Continuous Archiving and Point-In-Time-Recovery

Introduction

PostgreSQL is a widely used relational database that supports ACID transactions. The acronym ACID stands for atomicity, consistency, isolation, and durability. These are four key properties of database transactions that PostgreSQL supports to ensure the persistence and validity of data in the database.
One method PostgreSQL uses to maintain ACID properties is Write-Ahead Logging (WAL).
PostgreSQL first records any transaction on the database to the WAL log files before it writes the changes to the database cluster's data files.
With continuous archiving, the WAL files are copied to secondary storage, which has a couple of benefits. For example, a secondary database cluster can use the archived WAL file for replication purposes, but you can also use the files to perform point-in-time-recovery (PITR). That is, you can use the files to rollback a database cluster to a desirable point if an accident happens.
In this guideline, you will learn how to set up continuous archiving with a PostgreSQL 14 cluster on Ubuntu 20.04 and perform PITR on the cluster.

Getting Started

Step 1 - Configuring Continuous Archiving on the Database Cluster

In this first step, you need to configure your PostgreSQL 14 cluster to archive the cluster's WAL files in a directory different from the cluster's data directory. To do this, you must first create a new directory somewhere to archive the WAL files.

Create a new directory as follows:

```
$ mkdir database_archive
```

You now need to give the default PostgreSQL user, postgres, permission to write to this directory. You can achieve this by changing the ownership of the directory using the chown command:

```
$ sudo chown postgres:postgres database_archive
```

Now that you have a directory set up for the cluster to archive the WAL files into, you must
enable archiving in the postgresql.conf configuration file, which you can find in the /etc/postgresql/14/main/ directory by default.

Open the configuration file with your text editor:

```
$ sudo nano /etc/postgresql/14/main/postgresql.conf
```

Once you have opened the file, you'll uncomment the line with the archive_mode variable on it by removing the # from the start of the line. Also, change the value of archive_mode to on like the following:

/etc/postgresql/14/main/postgresql.conf

```
archive_mode = on

archive_command = 'test ! -f /path/to/database_archive/%f && cp %p /path/to/database_archive/%f'

Wal_level=replica

Archive_timeout=60
```

The archive command here first checks to see if the WAL file already exists in the archive, and if it doesn't, it copies the WAL file to the archive.

Replace the /path/to/database_archive with the path to the database_archive directory you
created earlier. For example, if you created this in your home directory: ~/database_archive.

Lastly, you need to configure the wal_level variable. wal_level dictates how much information
PostgreSQL writes to the log. For continuous archiving, this needs to be set to at least replica:

This is already the default value in PostgreSQL 14, so you shouldn't need to change it, but it is
something to remember if you ever go to change this variable.

You can now save and exit your file.

To implement the changes to your database cluster configuration file, you need to restart the cluster as follows:

```
$ sudo systemctl restart postgresql@14-main
```

If PostgreSQL restarts successfully, the cluster will archive every WAL file once it is full. By default, each WAL file is 16MB.

In the case that you need to archive a transaction immediately, you can force the database
cluster to change and archive the current WAL file by running the following command on the
cluster:

```
$ sudo -u postgres psql -c "SELECT pg_switch_wal();"
```

With the database cluster successfully copying the WAL files to the archive, you can now perform a physical backup of the database cluster's data files.

Step 2 Create Some Database And Insert Some Records

```
$ psql -u postgres -d postgres -h localhost -p 5432

create database test_db1;

create table test_tbl1(id int,name varchar(50));

insert into test_tbl1 values select generate_series(1,10) as id,md5(random()::text) as descr;

select count(*) from test_tbl1;

select now();

select pg_switch_wal();
```

Step 3 Performing a Physical Backup of the PostgreSQL Cluster

It is important to take regular backups of your database to help mitigate data loss should the worst happen. PostgreSQL allows you to take both logical and physical backups of the database cluster. However, for PITR,

you need to take a physical backup of the database cluster. That is, you need to make a copy of all the database's files in PostgreSQL's data directory. By default, the PostgreSQL 14 data directory is /var/lib/postgresql/14/main/.

In the previous step, you made the directory, database_archive, to store all the archived WAL files. In this step you need to create another directory, called database_backup, to store the physical backup you will take. Once again, make the directory:

```
$ mkdir database_backup
```

Now ensure that the postgres user has permission to write to the directory by changing the ownership:

```
$ sudo chown postgres:postgres database_backup
```

Now that you have a directory for the backup, you need to perform a physical backup of the database cluster's data files. Fortunately, PostgreSQL has the built-in pg_basebackup command that performs everything for you. Run the command as the postgres user:

```
$ sudo -u postgres pg_basebackup -D /path/to/database_backup
```

Replace /path/to/ with the path to your directory. With this physical backup of the database cluster, you are now able to perform point-in-time-recovery on the cluster.

Till this Time we have a backup upto fist 10 records only . Let’s Insert 20 more records .

insert into test_tbl1 values select generate_series(1,10) as id,md5(random()::text) as descr;

select count(*) from test_tbl1;

select now();

Now we have 20 records and Let’s note that time as well for PITR. Let’s insert 10 more records .

insert into test_tbl1 values select generate_series(1,10) as id,md5(random()::text) as descr;

select count(*) from test_tbl1;

select now();

So, Till this time we have 30 records, backup up to only first 10 records and we want to do PITR till first 20 records that’s why we need time after first 20 record inserted.

Step 4 Performing Point-In-Time-Recovery on the Database Cluster

Now that you have at least one physical backup of the database and you’re archiving the WAL files, you can now perform PITR, if you need to rollback the database to a previous state.
First, if the database is still running, you’ll need to shut it down. You can do this by running the systemctl stop command:

```
$ sudo systemctl stop postgresql@14-main
```

Once the database is no longer running, you need to remove all the files in PostgreSQL's data directory. But first, you need to move the pg_wal directory to a different place as this might contain unarchived WAL files that are important for recovery. Use the mv command to move the pg_wal directory as follows:

```
$ sudo mv /var/lib/postgresql/14/main/pg_wal ~/
```

Now, you can remove the /var/lib/postgresql/14/main directory entirely and recreate it as such:

```
$ sudo rm -rf /var/lib/postgresql/14/main
```

Followed by:

```
$ sudo mkdir /var/lib/postgresql/14/main
```

Now, you need to copy all the files from the physical backup you made in the previous step to the new empty data directory. You can do this with cp:

```
$ sudo cp -a /path/to/database_backup/. /var/lib/postgresql/14/main/
```

You also need to ensure the data directory has the postgres user as the owner and the appropriate permissions. Run the following command to change the owner:

```
$ sudo chown postgres:postgres /var/lib/postgresql/14/main
```

And update the permissions:

```
$ sudo chmod 700 /var/lib/postgresql/14/main
```

The WAL files in the pg_wal directory copied from the physical backup are outdated and not useful. You need to replace them with the WAL files in the pg_wal directory that you copied before you emptied out the PostgreSQL's data directory as some of the files might not have been archived before stopping the server.

Remove the pg_wal file in the /var/lib/postgresql/14/main directory as follows:

```
$ sudo rm -rf /var/lib/postgresql/14/main/pg_wal
```

Now copy the files from the pg_wal directory you saved before clearing out the data directory:

```
$ sudo cp -a ~/pg_wal /var/lib/postgresql/14/main/pg_wal
```

With the data directory restored correctly, you need to configure the recovery settings to ensure the database server recovers the archived WAL files correctly. The recovery settings are found in the postgresql.conf configuration file in the /etc/postgresql/14/main/ directory.

Open the configuration file:

```
$ sudo nano /etc/postgresql/14/main/postgresql.conf
```

Once you have the file open, locate the restore_command variable and remove the # character from the start of the line. Just like you did with

archive_command in the first step, you need to specify how PostgreSQL should recover the WAL files. Since the archive command just copies the files to the archive, the restore command will copy the files back. The restore_command variable will be similar to the following:

```
restore_command = 'cp /path/to/database_archive/%f  %p'

recovery_target_time = '2022-06-04 19:55:37'

# Edit recovery_target_time  according to your needs.
```

Next, you have the option to specify a recovery target. This is the point that the database cluster will try to recover to before leaving recovery mode. The recovery target can be a timestamp, transaction ID, log sequence number, the name of a restore point created with the pg_create_restore_point() command, or whenever the database reaches a consistent state. If no recovery target is specified, the database cluster will read through the entire log of WAL files in the archive.

Note: The recovery target must be a point in time after the physical backup you are using was taken. If you need to return to an earlier point, then you need to use an earlier backup of the database.

Once you have set restore_command and recovery_target_time, save and exit the file.

Before restarting the database cluster, you need to inform PostgreSQL that it should start in recovery mode. You can achieve this by creating an empty file in the cluster's data directory called recovery.signal. To create an empty file in the directory, use the touch command:

```
$ sudo touch /var/lib/postgresql/14/main/recovery.signal
```

And edit recovery.signal file like this:

```
restore_command = 'cp /path/to/database_archive/%f  %p'

recovery_target_time = '2022-06-04 19:55:37'
```

Now you can restart the database cluster by running:

```
$ sudo systemctl start postgresql@14-main
```

If the database started successfully, it will enter recovery mode. Once the database cluster reaches the recovery target, it will remove the recovery.signal file.

Now that you have successfully recovered your database cluster to the desired state, you can begin your normal database operations. If you want to recover to a different point in time, you can repeat this step.

After the database started successfully , it is on the read only mode. So, we have change on read/write mode like this:-

```
select pg_wal_replay_resume();
```

and test the desired results.

```
Select count(*) from test_tbl1;
```

Thank You!!!