**PostgreSQL High Availability Clustering Using DRBD and**

**Pacemaker on Ubuntu**

Rishabh Katiyar, 24 April 2025

Contents

[Introduction 3](#_Toc196400768)

[Initial Configuration 4](#_Toc196400769)

[Install PostgreSQL 5](#_Toc196400770)

[Installing DRBD 7](#_Toc196400771)

[System Configurations 8](#_Toc196400772)

[Firewall Configuration 8](#_Toc196400773)

[Configuring DRBD 9](#_Toc196400774)

[Creating a Filesystem 11](#_Toc196400775)

[PSQL Data Directory Configuration 11](#_Toc196400776)

[Install Pacemaker and Corosync 13](#_Toc196400777)

[Configuring Corosync 14](#_Toc196400778)

[Creating a Basic Pacemaker Configuration 15](#_Toc196400779)

[Configuring Cluster Resources 16](#_Toc196400780)

[Adding Network Connectivity Monitoring 18](#_Toc196400781)

# Introduction

This guide describes the steps performed on the ec2 instances for implementing High Availability (HA) PostgreSQL (14.17) cluster using DRBD, Pacemaker and Corosync on 22.04.1 Ubuntu servers.

DRBD based shared-nothing cluster - This cluster type makes use of local, directly attached storage whose content is synchronously replicated between cluster nodes. This adds an additional layer of redundancy in that PSQL’s data storage is available on more than one node. Like shared storage clusters, DRBD based clusters guarantee transaction integrity across failover

Initial Configuration  
  
Get three ec2 instances in AWS, two for hosting PostgreSQL and one to use as a client to connect with the DB cluster. Name them PG1, PG2 and Client respectively for easy identification.  
  
Let us assume we got the nodes with the following properties –

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Private IPv4 Address | Host Name | Secondary IP |
| PG1 | 172.31.15.48 | ip-172-31-15-48 | 172.31.5.200 |
| PG2 | 172.31.5.192 | ip-172-31-5-192 | 172.31.5.200 |
| Client | 172.31.13.225 | ip-172-31-13-225 |  |

Run all the upcoming commands with elevate privileges

|  |
| --- |
| sudo -s |

Install PostgreSQL  
  
On both the nodes, update the Package List

|  |
| --- |
| sudo apt update |

Install PostgreSQL

|  |
| --- |
| sudo apt install postgresql postgresql-contrib |

Confirm the Service is running

|  |
| --- |
| sudo systemctl status postgresql |

To start PostgreSQL (if not running):

|  |
| --- |
| sudo systemctl start postgresql |

To enable PostgreSQL to start on boot:

|  |
| --- |
| sudo systemctl enable postgresql |

Switch to the Default PostgreSQL User

PostgreSQL creates a default user named postgres.

To access the PostgreSQL prompt:

|  |
| --- |
| sudo -i -u postgres  psql |

Set Password for the postgres User

|  |
| --- |
| sudo -i -u postgres  psql |

In the PostgreSQL shell:

|  |
| --- |
| ALTER USER postgres PASSWORD 'your\_password'; |

**Enable Remote Access**

If you want to allow connections from other machines:

Edit PostgreSQL config:

|  |
| --- |
| sudo nano /etc/postgresql/\*/main/postgresql.conf |

Uncomment and change:

|  |
| --- |
| listen\_addresses = '\*' |

Edit client authentication config:

|  |
| --- |
| sudo nano /etc/postgresql/\*/main/pg\_hba.conf |

Add a line like:

|  |
| --- |
| host all all 0.0.0.0/0 md5 |

Restart PostgreSQL:

|  |
| --- |
| sudo systemctl restart postgresql |

Check connection with the PostgreSQL on both the nodes using client node

Stop PostgreSQL on both PG1 and PG2

|  |
| --- |
| systemctl stop postgresql@14-main |

#### Installing DRBD 1. **Update System**

|  |
| --- |
| sudo apt update  sudo apt upgrade -y |

#### 2. **Install DRBD Utilities**

On recent Ubuntu versions (20.04+), DRBD is part of the Linux kernel, and you just need the userspace utilities:

|  |
| --- |
| sudo apt install drbd-utils -y |

#### 3. **Verify Installation**

Check the version:

|  |
| --- |
| drbdadm --version |

System Configurations  
  
Add volume in both nodes PG1 and PG2, for the demo I used 10 GB volumes for each node. Assume device name is /dev/xvdd run the following commands to setup the disks in both the nodes.

|  |
| --- |
| pvcreate /dev/xvdd  vgcreate vg\_drbd /dev/vdb  lvcreate -L 20G -n lv\_psql vg\_drbd |

# Firewall Configuration

For setting up the HA, I had used security group that allowed all TCP and UDP ports. In the actual deployment setup adjust for the security accordingly.

# 

# Configuring DRBD

First, it is necessary to configure a DRBD resource to serve as a backing device for the PSQL database. It is recommended to use a separate DRBD resource for each database. The suggested method is to use individual logical volumes as the backing device for each DRBD resource. While this is not absolutely necessary, it will provide a degree of granularity and allow you to distribute databases across different nodes, rather than having all databases always run on a single node.

It is highly recommended that you put your resource configurations in separate resource files that reside in the /etc/drbd.d directory, in both nodes PG1 and PG2, whose name is identical to that of the resource such as /etc/drbd.d/psql.res. Its contents should look similar to this:

|  |
| --- |
| resource psql {          protocol        C;          device          /dev/drbd0;          disk            /dev/vg\_drbd/lv\_psql;          meta-disk internal;          on ip-172-31-15-48 {                  address 172.31.15.48:7788;          }          on ip-172-31-5-192  {                  address 172.31.5.192:7788;          }  } |

Copy this configuration to both DRBD nodes. Next, it will be necessary to bring the DRBD resource named psql up and online as it will serve as the backing storage for the database.

First create the metadata for the DRBD resource. This step must be done on both nodes:

|  |
| --- |
| drbdadm create-md psql |

Then load the DRBD module and bring the resource up on both nodes

|  |
| --- |
| drbdadm up psql |

The DRBD resource should now be in the connected state, the Secondary role on both nodes, and show a disk state of Inconsistent on both nodes. Verify their state by typing the following on either node:

|  |
| --- |
| # drbdadm status  psql role:Secondary  disk:Inconsistent  node-b role:Secondary  peer-disk:Inconsistent |

At this point we may either begin the initial device synchronization, or, as this is a brand new volume and identical on both nodes already (empty), we can safely skip the initial synchronization with the --clear-bitmap option. Run the following command on one node:

|  |
| --- |
| drbdadm --clear-bitmap new-current-uuid psql/0 |

The DRBD resource should now be in the connected state, the Secondary role on both nodes, and show a disk state of UpToDate on both nodes. Verify their state by typing the following on either node:

|  |
| --- |
| # drbdadm status  psql role:Secondary  disk:UpToDate  node-b role:Secondary  peer-disk:UpToDate |

Now that the resource disk state is no longer inconsistent, promote the resource to Primary on the node you wish to use for creation of the filesystem:

|  |
| --- |
| drbdadm primary psql |

# Creating a Filesystem

Once the DRBD resource has been created and initialized, you can create a filesystem on the new block device. This example assumes xfs as the filesystem type:

|  |
| --- |
| mkfs.xfs /dev/drbd0 |

You only need to create the filesystem on the Primary node. Other filesystems such as ext4 and btrfs may be deployed instead of xfs.

# PSQL Data Directory Configuration

After the filesystem has been created it can be mounted over the default PSQL data directory /var/lib/postgresql/14/main Temporarily mount the filesystem on the Primary node:

|  |
| --- |
| mount /dev/drbd0 /var/lib/postgresql/14/main |

Change the file ownership for the newly mounted filesystem on the Primary node:

|  |
| --- |
| chown postgres:postgres /var/lib/postgresql/14/main |

Next, start the psql service on the Primary node:

|  |
| --- |
| systemctl start postgresql@14-main |

Create table in postgres and insert some data to later check if the sync is happening between both the nodes.

**Troubleshooting**  
if /var/lib/postgresql/14/main is empty, run -

|  |
| --- |
| sudo -u postgres /usr/lib/postgresql/14/bin/initdb -D /var/lib/postgresql/14/main  sudo systemctl start postgresql@14-main |

**Testing DRBD Setup**

Once data is created in the primary node, stop postgres service there, unmount the directory and mark the primary node as secondary w.r.t. drbd, to achieve this use the following steps on primary node in the same order to

|  |
| --- |
| systemctl stop postgresql@14-main  sudo umount /var/lib/postgresql/14/main  drbdadm secondary psql |

Now run the following commands in the same order in second node to make it a primary postgresql node –

|  |
| --- |
| drbdadm primary psql  mount /dev/drbd0 /var/lib/postgresql/14/main  systemctl start postgresql@14-main |

Check if the created table and data is synced successfully, do not proceed further in the guide until sync of PG data is not working properly.

Once the data is getting synced, stop postgres service in both the nodes, starting up of this service will be taken care by pacemaker and Corosync.

# Install Pacemaker and Corosync

1. Update the system

|  |
| --- |
| sudo apt update && sudo apt upgrade -y |

2. Install Corosync and Pacemaker

|  |
| --- |
| sudo apt install -y corosync pacemaker |

3. Enable and start services

|  |
| --- |
| sudo systemctl enable corosync  sudo systemctl enable pacemaker  sudo systemctl start corosync  sudo systemctl start pacemaker |

# Configuring Corosync

Create and edit the file /etc/corosync/corosync.conf on both nodes, it should look like this:

|  |
| --- |
| totem {    version: 2    secauth: off    cluster\_name: cluster    transport: knet    rrp\_mode: passive  }  nodelist {    node {      ring0\_addr: 172.31.15.48      nodeid: 1      name: ip-172-31-15-48    }    node {      ring0\_addr: 172.31.5.192      nodeid: 2      name: ip-172-31-5-192    }  }  quorum {    provider: corosync\_votequorum    two\_node: 1  }  logging {    to\_syslog: yes  } |

Now that Corosync has been configured we can start the Corosync and Pacemaker services:

|  |
| --- |
| systemctl start corosync  systemctl start pacemaker |

Verify that everything has been started and is working correctly by issuing the following command

|  |
| --- |
| crm status |

# Creating a Basic Pacemaker Configuration

In a highly available 2 node cluster using DRBD, you should:

• Disable STONITH.

• Set Pacemaker’s "no quorum policy" to ignore loss of quorum.

• Set the default resource stickiness to 200.

To do so, issue the following commands from the CRM shell accessible from the crm command on either node (not both):

|  |
| --- |
| # crm  crm(live)# configure  crm(live)configure# property stonith-enabled="false"  crm(live)configure# property no-quorum-policy="ignore"  crm(live)configure# rsc\_defaults resource-stickiness="200"  crm(live)configure# commit  crm(live)configure# exit  bye |

# Configuring Cluster Resources

Run the following command to install the resource agents -

|  |
| --- |
| sudo apt install -y resource-agents |

**Basic Configuration**

In order to create the appropriate cluster resources, open the crm configuration shell as root and issue the following commands on either node (not both):

|  |
| --- |
| # crm  crm(live)# configure  crm(live)configure# primitive p\_drbd\_psql ocf:linbit:drbd \  params drbd\_resource="psql" \  op start interval="0s" timeout="240s" \  op stop interval="0s" timeout="100s" \  op monitor interval="29s" role="Master" \  op monitor interval="31s" role="Slave"  crm(live)configure# ms ms\_drbd\_psql p\_drbd\_psql \  meta master-max="1" master-node-max="1" \  clone-max="2" clone-node-max="1" \  notify="true"  crm(live)configure# primitive p\_fs\_psql ocf:heartbeat:Filesystem \  params device="/dev/drbd0" \  directory="/var/lib/postgresql/14/main" \  fstype="xfs" \  op start interval="0" timeout="60s" \  op stop interval="0" timeout="60s" \  op monitor interval="20" timeout="40s"  crm(live)configure# primitive p\_ip\_psql ocf:heartbeat:IPaddr2 \  params ip="172.31.5.200" cidr\_netmask="24" \  op start interval="0s" timeout="20s" \  op stop interval="0s" timeout="20s" \  op monitor interval="20s" timeout="20s"  crm(live)configure# primitive p\_psql ocf:heartbeat:pgsql \  params pgctl="/usr/lib/postgresql/14/bin/pg\_ctl" \  pgdata="/var/lib/postgresql/14/main" \  op start interval="0s" timeout="120s" \  op stop interval="0s" timeout="120s" \  op monitor interval="20s" timeout="30s"  crm(live)configure# group g\_psql \  p\_fs\_psql p\_ip\_psql p\_psql  crm(live)configure# colocation c\_psql\_on\_drbd \  inf: g\_psql ms\_drbd\_psql:Master  crm(live)configure# order o\_drbd\_before\_psql \  ms\_drbd\_psql:promote g\_psql:start  crm(live)configure# commit  crm(live)configure# exit |

Once this configuration has been committed, Pacemaker will:

• Start DRBD on both cluster nodes.

• Select one node for promotion to the DRBD Primary role.

• Mount the file system, configure the cluster IP address, and start the PostgreSQL server instance on the same node.

• Commence resource monitoring.

# Adding Network Connectivity Monitoring

Finally, Pacemaker may be configured to monitor the upstream network and ensure that PostgreSQL runs only on nodes that have connectivity to clients. In order to do so, pick one or more IP addresses that the cluster node can expect to always be accessible, such as the subnet’s default gateway, a core switch, or similar. The following example uses the default gateway with IP address 172.31.0.1.

Add the ping resources as follows, issue the following commands on either node (not both):

|  |
| --- |
| # crm  crm(live)# configure  crm(live)configure# primitive p\_ping\_gw ocf:pacemaker:ping \ params host\_list="172.31.0.1" \ dampen="5s" \ multiplier="1000" \ op start interval="0s" timeout="60s" \ op stop interval="0s" timeout="60s" \ op monitor interval="15s" timeout="60s"  crm(live)configure# clone cl\_ping p\_ping\_gw \  meta interleave="true"  crm(live)configure# location l\_drbd\_primary\_on\_ping ms\_drbd\_psql \  rule $role="Master" \  -inf: not\_defined pingd or pingd lte 0  crm(live)configure# commit  crm(live)configure# exit |

Once these changes have been committed, Pacemaker will:

• Monitor upstream IP addresses from both cluster nodes.

• Periodically update a node attribute for each node with a value corresponding to the number of reachable upstream hosts.

• Move resources away from any node that loses connectivity to upstream IP addresses.

Now, test out the HA cluster using client node and by manually stopping pacemaker and Corosync services or any other suitable methods.