**How To Set Up Physical Streaming Replication with PostgreSQL 12 on Ubuntu 20.04**

[How To Set Up Physical Streaming Replication with PostgreSQL 12 on Ubuntu 20.04 | DigitalOcean](https://www.digitalocean.com/community/tutorials/how-to-set-up-physical-streaming-replication-with-postgresql-12-on-ubuntu-20-04)

* By [Nathan McCulloch](https://www.digitalocean.com/community/users/nathanmcculloch)

Last Validated onNovember 23, 2020 Originally Published onNovember 11, 2020 6.1kviews

Introduction

Streaming replication is a popular method you can use to horizontally scale your relational databases. It uses two or more copies of the same database cluster running on separate machines. One database cluster is referred to as the primary and serves both read and write operations; the others, referred to as the replicas, serve only read operations. You can also use streaming replication to provide high availability of a system. If the primary database cluster or server were to unexpectedly fail, the replicas are able to continue serving read operations, or (one of the replicas) become the new primary cluster.

[PostgreSQL](https://www.postgresql.org/) is a widely used [relational database](https://www.digitalocean.com/community/tutorials/understanding-relational-databases) that supports both logical and physical replication. *Logical replication* streams high-level changes from the primary database cluster to the replica databases. Using logical replication, you can stream changes to just a single database or table in a database. However, in *physical replication*, changes to the [WAL (Write-Ahead-Logging)](https://en.wikipedia.org/wiki/Write-ahead_logging) log file are streamed and replicated in the replica clusters. As a result, you can’t replicate specific areas of a primary database cluster, but instead all changes to the primary are replicated.

In this tutorial, you will set up physical streaming replication with PostgreSQL 12 on Ubuntu 20.04 using two separate machines running two separate PostgreSQL 12 clusters. One machine will be the **primary** and the other, the **replica**.

Prerequisites

To complete this tutorial, you will need the following:

* Two separate machines Ubuntu 20.04 machines; one referred to as the **primary** and the other referred to as the **replica**. You can set these up with our [Initial Server Setup Guide](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-20-04), including non-root users with sudo permissions and a firewall.
* Your firewalls configured to allow HTTP/HTTPS and traffic on port 5432—the default port used by PostgreSQL 12. You can follow [How To Set Up a Firewall with ufw on Ubuntu 20.04](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-with-ufw-on-ubuntu-20-04) to configure these firewall settings.
* PostgreSQL 12 running on both Ubuntu 20.04 Servers. Follow **Step 1** of the [How To Install and Use PostgreSQL on Ubuntu 20.04](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-20-04) tutorial that covers the installation and basic usage of PostgreSQL on Ubuntu 20.04.

Step 1 — Configuring the Primary Database to Accept Connections

In this first step, you’ll configure the **primary** database to allow your **replica** database(s) to connect. By default, PostgreSQL only listens to the localhost (127.0.0.1) for connections. To change this, you’ll first edit the listen\_addresses configuration parameter on the **primary** database.

On your **primary** server, open the PostgreSQL configuration file, postgresql.conf, found in the /etc/postgresql/12/main/ directory:

* sudo nano /etc/postgresql/12/main/postgresql.conf

Once the file is open, locate the listen\_addresses variable and change the value from localhost to the IP address of your **primary** server. You will also need to remove the # character from the start of the line like the following:

/etc/postgresql/12/main/postgresql.conf

. . .

listen\_addresses = 'your\_primary\_IP\_address'

. . .

Save and exit the file.

Once you’re done, your **primary** database will now be ready to accept connections from other machines on the IP address you entered. Next, you’ll create a role with the appropriate permissions that the **replica** will use when connecting to the **primary**.

Step 2 — Creating a Special Role with Replication Permissions

Now, you need to create a role in the **primary** database that has permission to replicate the database. Your **replica** will use this role when connecting to the **primary**. Creating a separate role just for replication also has security benefits. Your **replica** won’t be able to manipulate any data on the **primary**; it will only be able to replicate the data.

First, connect to the database cluster as the postgres user with the following command:

* sudo -u postgres psql

To create a role, you need to run the CREATE ROLE command on the cluster as follows:

* CREATE ROLE test WITH REPLICATION PASSWORD 'testpassword' LOGIN;

You’ll receive the following output:

Output

CREATE ROLE

This command creates a role named test with the password 'testpassword', which has permission to replicate the database cluster.

PostgreSQL has a special replication pseudo-database that the **replica** connects to, but you first need to edit the /etc/postgresql/12/main/pg\_hba.conf configuration file to allow your **replica** to access it. So, exit the PostgreSQL command prompt by running:

* \q

Now that you’re back at your terminal command prompt, open the /etc/postgresql/12/main/pg\_hba.conf configuration file using nano:

* sudo nano /etc/postgresql/12/main/pg\_hba.conf

Append the following line to the end of the pg\_hba.conf file:

/etc/postgresql/12/main/pg\_hba.conf

. . .

host replication test your-replica-IP/32 md5

This ensures that your **primary** allows your **replica** to connect to the replication pseudo-database using the role, test, you created earlier. The host value means to accept non-local connections via plain or SSL-encrypted TCP/IP sockets. replication is the name of the special pseudo-database that PostgreSQL uses for replication. Finally, the value md5 is the type of authentication used. If you want to have more than one **replica**, just add the same line again to the end of the file with the IP address of your other **replica**.

To ensure these changes to the configuration file are implemented, you need to restart the **primary** cluster using:

* sudo systemctl restart postgresql@12-main

If your **primary** cluster restarted successfully, it is correctly set up and ready to start streaming once your **replica** connects. Next, you’ll move on to setting up your **replica** cluster.

Step 3 — Backing Up the Primary Cluster on the Replica

As you are setting up physical replication with PostgreSQL in this tutorial, you need to perform a physical backup of the **primary** cluster’s data files into the **replica’s** data directory. To do this, you’ll first clear out all the files in the **replica’s** data directory. The default data directory for PostgreSQL on Ubuntu is /var/lib/postgresql/12/main/.

You can also find PostgreSQL’s data directory by running the following command on the **replica’s** database:

* SHOW data\_directory;

Once you have the location of the data directory, run the following command to remove everything:

* sudo -u postgres rm -r /var/lib/postgresql/12/main/\*

Since the default owner of the files in the directory is the **postgres** user, you will need to run the command as postgres using sudo -u postgres.

**Note:**  
If in the exceedingly rare case a file in the directory is corrupted and the command does not work, remove the main directory all together and recreate it with the appropriate permissions as follows:

* sudo -u postgres rm -r /var/lib/postgresql/12/main
* sudo -u postgres mkdir /var/lib/postgresql/12/main
* sudo -u postgres chmod 700 /var/lib/postgresql/12/main

Now that the **replica’s** data directory is empty, you can perform a physical backup of the **primary’s** data files. PostgreSQL conveniently has the utility pg\_basebackup that simplifies the process. It even allows you to put the server into standby mode using the -R option.

Execute the pg\_basebackup command on the **replica** as follows:

* sudo -u postgres pg\_basebackup -h primary-ip-addr -p 5432 -U test -D /var/lib/postgresql/12/main/ -Fp -Xs -R
* The -h option specifies a non-local host. Here, you need to enter the IP address of your server with the **primary** cluster.
* The -p option specifies the port number it connects to on the **primary** server. By default, PostgreSQL uses port :5432.
* The -U option allows you to specify the user you connect to the **primary** cluster as. This is the role you created in the previous step.
* The -D flag is the output directory of the backup. This is your **replica’s** data directory that you emptied just before.
* The -Fp specifies the data to be outputted in the plain format instead of as a tar file.
* -Xs streams the contents of the WAL log as the backup of the **primary** is performed.
* Lastly, -R creates an empty file, named standby.signal, in the **replica’s** data directory. This file lets your **replica** cluster know that it should operate as a standby server. The -R option also adds the connection information about the **primary** server to the postgresql.auto.conf file. This is a special configuration file that is read whenever the regular postgresql.conf file is read, but the values in the .auto file override the values in the regular configuration file.

When the pg\_basebackup command connects to the **primary**, you will be prompted to enter the password for the role you created in the previous step. Depending on the size of your **primary** database cluster, it may take some time to copy all the files.

Your **replica** will now have all the data files from the **primary** that it requires to begin replication. Next, you’ll be putting the **replica** into standby mode and start replicating.

Step 4 — Restarting and Testing the Clusters

Now that the **primary** cluster’s data files have been successfully backed up on the **replica**, the next step is to restart the **replica** database cluster to put it into standby mode. To restart the **replica** database, run the following command:

* sudo systemctl restart postgresql@12-main

If your **replica** cluster restarted in standby mode successfully, it should have already connected to the **primary** database cluster on your other machine. To check if the **replica** has connected to the **primary** and the **primary** is streaming, connect to the **primary** database cluster by running:

* sudo -u postgres psql

Now query the pg\_stat\_replication table on the **primary** database cluster as follows:

* SELECT client\_addr, state FROM pg\_stat\_replication;

Running this query on the **primary** cluster will output something similar to the following:

Output

client\_addr | state

------------------+-----------

your\_replica\_IP | streaming

If you have similar output, then the **primary** is correctly streaming to the **replica**.

Conclusion

You now have two Ubuntu 20.04 servers each with a PostgreSQL 12 database cluster running with physical streaming between them. Any changes now made to the **primary** database cluster will also appear in the **replica** cluster.

You can also add more replicas to your setup if your databases need to handle more traffic.

If you wish to learn more about physical streaming replication including how to set up synchronous replication to ensure zero chance of losing any mission-critical data, you can read the entry in the official PostgreSQL [docs](https://www.postgresql.org/docs/current/warm-standby.html).

You can check out our [PostgreSQL topic page](https://www.digitalocean.com/community/tags/postgresql) for more tutorials and content.