PostgreSQL Backups and Recoveries

1. Logical Backup (pg_dump and pg_dumpall)

Logical backups are ideal for smaller databases or when you need a backup of specific tables or schemas.

a) pg_dump – Backup of a Single Database

Use `pg_dump` for taking logical backups of an individual database. This is ideal for backing up specific databases or tables.

```
pg_dump -U postgres -d sai_db -F c -f /postgres/backup/sai_db_backup.sql
```

- **-U**: Specifies the PostgreSQL user.
- **-d**: Specifies the database name (`sai_db`).
- **-F c**: Defines the custom format for the backup.
- **-f**: Specifies the backup file location.

b) pg_dump – Backup of a Specific Table

To backup only `sai_table` in the schema `sainath`:

```
pg_dump -U postgres -d sai_db -t sainath.sai_table -F c -f
/postgres/backup/sai_table_backup.sqlc
```

c) pg_dumpall – Backup of All Databases

`pg_dumpall` is useful when you need a backup of the entire PostgreSQL cluster, including global objects like roles and tablespaces.

```
pg_dumpall -U postgres -f /postgres/backup/pg_cluster_backup.sql
```

2. Physical Backup (pg_basebackup)

To perfor Point-In-Time Recovery (PITR) without `recovery.conf` using a base backup and archived WAL files in PostgreSQL 16, we'll take a `pg_basebackup` in `/postgres/sainath/backup`, then restore it in a new PostgreSQL data directory (`/postgres/sainath/data`). Here's a step-by-step guide:

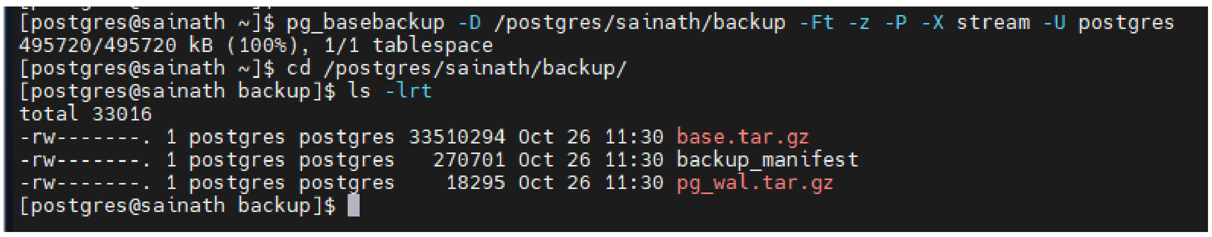
Step 1: Take a Base Backup with `pg_basebackup`

First, ensure PostgreSQL is running, and take a base backup. This will capture the current state of the database, which will be used for PITR.

```
mkdir -p /postgres/sainath/backup
```

```
pg_basebackup -D /postgres/sainath/backup -Ft -z -P -X stream -U postgres
```

- **-D**: Specifies the destination directory for the base backup (`/postgres/sainath/backup`).
- **-Ft**: Creates the backup in tar format.
- **-z**: Compresses the backup.
- **-P**: Shows progress during the backup.
- **-X stream**: Streams WAL files along with the base backup to ensure a consistent state.
- **-U**: Specifies the PostgreSQL user.



```
[postgres@sainath ~]$ pg_basebackup -D /postgres/sainath/backup -Ft -z -P -X stream -U postgres
495720/495720 kB (100%), 1/1 tablespace
[postgres@sainath ~]$ cd /postgres/sainath/backup/
[postgres@sainath backup]$ ls -lrt
total 33016
-rw-------. 1 postgres postgres 33510294 Oct 26 11:30 base.tar.gz
-rw-------. 1 postgres postgres   270701 Oct 26 11:30 backup_manifest
-rw-------. 1 postgres postgres    18295 Oct 26 11:30 pg_wal.tar.gz
[postgres@sainath backup]$
```

Step 2: Set Up WAL Archiving (if Not Already Enabled)

To perform PITR, you need WAL archiving enabled. In `postgresql.conf`, configure the following:

```
archive_mode = on
archive_command = 'cp %p /postgres/sainath/archive/%f'
wal_level = replica
```

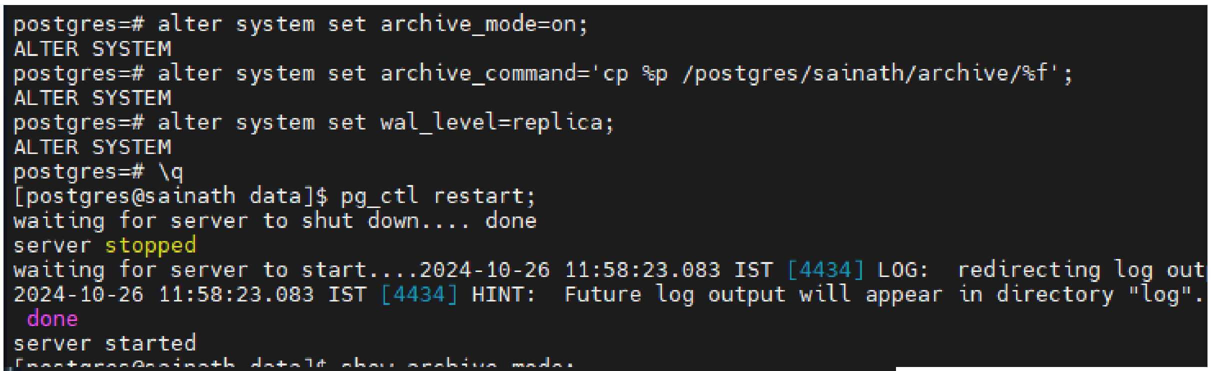
- **archive_mode**: Enables WAL archiving.
- **archive_command**: Copies WAL files to the specified archive directory (replace `/home/PG/archive/` with your preferred archive path).
- **wal_level**: Set to at least `replica` to support archiving.

Restart PostgreSQL after updating these settings

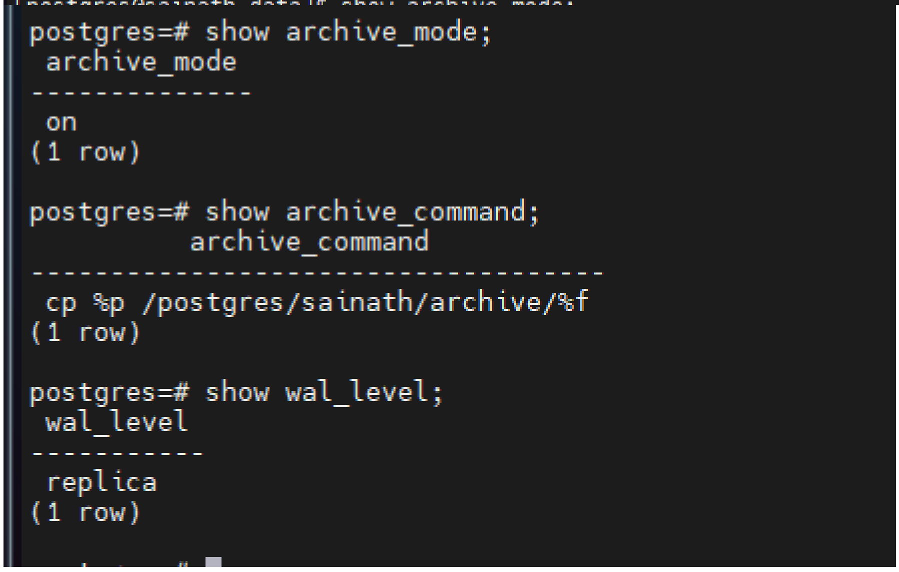
```
sudo systemctl restart postgresql
```

or

we can change the parameters by using Alter system set also :



```
postgres=# alter system set archive_mode=on;
ALTER SYSTEM
postgres=# alter system set archive_command='cp %p /postgres/sainath/archive/%f';
ALTER SYSTEM
postgres=# alter system set wal_level=replica;
ALTER SYSTEM
postgres=# \q
[postgres@sainath data]$ pg_ctl restart;
waiting for server to shut down.... done
server stopped
waiting for server to start....2024-10-26 11:58:23.083 IST [4434] LOG:  redirecting log out
2024-10-26 11:58:23.083 IST [4434] HINT:  Future log output will appear in directory "log".
 done
server started
```



```
postgres=# show archive_mode;
 archive_mode
--------------
 on
(1 row)

postgres=# show archive_command;
          archive_command
------------------------------------
 cp %p /postgres/sainath/archive/%f
(1 row)

postgres=# show wal_level;
 wal_level
-----------
 replica
(1 row)
```

Step 3: Stop the PostgreSQL Server

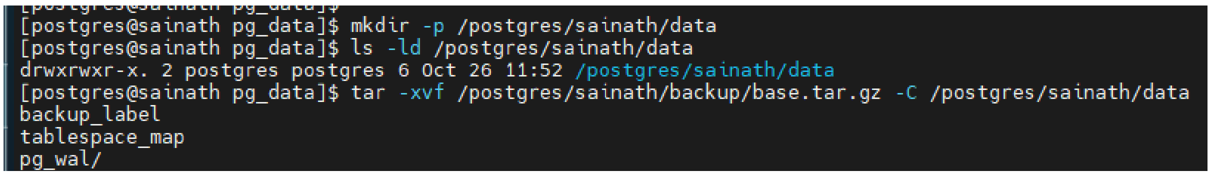
Stop the PostgreSQL server to prepare for restoring from the base backup in the new data directory.

```
sudo systemctl stop postgresql
```

Step 4: Restore the Base Backup to the New Data Directory

Extract the base backup taken in `/postgres/sainath/backup` to the new data directory (`/postgres/sainath/data`):

```
mkdir -p /postgres/sainath/data
tar -xvf /postgres/sainath/backup/base.tar.gz -C /postgres/sainath/data
```



```
[postgres@sainath pg_data]$ mkdir -p /postgres/sainath/data
[postgres@sainath pg_data]$ ls -ld /postgres/sainath/data
drwxrwxr-x. 2 postgres postgres 6 Oct 26 11:52 /postgres/sainath/data
[postgres@sainath pg_data]$ tar -xvf /postgres/sainath/backup/base.tar.gz -C /postgres/sainath/data
backup_label
tablespace_map
pg_wal/
```

Step 5: Configure `postgresql.conf` for PITR in the New Data Directory

In the new data directory **(`/postgres/sainath/data`)**, edit **`postgresql.conf`** to configure the recovery parameters:

```
restore_command = 'cp /postgres/sainath/archive/%f %p'
recovery_target_time = '2024-10-25 10:30:00'  # Specify your desired
recovery point in time here
```

- **restore_command**: Specifies how PostgreSQL retrieves archived WAL files. Replace `/path/to/archive/` with the actual location where WAL files are stored.
- **recovery_target_time**: Specifies the exact point in time you want to restore the database, in `YYYY-MM-DD HH:MM:SS` format.

Or

- • **Log Sequence Number (LSN)** to which the recovery should proceed.
- • You can find LSNs in the PostgreSQL logs or via SQL queries.
-
- recovery_target_lsn = '0/3000000'

Step 6: Create a `recovery.signal` File

In the new data directory (`/postgres/sainath/data`), create an empty `recovery.signal` file. This file instructs PostgreSQL to enter recovery mode and stop at the specified recovery target.

```
touch /postgres/sainath/data/recovery.signal
```

Step 7: Start PostgreSQL Using the New Data Directory

Now start PostgreSQL, pointing it to the new data directory. PostgreSQL will enter recovery mode, apply the archived WAL logs, and stop at the specified recovery time.

```
pg_ctl -D /postgres/sainath/data -l logfile start
```

- **-D**: Specifies the new data directory (`/postgres/sainath/data`).
- **-l logfile**: Logs output to a file for monitoring purposes.

Step 8: Monitor Recovery Progress

Check the PostgreSQL logs (or the `logfile` specified) to monitor the recovery progress. PostgreSQL will apply the archived WAL files and stop at the specified **`recovery_target_time`.**

Step 9: Post-Recovery Cleanup

Once the recovery is complete, remove the `recovery.signal` file to prevent PostgreSQL from re-entering recovery mode on the next restart.

```
rm /postgres/sainath/data/recovery.signal
```

Step 10: Restart PostgreSQL to Resume Normal Operation

Restart PostgreSQL to resume normal operation in the new data directory.

```
pg_ctl -D /postgres/sainath/data -l logfile restart
```

Scheduling Backups with Cron

To automate backups, schedule them using `cron`. Here's an example of a daily logical backup at midnight.

```
0 0 * * * pg_dump -U postgres -d sai_db -F c -f
/postgres/backup/sai_db_backup_$(date +\%F).sql
```

This command schedules a daily backup with a timestamped filename.