Setting Up PostgreSQL Replication: A Step-by-Step Guide

Introduction

PostgreSQL is a versatile open-source relational database management system renowned for its performance and robustness. Setting up replication between a master and a slave server is a crucial step in ensuring high availability and data redundancy in a PostgreSQL environment. In this guide, we'll walk you through the process of installing PostgreSQL on a slave server and configuring replication to replicate data from a master server.

Prerequisites

Before you begin, make sure you have:

- A slave server running a compatible operating system (in this guide, we use CentOS 7).
- Root or sudo access on the slave server.
- Basic familiarity with PostgreSQL concepts.

Step 1: Install PostgreSQL on the Slave Server

1. Install the PostgreSQL repository RPM:

```
sudo yum install -y
https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-
redhat-repo-latest.noarch.rpm
```

2. Install PostgreSQL server and client packages:

```
sudo yum install -y postgresql12-server postgresql-12
```

Step 2: Initialize and Start PostgreSQL on the Slave Server

1. Initialize the database and enable automatic start:

```
sudo /usr/pgsql-12/bin/postgresql-12-setup initdb
sudo systemctl enable postgresql-12
sudo systemctl start postgresql-12
systemctl status postgresql-12
```

Step 3: Configure the Master Database

1. Create a replication user on the master:

```
CREATE USER replicator WITH REPLICATION ENCRYPTED PASSWORD 'replica';
```

Allow the slave server to connect to the master by modifying `pg_hba.conf` on the master:

```
psql -c "show hba_file;"
# Navigate to the directory and append the following line to pg_hba.conf:
echo "host replication replicator 192.168.1.100/32 md5" >> pg_hba.conf
# Reload the configuration:
psql -c "select pg_reload_conf();"
```

Step 4: Configure the Slave Database

1. Verify connectivity from the slave to the master:

```
telnet 192.168.1.101 5432
```

Copy data directory files from the master:

```
cp -rp /var/lib/pgsql/12/data /var/lib/pgsql/12/data_orig
rm -rf /var/lib/pgsql/12/data
```

Change ownership of the PostgreSQL directory to match the master:

```
chown -R postgres:postgres /var/lib/pgsql
chmod -R 775 /var/lib/pgsql
```

Empty specific directories on the slave:

```
rm -rf /var/lib/pgsql/pg_tablespace_02/*
rm -rf /var/lib/pgsql/pg_tablespace_01/*
rm -rf /var/lib/pgsql/pg_wal/*
```

Copy files from the master using `pg_basebackup`:

```
pg_basebackup -h 192.168.1.101 -D /var/lib/pgsql/12/data -U replicator -P -v
-R -X stream
```

Here’s a brief explanation of the command-line options used in the `pg_basebackup` command:

- `-h`: Specifies the hostname or IP address of the master server from which to take the backup.
- `-D`: Specifies the directory where the backup data will be stored on the local machine (slave server).
- `-U`: Specifies the username to use when connecting to the master server for replication.
- `-P`: Prompts for the replication password of the specified user. After entering the password, the backup process begins.
- `-v`: Enables verbose output, providing more detailed information about the backup process.
- `-R`: Enables the creation of a recovery.conf file in the backup directory. This is essential for setting up a standby server for replication.
- `-X stream`: Specifies that the backup method should use streaming replication, allowing continuous replication from the master to the standby server.

Back up important configuration files on the slave:

```
cd /var/lib/pgsql/12/data
mv pg_hba.conf pg_hba_backup.conf
mv postgresql.auto.conf postgresql_auto_backup.conf
mv postgresql.conf postgresql_backup.conf
```

Replace the backed-up files with those from another standby or the master.

Start the PostgreSQL service on the slave:

```
systemctl start postgresql-12
```

Monitor replication status:

```
psql -c "\x" -c "SELECT * FROM pg_stat_wal_receiver;"
```

Test Scenario: Validating Replication

Now that we've successfully set up replication between the master and the slave server, let's perform a simple test to ensure that data changes made on the primary (master) server are correctly replicated to the standby (slave) server.

1. Create a Database and Table on Primary:
   Connect to the primary server's PostgreSQL instance using a database management tool or the command line. Then, create a new database and a test table:

```
CREATE DATABASE test_replication;
\c test_replication
CREATE TABLE test_table (id serial primary key, value text);
```

2. Insert Test Records:

INSERT INTO test_table (value) VALUES (‘Record 1’);
INSERT INTO test_table (value) VALUES (‘Record 2’);

3. Check Slave Server:

On the slave server, connect to its PostgreSQL instance and the replicated database:

```
psql -U postgres -d test_replication
SELECT * FROM test_table;
```

You should see the same records that were inserted on the primary server.

Conclusion

By following these steps, you have successfully set up PostgreSQL replication on your slave server. This replication configuration enhances data availability and redundancy, ensuring the continuity of your database services even in the face of failures.

Remember that a well-managed PostgreSQL environment requires regular monitoring, maintenance, and backups. With this guide, you’re now equipped to build a robust and highly available PostgreSQL setup that can handle your organization’s critical data with confidence.

Feel free to explore advanced topics like failover, monitoring tools, and load balancing to further enhance your PostgreSQL deployment. Happy data managing!