-07-13 17:26:19] [INFO] creating directory "/var/lib/pgsql/9.4/data"...
[2015-07-13 17:26:19] [INFO] executing: '/usr/pgsql-9.4/bin/pg_basebackup -l "repmgr base backup"  -h
pgdbsrv02 -p 5432 -U repmgr -D /var/lib/pgsql/9.4/data --xlog-method=stream'
[2015-07-13 17:26:22] [NOTICE] standby clone (using pg_basebackup) complete
[2015-07-13 17:26:22] [NOTICE] HINT: you can now start your PostgreSQL server
[2015-07-13 17:26:22] [NOTICE] for example : pg_ctl -D /var/lib/pgsql/9.4/data start
```

IMPORTANT if the command execution ask for the password configured for the repmgr user, execute the restorecon command to avoid SELinux issues when it is enforcing.

1. Perform a partial synchronization which can be a lot faster than the first procedure, to synchronize only the things that have been changed during the outage on the DB server, which assumes the data directory exist and it is not corrupted. If you're not sure about the state of the data directory, use the first procedure:

```
[root@pgdbsrv01 ~]# su - postgres
-bash-4.1$ /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf --verbose --force --rsync-only -D
/var/lib/pgsql/9.4/data -d repmgr -p 5432 -U repmgr -R postgres standby clone pgdbsrv02
[2015-07-13 17:25:17] [NOTICE] opening configuration file: /etc/repmgr/9.4/repmgr.conf
[2015-07-13 17:25:17] [NOTICE] destination directory '/var/lib/pgsql/9.4/data' provided
[2015-07-13 17:25:17] [INFO] connecting to upstream node
[2015-07-13 17:25:17] [INFO] connected to upstream node, checking its state
[2015-07-13 17:25:17] [INFO] Successfully connected to upstream node. Current installation size is 39 MB
[2015-07-13 17:25:17] [NOTICE] starting backup...
[2015-07-13 17:25:17] [WARNING] directory "/var/lib/pgsql/9.4/data" exists but is not empty
[2015-07-13 17:25:22] [INFO] standby clone: master data directory '/var/lib/pgsql/9.4/data'
[2015-07-13 17:25:22] [INFO] rsync command line: 'rsync --archive --checksum --compress --progress
--rsh=ssh --delete --checksum --exclude=postmaster.pid --exclude=postmaster.opts
--exclude=global/pg_control --exclude=postgresql.auto.conf.tmp --exclude=pgsql_tmp* --exclude=pg_xlog/*
--exclude=pg_log/* --exclude=pg_stat_tmp/* --exclude=pg_replslot/*
postgres@pgdbsrv02:/var/lib/pgsql/9.4/data/* /var/lib/pgsql/9.4/data'
receiving incremental file list
...
total size is 41881588  speedup is 238.68
[2015-07-13 17:25:22] [INFO] standby clone: local control file '/var/lib/pgsql/9.4/data/global'
[2015-07-13 17:25:22] [INFO] standby clone: master control file
'/var/lib/pgsql/9.4/data/global/pg_control'
[2015-07-13 17:25:22] [INFO] rsync command line: 'rsync --archive --checksum --compress --progress
--rsh=ssh --delete --checksum postgres@pgdbsrv02:/var/lib/pgsql/9.4/data/global/pg_control
/var/lib/pgsql/9.4/data/global'
receiving incremental file list
pg_control
        8192 100%    7.81MB/s    0:00:00 (xfer#1, to-check=0/1)

sent 102 bytes  received 244 bytes  692.00 bytes/sec
total size is 8192  speedup is 23.68
[2015-07-13 17:25:22] [NOTICE] notifying master about backup completion...
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
[2015-07-13 17:25:23] [NOTICE] standby clone (using rsync) complete
[2015-07-13 17:25:23] [NOTICE] HINT: you can now start your PostgreSQL server
[2015-07-13 17:25:23] [NOTICE] for example : pg_ctl -D /var/lib/pgsql/9.4/data start
```

As you can see the command is basically the same, the only change is the addition of the `--rsync-only` option.

At this point, the data is (more or less) in sync with the current primary on *pgdbsrv02*, so let's start *pgdbsrv01* to act as the standby:

On *RHEL/CentOS 6*:

```
[root@pgdbsrv01 ~]# service postgresql-9.4 start
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv01 ~]# systemctl start postgresql-9.4
```

Repmgr notices automatically that the new standby got available:

```
[root@pgdbsrv01 ~]# sudo su - postgres -c " /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf
cluster show"
Role       | Connection String
   standby | host=pgdbsrv1 user=repmgr dbname=repmgr
 * master  | host=pgdbsrv2 user=repmgr dbname=repmgr
```

For *pgpool-II*, we need to re-attach the failed node in order for it to be visible and usable as a standby-node:

```
|----
[root@onmssrv01 ~]# pcp_attach_node 0 localhost 9898 postgres postgres 0
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "show pool_nodes"
Password for user pgpool:
 node_id | hostname | port | status | lb_weight |  role
---------+----------+------+--------+-----------+---------
 0       | pgdbsrv01 | 5432 | 2      | 0.500000  | standby
 1       | pgdbsrv02 | 5432 | 2      | 0.500000  | primary
(2 rows)
|----
```

For the `pcp_attach_node` command, it is required to use the authentication credentials defined on /etc/pgpool-II/pcp.conf. Also, it is required to know the node_id of the node you want to re-attach. The "show pool_nodes" will give you all the information (including the `node_id`). In this case the *ID* for *pgdbsrv01* is `0`.

At this point, we're back in a redundant status where the old standby (*pgdbsrv02*) functions as the primary and the old primary (*pgdbsrv01*) functions as a standby. If both machines are equal, you can leave the situation as is and continue to use it in this way.

Recover to the original situation

This is completely optional and not required. Be sure that *pgpool-II* is running on one of the *OpenNMS* servers before proceed.

In case you want to go back to the initial setup (*pgdbsrv01* as primary, *pgdbsrv02* as standby), you can initiate a manual failover by stopping the current new primary (*pgdbsrv02*) and recover it to use as a standby:

On *RHEL/CentOS 6*:

```
[root@pgdbsrv02 ~]# service postgresql-9.4 stop
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv02 ~]# systemctl stop postgresql-9.4
```

When the master (*pgdbsrv02*) stopped, a failover is triggered and *pgdbsrv01* gets assigned as primary again:

```
|----
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "show pool_nodes"
Password for user pgpool:
 node_id | hostname  | port | status | lb_weight |  role
---------+-----------+------+--------+-----------+---------
 0       | pgdbsrv01 | 5432 | 2      | 0.500000  | primary
 1       | pgdbsrv02 | 5432 | 3      | 0.500000  | standby
(2 rows)
|----
```

Now, sync *pgdbsrv02* with the new primary (*pgdbsrv01*) and start it (this time, use `--rsync-only` as this is an intentional action and the data directory is not corrupted):

```
[root@pgdbsrv02 ~]# sudo su - postgres
-bash-4.2$ /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf --verbose --force --rsync-only -D
/var/lib/pgsql/9.4/data -d repmgr -p 5432 -U repmgr -R postgres standby clone pgdbsrv01
```

Then, start the *postgresql-9.4* service.

On *RHEL/CentOS 6*:

```
[root@pgdbsrv02 ~]# service postgresql-9.4 start
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv02 ~]# systemctl start postgresql-9.4
```

Finally, Re-attach *pgdbsrv02* to *pgpool-II* in order to use it as a slave:

```
|----
[root@onmssrv01 ~]# pcp_attach_node 0 localhost 9898 postgres postgres 1
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "show pool_nodes"
 node_id | hostname | port | status | lb_weight |  role
---------+----------+------+--------+-----------+---------
 0       | pgdbsrv01| 5432 | 2      | 0.500000  | primary
 1       | pgdbsrv02| 5432 | 2      | 0.500000  | standby
(2 rows)
|----
```

After going trough all of the above steps, we're back to the initial situation where *pgdbsrv01* acts as the primary and *pgdbsrv02* acts as the standby.

IMPORTANT	When you're done testing *pgpool-II*, remember to stop the service on the *OpenNMS* server where it is running.

Recover to the original situation

This is completely optional and not required. Be sure that *pgpool-II* is running on one of the *OpenNMS* servers before proceed.

In case you want to go back to the initial setup (*pgdbsrv01* as primary, *pgdbsrv02* as standby), you can initiate a manual

failover by stopping the current new primary (*pgdbsrv02*) and recover it to use as a standby:

On *RHEL/CentOS 6*:

```
[root@pgdbsrv02 ~]# service postgresql-9.4 stop
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv02 ~]# systemctl stop postgresql-9.4
```

When the master (*pgdbsrv02*) stopped, a failover is triggered and *pgdbsrv01* gets assigned as primary again:

```
|----
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "show pool_nodes"
Password for user pgpool:
 node_id | hostname  | port | status | lb_weight |  role
---------+-----------+------+--------+-----------+---------
 0       | pgdbsrv01 | 5432 | 2      | 0.500000  | primary
 1       | pgdbsrv02 | 5432 | 3      | 0.500000  | standby
(2 rows)
|----
```

Now, sync *pgdbsrv02* with the new primary (*pgdbsrv01*) and start it (this time, use `--rsync-only` as this is an intentional action and the data directory is not corrupted):

```
[root@pgdbsrv02 ~]# sudo su - postgres
-bash-4.2$ /usr/pgsql-9.4/bin/repmgr -f /etc/repmgr/9.4/repmgr.conf --verbose --force --rsync-only -D
/var/lib/pgsql/9.4/data -d repmgr -p 5432 -U repmgr -R postgres standby clone pgdbsrv01
```

Then, start the *postgresql-9.4* service.

On *RHEL/CentOS 6*:

```
[root@pgdbsrv02 ~]# service postgresql-9.4 start
```

On *RHEL/CentOS 7*:

```
[root@pgdbsrv02 ~]# systemctl start postgresql-9.4
```

Finally, Re-attach *pgdbsrv02* to *pgpool-II* in order to use it as a slave:

```
|----
[root@onmssrv01 ~]# pcp_attach_node 0 localhost 9898 postgres postgres 1
[root@onmssrv01 ~]# psql -U pgpool -h onmssrv01 -p 9999 -d postgres -c "show pool_nodes"
 node_id | hostname | port | status | lb_weight |  role
---------+----------+------+--------+-----------+---------
 0       | pgdbsrv01| 5432 | 2      | 0.500000  | primary
 1       | pgdbsrv02| 5432 | 2      | 0.500000  | standby
(2 rows)
|----
```

After going trough all of the above steps, we're back to the initial situation where *pgdbsrv01* acts as the primary and *pgdbsrv02* acts as the standby.

IMPORTANT	When you're done testing *pgpool-II*, remember to stop the service on the *OpenNMS* server where it is running.

Troubleshooting

In case you have trouble starting the *pgpool* service, be sure that it is not running on the other *OpenNMS* machine, and also that the socket files (i.e. `/tmp/.s.PGSQL.9898` and `/tmp/.s.PGSQL.9999`) don't exist. If they exist, remove them prior start. In theory this cleanup is always performed by the *pgpool* initialization script, but it is important to understand why.

The cleanup is performed through the script located at `/usr/local/bin/dbcleanup.sh` as mentioned on an earlier section. The reason to have this integrated with the *pgpool* initialization script is to be sure that *OpenNMS* will be able to initialize the *DB Connection Pool* without issues. Because of this, prior every single start request of the *OpenNMS* application (either manually or through the cluster), *pgpool* must be restarted first. As an alternative, prior starting *OpenNMS*, execute the `dbcleanup.sh` script to be sure that there are no connections against the *DB* servers when *OpenNMS* starts.