How to Setup PostgreSQL Database Replication

Physical PostgreSQL Replication

This is the most common type of replication in PostgreSQL. Physical replication maintains a full copy of the entire data of a cluster. It uses exact block addresses and employs byte-by-byte replication. In simpler terms, the entire set of data on the primary server is copied to the replica which acts as a standby node.

Physical replication does not replicate a specific object of the primary database cluster such as a single row of data in a table. Instead, it works on a disk block level and mirrors all data to replica nodes; including all the tables in each database. This replication requires all replicas to be identical.

Use Cases

It is mostly used for disaster recovery setups and backups since all replicas are identical.

Recommended when dealing with large volumes of data

Pros

- It is easy to implement since all the database clusters are identical.
- It ensures data consistency and high availability at any point since all the replicas hold identical copies of data.
- It is Ideal for read-only operations on the replicas.
- It's very efficient since it does not require any special handling.

Cons

- It is bandwidth-intensive since the entire data is copied and not just small sections of the primary cluster.
- It does not offer multi-master database replication.

Logical PostgreSQL Replication

Logical replication was first introduced in PostgreSQL 9.0. It works by replicating data objects and their changes based on a unique identifier such as a primary key. In simpler terms, logical replication copies database objects in a row-based model as opposed to physical replication which sends everything to the replica nodes.

Thus, logical replication offers fine-grained control over data replication as opposed to physical replication.

Use Cases

Typical use cases for logical replication include:

- Replicating between different major versions of PostgreSQL.
- Replicating between PostgreSQL instances hosted on different platforms e.g from Linux to Windows.
- Sending incremental changes in a database to replicas as they happen in real-time.
- Granting access to replicated data to various groups of users.

Pros

- It is Ideal for backing up incremental data.
- Most recommended for high availability clusters thanks to better performance and low data loss.

- Bandwidth optimization since only row changes for committed transactions of data is sent to the replicas instead of the entire block of data.
- Used in multi-master replication which is not possible using physical replication.
- Supports replication across various OS platforms e.g Linux to windows and vice versa.

Cons

- It cannot stream large volumes of transactions as they happen in real-time.
- The replication process is more complex than physical replication.
- High resource utilization on the replica nodes.

Setup Physical PostgreSQL Replication

Primary IP : 192.168.29.144

Secondary IP : 192.168.29.145

Install PostgreSQL on both Primary and Secondary Node Take note that you need to install the same version of PostgreSQL on both servers.

At the time of writing this guide, the latest version of PostgreSQL

Step 1 : Configure Primary Node :

You need to create a replication user that will be used to initiate the replication process from the primary node.

```
CREATE ROLE replica_user WITH REPLICATION LOGIN PASSWORD 'P@ssword321';
```

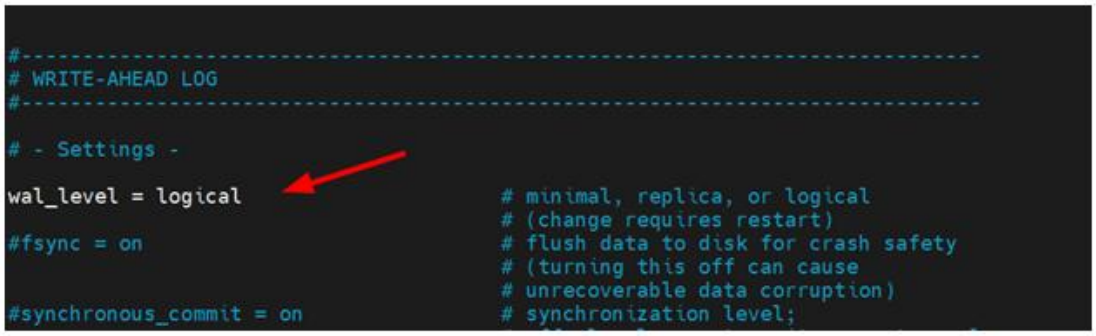
Next, you need to make a few tweaks to the main configuration file. Use your preferred text editor to access the following configuration file:

sudo vi /var/lib/pgsql/data/postgresql.conf

With the file open, scroll down and locate the `listen_addresses` directive. The directive specifies the host under which the PostgreSQL database server listens for connections.

Next, locate the `wal_level` directive. The setting specifies the amount of information to be written to the Write Ahead Log (WAL) file.

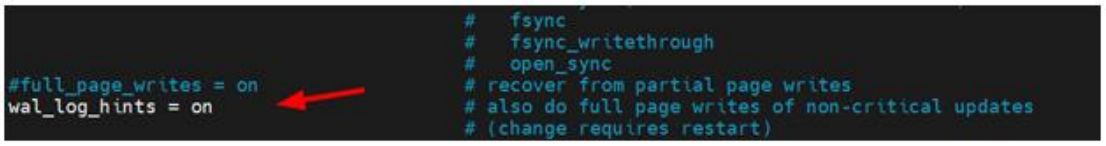
Uncomment the line and set it to `logical` as shown.



Next, locate the `wal_log_hints` directive. By default, it is set to `off`.

When set to `on` the value allows the PostgreSQL server to write the entire content of each disk page to the WAL file during the first modification of the page.

Uncomment it and set it to `on`.



That's all for the changes needed in this configuration file. Save the changes and exit.

Next access /var/lib/pgsql/data/pg_hba.conf configuration file

sudo vi /var/lib/pgsql/data/pg_hba.conf

Append this line at the end of the configuration file. This allows the replica (139.144.169.79) to connect with the master node using the `replica_user.`

```
host    replication    replica_user    192.168.29.145/32    md5
```

Save the changes and close the file. The restart PostgreSQL service.

sudo systemctl restart postgresql

Step 2 : Configure Replica Node :

Before the replica node can start replicating data from the master node, you need to create a copy of the primary node's data directory to the replica's data directory. To achieve this, first, stop the PostgreSQL service on the replica node.

sudo systemctl stop postgresql

Next, remove all files in the replica's data directory in order to start on a clean slate and make room for the primary node data directory.

sudo rm -rv /var/lib/pgsql/data/

Now run the `pg_basebackup` utility as shown to copy data from the primary node to the replica node.

```
sudo pg_basebackup -h 192.168.29.144 -U replica_user -X stream -C -S replica_1 -v -R -W -D /var/lib/pgsql/data
```

```
client@replica:~$
client@replica:~$
client@replica:~$ sudo pg_basebackup -h 194.195.208.82 -U replica_user -X stream -C -S replica_1 -v -R -W -D /var/lib/postgr
esql/14/main/
Password:
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/4000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created replication slot "replica_1"
pg_basebackup: write-ahead log end point: 0/4000100
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed
client@replica:~$
```

Afterward, execute the following command on the replica to grant ownership of the data directory to the postgres user.

```
sudo chown postgres -R /var/lib/pgsql/data
```

Now, start the PostgreSQL server. The replica will now be running in hot standby mode.

Step 3 : Test the Replication Setup

To verify that the replica is connected to the primary node and that the primary is streaming, log into the primary server and switch to the `postgres` user.

sudo -u postgres psql

Next, query the `pg_stat_replication` table which contains vital information about the replication. In this command, we are retrieving information about the replica's IP address and the state of the primary server.

Select client_addr, state from pg_stat_replication ;

```
postgres=#
postgres=# SELECT client_addr, state FROM pg_stat_replication;
  client_addr   |   state
----------------+-----------
 139.144.169.79 | streaming
(1 row)
```

You should get the following output confirmation that your setup is working.