**Set Up Highly Available PostgreSQL12 Cluster on CentOS, RHEL 8**

<https://www.techsupportpk.com/2020/02/how-to-create-highly-available-postgresql-cluster-centos-rhel-8.html>

This guide will walk you through the steps to set up a highly available PostgreSQL cluster using Patroni and HAProxy on CentOS8. These steps can also be applied, (changes may be required) if you are running an earlier release of CentOS or RHEL Linux. 

**Prerequisites**

To follow this tutorial along, you will need at least 4 (physical or virtual) machines installed with CentOS8. We will use these 4 virtual machines prepared with CentOS Linux release 8.3.2011 throughout this guide for postgres cluster set up.

HOSTNAME         IP ADDRESS PURPOSE

pg\_node1 192.168.10.1 Postgresql+Patroni

pg\_node2 192.168.10.2 Postgresql+Patroni

etcd\_node1 192.168.10.3 etcd

haproxy\_node1 192.168.10.4 HAProxy

**Note:** For a production environment, it is recommended to set up at least (2-nodes postgres+patroni), (2-nodes etcd), and (2-nodes HAProxy), in order to achieve high availability and to avoid a single point of failure.

When all prerequisites are in place, you may proceed with the following steps to set up your PostgreSQL HA cluster.

**Install Epel Repo**

We will install epel repository on all of the nodes, (pg\_node1, pg\_node2, etcd\_node1, and haproxy\_node1) in our case, with below command:

sudo dnf -y install epel-release  
sudo dnf config-manager --set-enabled PowerTools  
sudo dnf -y install yum-utils  
sudo dnf -y update

Reboot your machine when you are done with the above in order to take changes into effect:

sudo shutdown -r now

Make sure you repeat the same on each node before proceeding to next step.

**Install PostgreSQL**

For this guide, we will install PostgreSQL version 12 on two of the nodes (pg\_node1,pg\_node2) in our case, like below:

sudo dnf -y install https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86\_64/pgdg-redhat-repo-latest.noarch.rpm  
sudo yum-config-manager --enable pgdg12  
sudo dnf -qy module disable postgresql  
sudo dnf -y install postgresql12-server postgresql12 postgresql12-devel

**Install Patroni**

Patroni is a cluster manager used to customize and automate deployment and maintenance of PostgreSQL HA (High Availability) clusters. You should check the latest available release from [Github](https://github.com/cybertec-postgresql/patroni-packaging/releases) page.

For this guide, the currently available release is 1.6.5-1 and we will install it on (pg\_node1 and pg\_node2) in our case, like below:

sudo dnf -y install https://github.com/cybertec-postgresql/patroni-packaging/releases/download/1.6.5-1/patroni-1.6.5-1.rhel7.x86\_64.rpm

Patroni uses the YAML file to store its configuration. So, you will need to create a configuration file for Patroni on (pg\_node1 and pg\_node2) like below:

sudo cp -p /opt/app/patroni/etc/postgresql.yml.sample /opt/app/patroni/etc/postgresql.yml

Next edit the postgresql.yml file with any of your favorite text editor like below:

sudo nano /opt/app/patroni/etc/postgresql.yml

Remove everything from this file, and add the following configuration parameters. Make sure, you change namespace, listen and connect\_address to reflect yours.

scope: postgres  
namespace: /pg\_cluster/  
name: pg\_node1  
  
restapi:  
 listen: 192.168.10.1:8008  
 connect\_address: 192.168.10.1:8008  
  
etcd:  
 host: 192.168.10.3:2379  
  
bootstrap:  
 dcs:  
 ttl: 30  
 loop\_wait: 10  
 retry\_timeout: 10  
 maximum\_lag\_on\_failover: 1048576  
 postgresql:  
 use\_pg\_rewind: true  
 use\_slots: true  
 parameters:  
  
 initdb:  
 - encoding: UTF8  
 - data-checksums  
  
 pg\_hba:  
 - host replication replicator 127.0.0.1/32 md5  
 - host replication replicator 192.168.10.1/0 md5  
 - host replication replicator 192.168.10.2/0 md5  
 - host all all 0.0.0.0/0 md5  
  
 users:  
 admin:  
 password: admin  
 options:  
 - createrole  
 - createdb  
  
postgresql:  
 listen: 192.168.10.1:5432  
 connect\_address: 192.168.10.1:5432  
 data\_dir: /var/lib/pgsql/12/data  
 bin\_dir: /usr/pgsql-12/bin  
 pgpass: /tmp/pgpass  
 authentication:  
 replication:  
 username: replicator  
 password: replicator  
 superuser:  
 username: postgres  
 password: postgres  
  
tags:  
 nofailover: false  
 noloadbalance: false  
 clonefrom: false  
 nosync: false

Save and close the editor when you are finished.

Next, edit **postgresql.yml** file on **(pg\_node2**) in our case, and add the following configuration parameters. Make sure, you change namespace, listen and connect\_address to reflect yours:

scope: postgres  
namespace: /pg\_cluster/  
name: pg\_node2  
  
restapi:  
 listen: 192.168.10.2:8008  
 connect\_address: 192.168.10.2:8008  
  
etcd:  
 host: 192.168.10.3:2379  
  
bootstrap:  
 dcs:  
 ttl: 30  
 loop\_wait: 10  
 retry\_timeout: 10  
 maximum\_lag\_on\_failover: 1048576  
 postgresql:  
 use\_pg\_rewind: true  
 use\_slots: true  
 parameters:  
  
 initdb:  
 - encoding: UTF8  
 - data-checksums  
  
 pg\_hba:  
 - host replication replicator 127.0.0.1/32 md5  
 - host replication replicator 192.168.10.1/0 md5  
 - host replication replicator 192.168.10.2/0 md5  
 - host all all 0.0.0.0/0 md5  
  
 users:  
 admin:  
 password: admin  
 options:  
 - createrole  
 - createdb  
  
postgresql:  
 listen: 192.168.10.2:5432  
 connect\_address: 192.168.10.2:5432  
 data\_dir: /var/lib/pgsql/12/data  
 bin\_dir: /usr/pgsql-12/bin  
 pgpass: /tmp/pgpass  
 authentication:  
 replication:  
 username: replicator  
 password: replicator  
 superuser:  
 username: postgres  
 password: postgres  
  
tags:  
 nofailover: false  
 noloadbalance: false  
 clonefrom: false  
 nosync: false

Save and close the editor when you are finished.

**Install etcd**

etcd is a strongly consistent, distributed key-value store that provides a reliable way to store data that needs to be accessed by a distributed system or cluster of machines. We will use etcd to store the state of the Postgres cluster in order to keep the Postgres cluster up and running.

Log in to (etcd\_node1) in our case, and type below command to install etcd:

sudo dnf -y install http://mirror.centos.org/centos/7/extras/x86\_64/Packages/etcd-3.3.11-2.el7.centos.x86\_64.rpm

**Configure etcd**

We will edit default etcd configuration file to make few changes.

sudo nano /etc/etcd/etcd.conf

Locate and uncomment the following parameters, and make sure you update these highlighted values to reflect yours:

[Member]  
ETCD\_LISTEN\_PEER\_URLS="http://192.168.10.3:2380,http://localhost:2380"  
ETCD\_LISTEN\_CLIENT\_URLS="http://192.168.10.3:2379,http://localhost:2379"  
  
[Clustering]  
ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="http://192.168.10.3:2380"  
ETCD\_ADVERTISE\_CLIENT\_URLS="http://192.168.10.3:2379"  
ETCD\_INITIAL\_CLUSTER="default=http://192.168.10.3:2380"  
ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"  
ETCD\_INITIAL\_CLUSTER\_STATE="new"

Save and close the editor when you are finished.

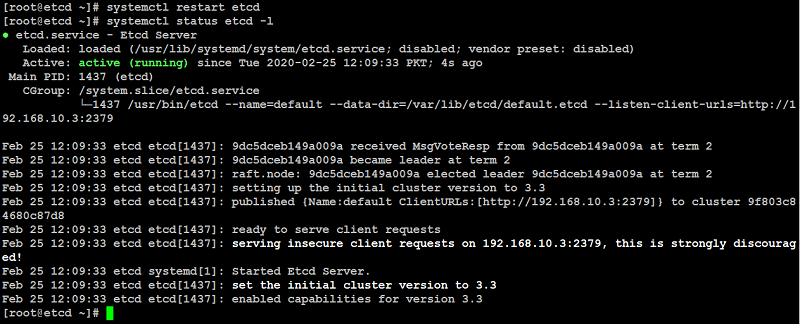
Next, start the etcd to take the changes into effect:

sudo systemctl enable etcd

sudo systemctl start etcd

sudo systemctl status etcd

The etcd is now active and running.



Reboot your (etcd\_node1) if etcd failed to start.

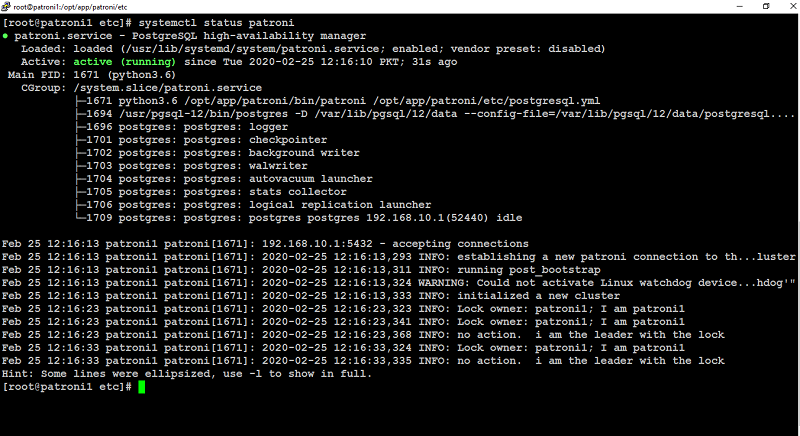
Once your etcd is up and running, you need to go back to (pg\_node1 and pg\_node2) in our case, to start patroni like below:

sudo systemctl enable patroni

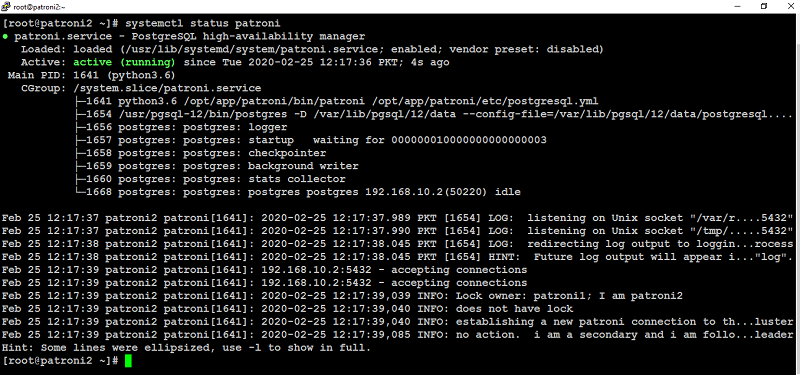
sudo systemctl start patroni

sudo systemctl status patroni

If you look carefully at the bottom of the following output, you can see that the (pg\_node1) is acting as leader (master) node in the cluster.



When you start patroni on subsequent nodes, (pg\_node2) for example, you will see (pg\_node2) is acting as secondary.



**Install HAProxy**

HAProxy is free, open source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications that spreads requests across multiple servers. HAProxy forwards the connection to whichever node is currently the master. It does this using a REST endpoint that Patroni provides. Patroni ensures that, at any given time, only the master node will appear as online, forcing HAProxy to connect to the correct node.

Log in to your (haproxy\_node1) in our case, and install haproxy with below command:

sudo dnf -y install haproxy

**Configure HAProxy**

With the Postgres cluster, you need a method to connect to the master regardless of which of the nodes in the cluster is the master. This is where HAProxy steps in. Database client or applications, (psql) for example, will connect to haproxy, and haproxy will make sure you connect to the master node in the cluster.

We will edit and make few changes in the default haproxy.cfg file:

sudo cp -p /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.bkp

sudo nano /etc/haproxy/haproxy.cfg

Remove everything from this file, and add the following configuration parameters.

global  
 log 127.0.0.1 local2  
  
 chroot /var/lib/haproxy  
 pidfile /var/run/haproxy.pid  
 user haproxy  
 group haproxy  
 daemon  
  
 stats socket /var/lib/haproxy/stats  
  
defaults  
 mode tcp  
 log global  
 retries 3  
 timeout queue 1m  
 timeout connect 10s  
 timeout client 1m  
 timeout server 1m  
 timeout check 10s  
 maxconn 3000  
  
listen stats  
 mode http  
 bind \*:7000  
 stats enable  
 stats uri /  
  
listen postgres  
 bind \*:5000  
 option httpchk  
 http-check expect status 200  
 default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions  
 server pg\_node1 192.168.10.1:5432 maxconn 1000 check port 8008  
 server pg\_node2 192.168.10.2:5432 maxconn 1000 check port 8008

Make sure you replace the highlighted text with yours.

Save and close the editor when you are finished.

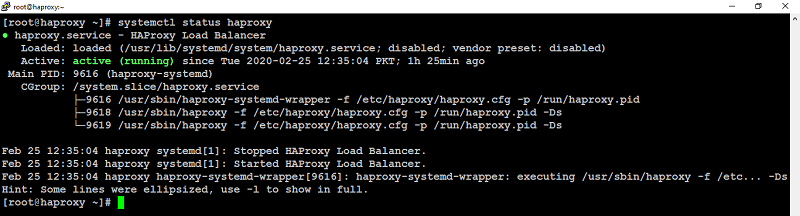
Next, start HAProxy to take changes into effect:

sudo systemctl start haproxy

sudo systemctl enable haproxy

sudo systemctl status haproxy

HAProxy is now active and running:

[](https://1.bp.blogspot.com/-5DyhHcJ8Q4k/YA_s8BeMkdI/AAAAAAAATDM/xsIdsVPRzLc-znSmomCFYMNSiBgo9GQhACLcBGAsYHQ/s800/haproxy_status_centos.png)

If haproxy failed to start, you should check configuration file syntax errors with below command:

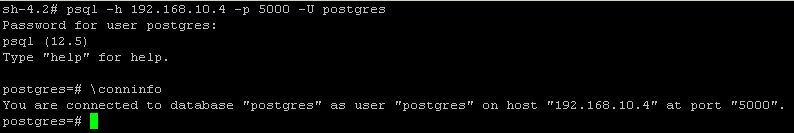
sudo haproxy -c -V -f /etc/haproxy/haproxy.cfg

**Test Postgres Cluster**

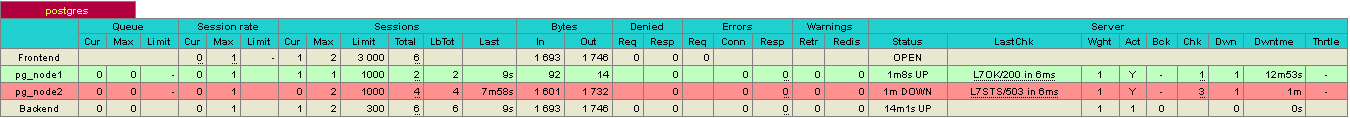
You can test and verify your Postgres cluster by initiating a connection request to (haproxy\_node1) in our case, from any of your applications, (psql) for example, and see if this successfully establish connection to the database master node in the cluster.

psql -h 192.168.10.4 -p 5000 -U postgres

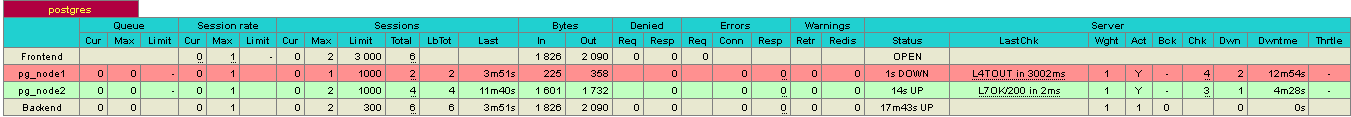
As you can see in the image below, the (psql) successfully connected to a database node in the  cluster via haproxy.



Next, open up a web browser and access **http://192.168.10.4:7000**, and you will see a haproxy dashboard like below:



As you can see, the **pg\_node1** row is highlighted in green. This indicates that 192.168.10.1 is currently acting as the master. If you kill the primary node pg\_node1 using (**sudo systemctl stop patroni**) or by completely shutting down the server, the dashboard will look similar to like below:



As you can see, in the Postgres section, the **pg\_node1** row is now red and the **pg\_node2** row is highlighted in green. This indicates that 192.168.10.2 is currently acting as the master.

Please note that, in this particular scenario, it just so happens that the second Postgres server is promoted to master. This might not always be the case if you have more than two nodes in the cluster. It is equally likely that the third, fourth or fifth node may be promoted to master.

**Test Replication**

We will create a test database to see if it is replicated to other nodes in the cluster. For this guide, we will use (psql) to connect to database via haproxy like below:

psql -h 192.168.10.4 -p 5000 -U postgres

From the Postgres prompt, create a test database like below:

create database testdb;  
create user testuser with encrypted password 'testpass';  
grant all privileges on database testdb to testuser;  
  
\q

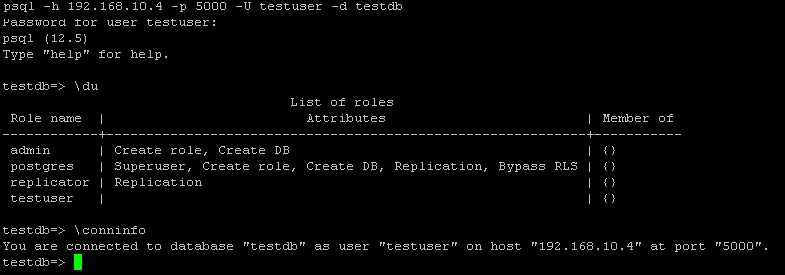
Next, stop patroni on the master node (pg\_node1) in our case with below command:

sudo systemctl stop patroni

Next, connect to database using psql, and this time haproxy will make connection to (pg\_node2) acting as master node as we have already stopped patroni on (pg\_node1):

sudo psql -h 192.168.10.4 -p 5000 testuser -d testdb

As you can see in the output below, connection to testdb was successfully via haproxy on (pg\_node2).



Now bring up the first node with (systemctl start patroni), it will rejoin the cluster as a slave and will automatically sync up with the master.

**Patroni Operations – switchover and failover**

With patronictl, you can administer, manage and troubleshoot your Postgres cluster. Type below command to list the options and commands you can use with patronictl:

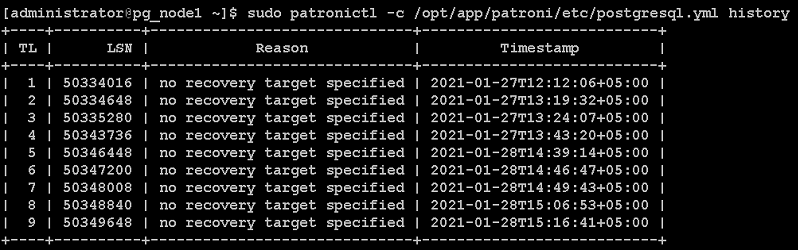
sudo patronictl --help

This will show you the options and commands you can use with patronictl.

Options:  
 -c, --config-file TEXT Configuration file  
 -d, --dcs TEXT Use this DCS  
 -k, --insecure Allow connections to SSL sites without certs  
 --help Show this message and exit.  
  
Commands:  
 configure Create configuration file  
 dsn Generate a dsn for the provided member, defaults to a dsn of...  
 edit-config Edit cluster configuration  
 failover Failover to a replica  
 flush Discard scheduled events (restarts only currently)  
 history Show the history of failovers/switchovers  
 list List the Patroni members for a given Patroni  
 pause Disable auto failover  
 query Query a Patroni PostgreSQL member  
 reinit Reinitialize cluster member  
 reload Reload cluster member configuration  
 remove Remove cluster from DCS  
 restart Restart cluster member  
 resume Resume auto failover  
 scaffold Create a structure for the cluster in DCS  
 show-config Show cluster configuration  
 switchover Switchover to a replica  
 version Output version of patronictl command or a running Patroni

For example you can check failover history across nodes in the cluster with below command:

sudo patronictl -c /opt/app/patroni/etc/postgresql.yml history



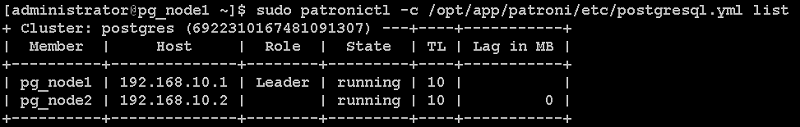
The failover is executed automatically, when the Leader node is getting unavailable for unplanned reason. If you wish to test failover across the nodes in the cluster, you can manually initiate failover to a replica node with below command:

sudo patronictl -c /opt/app/patroni/etc/postgresql.yml failover



You can also check cluster state, role and members with below command:

sudo patronictl -c /opt/app/patroni/etc/postgresql.yml list

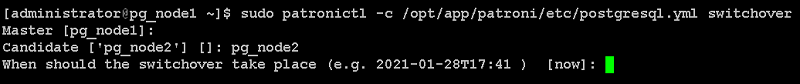


In some cases it is necessary to perform maintenance task on a single node. For example applying patches or release updates. When you manually disable auto failover, patroni won’t change the state of the PostgreSQL. For example it will not try to start the cluster when it is stopped.

You can disable auto failover with below command:

sudo patronictl -c /opt/app/patroni/etc/postgresql.yml pause

There are two possibilities to run a switchover, either in scheduled mode or immediately. At the given time, the switchover will take place, and you will see in the logfile an entry of switchover activity.



If you go with **[now]** option, switchover will take place immediately.

**Conclusion**

Your Postgres cluster is now ready to serve your purpose. However, if you would like to make it more robust and highly available, here are a few more steps you can take to improve it further:

Use a larger etcd cluster to improve availability.

Use PgBouncer to pool connections.

Add a second node in HAProxy and configure IP failover to avoid single point of failure.