# **PostgreSQL 17: How to Backup and Restore Using**pg\_basebackup**on the Same Host**

[](https://medium.com/@jramcloud1?source=post_page---byline--107d1c63ccbd---------------------------------------)

[Jeyaram Ayyalusamy](https://medium.com/@jramcloud1?source=post_page---byline--107d1c63ccbd---------------------------------------)

Following

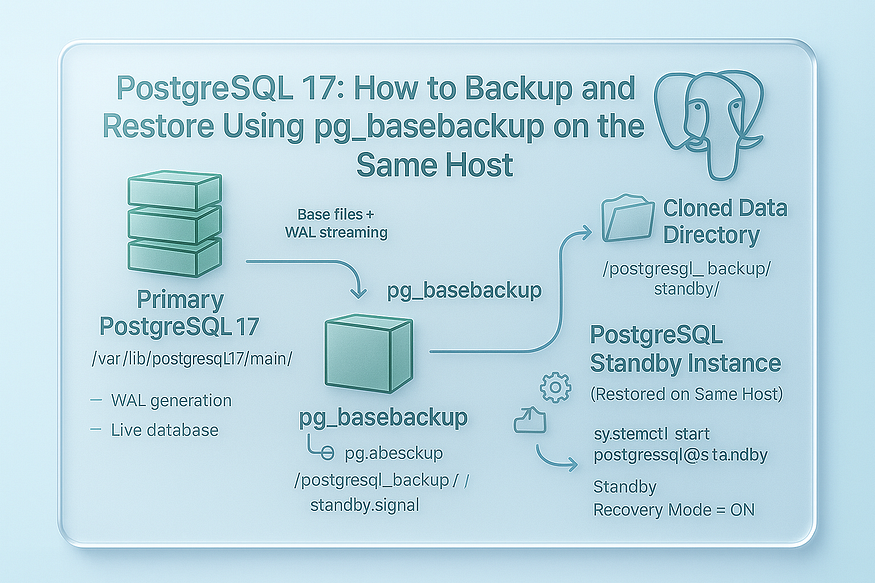
28 min read

·

Jun 30, 2025

51

Press enter or click to view image in full size



## **🛡 PostgreSQL 17: How to Backup and Restore Using**pg\_basebackup**on the Same Host**

When managing PostgreSQL databases, ensuring a ****reliable, consistent, and fast backup strategy**** is essential for any production environment. One of the most efficient tools available natively within PostgreSQL is pg\_basebackup.

This guide walks you through everything you need to know about using pg\_basebackup to perform a ****full backup and restore**** operation ****on the same host****. Whether you're preparing for disaster recovery, building replication, or setting up Point-in-Time Recovery (PITR), pg\_basebackup should be part of your toolkit.

## **🔎 Why Use**pg\_basebackup**?**

pg\_basebackup is a PostgreSQL-native utility designed to take ****binary-level, physical backups**** of the database cluster. Here's why it's so widely recommended:

## **✅ 1. Zero Downtime Backups**

You can run pg\_basebackup while PostgreSQL is actively running. Thanks to PostgreSQL’s Write-Ahead Logging (WAL) system, the tool can stream all changes happening during the backup to ensure consistency.

No need to stop your database. Your applications and users won’t even notice.

## **✅ 2. PITR & Replication-Ready**

When used with WAL streaming (-X flag) or WAL archiving, pg\_basebackup enables:

* ****Point-in-Time Recovery**** (PITR): Rewind your database to any moment after the backup.
* ****Standby Initialization****: The backup can be used to set up a streaming replica (standby node) for high availability.

It automatically generates the standby.signal file (if using -R flag), preparing the backup directory for replication mode.

## **✅ 3. Cluster-Wide Coverage**

Unlike pg\_dump, which backs up ****individual databases**** (logical backup), pg\_basebackup backs up the ****entire cluster**** including:

* All user databases
* Global objects (roles, tablespaces)
* Configuration files
* WAL segments
* Replication metadata

This is a ****complete snapshot**** of everything inside your PostgreSQL instance.

## **✅ 4. Perfect for Cloud-Based PostgreSQL**

Running PostgreSQL on ****AWS EC2, Azure VM, GCP Compute Engine****, or on-premise Linux servers? pg\_basebackup is ideal because:

* It can be automated with cron or shell scripts.
* It doesn’t require third-party software.
* It can be used as part of an HA or DR setup with minimal configuration.

Whether you’re backing up a 1GB test instance or a multi-terabyte production node, pg\_basebackup scales gracefully.

## **🔧 Prerequisites Before Using**pg\_basebackup

Before jumping into backup/restore steps, make sure your system meets the following conditions:

## **✅ PostgreSQL 17 Installed & Running**

Check if PostgreSQL is installed and running:

psql --version  
sudo systemctl status postgresql-17

You should see something like:

[postgres@ip-172-31-20-155 ~]$ psql --version  
psql (PostgreSQL) 17.5  
[postgres@ip-172-31-20-155 ~]$  
  
  
[postgres@ip-172-31-20-155 ~]$ sudo systemctl status postgresql-17  
● postgresql-17.service - PostgreSQL 17 database server  
 Loaded: loaded (/usr/lib/systemd/system/postgresql-17.service; enabled; preset: disabled)  
 Active: active (running) since Fri 2025-06-27 17:00:27 UTC; 7h ago  
 Invocation: 0594acf74fad4411b78da3700cb1e647  
 Docs: https://www.postgresql.org/docs/17/static/  
 Process: 74444 ExecStartPre=/usr/pgsql-17/bin/postgresql-17-check-db-dir ${PGDATA} (code=exited, status=0/SUCCESS)  
 Main PID: 74450 (postgres)  
 Tasks: 8 (limit: 5687)  
 Memory: 37.7M (peak: 41.3M)  
 CPU: 13.578s  
 CGroup: /system.slice/postgresql-17.service  
 ├─74450 /usr/pgsql-17/bin/postgres -D /var/lib/pgsql/17/data/  
 ├─74451 "postgres: logger "  
 ├─74452 "postgres: checkpointer "  
 ├─74453 "postgres: background writer "  
 ├─74455 "postgres: walwriter "  
 ├─74456 "postgres: autovacuum launcher "  
 ├─74457 "postgres: archiver failed on 000000010000000200000035"  
 └─74458 "postgres: logical replication launcher "  
  
Jun 27 17:00:27 ip-172-31-20-155.ec2.internal systemd[1]: Starting postgresql-17.service - PostgreSQL 17 database server...  
Jun 27 17:00:27 ip-172-31-20-155.ec2.internal postgres[74450]: 2025-06-27 17:00:27.611 UTC [74450] LOG: redirecting log output to logging collector process  
Jun 27 17:00:27 ip-172-31-20-155.ec2.internal postgres[74450]: 2025-06-27 17:00:27.611 UTC [74450] HINT: Future log output will appear in directory "log".  
Jun 27 17:00:27 ip-172-31-20-155.ec2.internal systemd[1]: Started postgresql-17.service - PostgreSQL 17 database server.  
[postgres@ip-172-31-20-155 ~]$

## **✅ Sample Database (Optional)**

To verify the effectiveness of your backup, it’s useful to load a sample database like dvdrental.

Install wget in you linux machine

[postgres@ip-172-31-20-155 ~]$ sudo yum install wget -y  
Updating Subscription Management repositories.  
Unable to read consumer identity  
  
This system is not registered with an entitlement server. You can use "rhc" or "subscription-manager" to register.  
  
Last metadata expiration check: 3:16:56 ago on Fri Jun 27 21:10:15 2025.  
Dependencies resolved.  
====================================================================================================================================================================================================================  
 Package Architecture Version Repository Size  
====================================================================================================================================================================================================================  
Installing:  
 wget x86\_64 1.24.5-5.el10 rhel-10-appstream-rhui-rpms 807 k  
  
Transaction Summary  
====================================================================================================================================================================================================================  
Install 1 Package  
  
Total download size: 807 k  
Installed size: 3.3 M  
Downloading Packages:  
wget-1.24.5-5.el10.x86\_64.rpm 15 MB/s | 807 kB 00:00  
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------  
Total 9.7 MB/s | 807 kB 00:00  
Running transaction check  
Transaction check succeeded.  
Running transaction test  
Transaction test succeeded.  
Running transaction  
 Preparing : 1/1  
 Installing : wget-1.24.5-5.el10.x86\_64 1/1  
 Running scriptlet: wget-1.24.5-5.el10.x86\_64 1/1  
Installed products updated.  
  
Installed:  
 wget-1.24.5-5.el10.x86\_64  
  
Complete!  
[postgres@ip-172-31-20-155 ~]$

You can download it from [https://neon.com/postgresql/postgresql-getting-started/postgresql-sample-database](https://neon.com/postgresql/postgresql-getting-started/postgresql-sample-database" \t "https://medium.com/@jramcloud1/_blank) and restore it using:

wget https://neon.com/postgresqltutorial/dvdrental.zip

Output:

[postgres@ip-172-31-20-155 ~]$ wget https://neon.com/postgresqltutorial/dvdrental.zip  
--2025-06-28 00:28:58-- https://neon.com/postgresqltutorial/dvdrental.zip  
Resolving neon.com (neon.com)... 76.76.21.21  
Connecting to neon.com (neon.com)|76.76.21.21|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 550906 (538K) [application/zip]  
Saving to: ‘dvdrental.zip’  
  
dvdrental.zip 100%[======================================================================================================================>] 537.99K --.-KB/s in 0.005s  
  
2025-06-28 00:28:58 (113 MB/s) - ‘dvdrental.zip’ saved [550906/550906]  
  
[postgres@ip-172-31-20-155 ~]$  
  
  
[postgres@ip-172-31-20-155 ~]$ ls -ltr  
total 540  
drwx------. 4 postgres postgres 51 Jun 21 16:18 17  
-rw-r--r--. 1 postgres postgres 550906 Jun 27 19:08 dvdrental.zip  
[postgres@ip-172-31-20-155 ~]$  
  
[postgres@ip-172-31-20-155 ~]$ unzip dvdrental.zip  
Archive: dvdrental.zip  
 inflating: dvdrental.tar  
  
  
[postgres@ip-172-31-20-155 ~]$ ls -ltr  
total 3312  
-rw-r--r--. 1 postgres postgres 2835456 May 12 2019 dvdrental.tar  
drwx------. 4 postgres postgres 51 Jun 21 16:18 17  
-rw-r--r--. 1 postgres postgres 550906 Jun 27 19:08 dvdrental.zip

Create dvdrental database and restore

createdb dvdrental  
  
  
postgres=# \l+  
 List of databases  
 Name | Owner | Encoding | Locale Provider | Collate | Ctype | Locale | ICU Rules | Access privileges | Size | Tablespace | Description  
-------------+----------+----------+-----------------+---------+---------+--------+-----------+-----------------------+---------+------------+--------------------------------------------  
 dvdrental | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | | 7441 kB | pg\_default |  
 metrics\_lab | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | | 7979 kB | pg\_default |  
 postgres | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | | 1239 MB | pg\_default | default administrative connection database  
 template0 | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | =c/postgres +| 7361 kB | pg\_default | unmodifiable empty database  
 | | | | | | | | postgres=CTc/postgres | | |  
 template1 | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | =c/postgres +| 7595 kB | pg\_default | default template for new databases  
 | | | | | | | | postgres=CTc/postgres | | |  
(5 rows)  
  
postgres=#  
  
  
  
  
[postgres@ip-172-31-20-155 ~]$  
[postgres@ip-172-31-20-155 ~]$ pg\_restore -d dvdrental /var/lib/pgsql/dvdrental.tar  
[postgres@ip-172-31-20-155 ~]$

pg\_restore -d dvdrental -v /var/lib/pgsql/dvdrental.tar

This will populate the server with realistic test data to validate backup integrity.

[postgres@ip-172-31-20-155 ~]$ pg\_restore -d dvdrental -v /var/lib/pgsql/dvdrental.tar  
pg\_restore: connecting to database for restore  
pg\_restore: creating TYPE "public.mpaa\_rating"  
pg\_restore: creating DOMAIN "public.year"  
pg\_restore: creating FUNCTION "public.\_group\_concat(text, text)"  
pg\_restore: creating FUNCTION "public.film\_in\_stock(integer, integer)"  
pg\_restore: creating FUNCTION "public.film\_not\_in\_stock(integer, integer)"  
pg\_restore: creating FUNCTION "public.get\_customer\_balance(integer, timestamp without time zone)"  
pg\_restore: creating FUNCTION "public.inventory\_held\_by\_customer(integer)"  
pg\_restore: creating FUNCTION "public.inventory\_in\_stock(integer)"  
pg\_restore: creating FUNCTION "public.last\_day(timestamp without time zone)"  
pg\_restore: creating FUNCTION "public.last\_updated()"  
pg\_restore: creating SEQUENCE "public.customer\_customer\_id\_seq"  
pg\_restore: creating TABLE "public.customer"  
pg\_restore: creating FUNCTION "public.rewards\_report(integer, numeric)"  
pg\_restore: creating AGGREGATE "public.group\_concat(text)"  
pg\_restore: creating SEQUENCE "public.actor\_actor\_id\_seq"  
pg\_restore: creating TABLE "public.actor"  
pg\_restore: creating SEQUENCE "public.category\_category\_id\_seq"  
pg\_restore: creating TABLE "public.category"  
pg\_restore: creating SEQUENCE "public.film\_film\_id\_seq"  
pg\_restore: creating TABLE "public.film"  
pg\_restore: creating TABLE "public.film\_actor"  
pg\_restore: creating TABLE "public.film\_category"  
pg\_restore: creating VIEW "public.actor\_info"  
pg\_restore: creating SEQUENCE "public.address\_address\_id\_seq"  
pg\_restore: creating TABLE "public.address"  
pg\_restore: creating SEQUENCE "public.city\_city\_id\_seq"  
pg\_restore: creating TABLE "public.city"  
pg\_restore: creating SEQUENCE "public.country\_country\_id\_seq"  
pg\_restore: creating TABLE "public.country"  
pg\_restore: creating VIEW "public.customer\_list"  
pg\_restore: creating VIEW "public.film\_list"  
pg\_restore: creating SEQUENCE "public.inventory\_inventory\_id\_seq"  
pg\_restore: creating TABLE "public.inventory"  
pg\_restore: creating SEQUENCE "public.language\_language\_id\_seq"  
pg\_restore: creating TABLE "public.language"  
pg\_restore: creating VIEW "public.nicer\_but\_slower\_film\_list"  
pg\_restore: creating SEQUENCE "public.payment\_payment\_id\_seq"  
pg\_restore: creating TABLE "public.payment"  
pg\_restore: creating SEQUENCE "public.rental\_rental\_id\_seq"  
pg\_restore: creating TABLE "public.rental"  
pg\_restore: creating VIEW "public.sales\_by\_film\_category"  
pg\_restore: creating SEQUENCE "public.staff\_staff\_id\_seq"  
pg\_restore: creating TABLE "public.staff"  
pg\_restore: creating SEQUENCE "public.store\_store\_id\_seq"  
pg\_restore: creating TABLE "public.store"  
pg\_restore: creating VIEW "public.sales\_by\_store"  
pg\_restore: creating VIEW "public.staff\_list"  
pg\_restore: processing data for table "public.actor"  
pg\_restore: processing data for table "public.address"  
pg\_restore: processing data for table "public.category"  
pg\_restore: processing data for table "public.city"  
pg\_restore: processing data for table "public.country"  
pg\_restore: processing data for table "public.customer"  
pg\_restore: processing data for table "public.film"  
pg\_restore: processing data for table "public.film\_actor"  
pg\_restore: processing data for table "public.film\_category"  
pg\_restore: processing data for table "public.inventory"  
pg\_restore: processing data for table "public.language"  
pg\_restore: processing data for table "public.payment"  
pg\_restore: processing data for table "public.rental"  
pg\_restore: processing data for table "public.staff"  
pg\_restore: processing data for table "public.store"  
pg\_restore: executing SEQUENCE SET actor\_actor\_id\_seq  
pg\_restore: executing SEQUENCE SET address\_address\_id\_seq  
pg\_restore: executing SEQUENCE SET category\_category\_id\_seq  
pg\_restore: executing SEQUENCE SET city\_city\_id\_seq  
pg\_restore: executing SEQUENCE SET country\_country\_id\_seq  
pg\_restore: executing SEQUENCE SET customer\_customer\_id\_seq  
pg\_restore: executing SEQUENCE SET film\_film\_id\_seq  
pg\_restore: executing SEQUENCE SET inventory\_inventory\_id\_seq  
pg\_restore: executing SEQUENCE SET language\_language\_id\_seq  
pg\_restore: executing SEQUENCE SET payment\_payment\_id\_seq  
pg\_restore: executing SEQUENCE SET rental\_rental\_id\_seq  
pg\_restore: executing SEQUENCE SET staff\_staff\_id\_seq  
pg\_restore: executing SEQUENCE SET store\_store\_id\_seq  
pg\_restore: creating CONSTRAINT "public.actor actor\_pkey"  
pg\_restore: creating CONSTRAINT "public.address address\_pkey"  
pg\_restore: creating CONSTRAINT "public.category category\_pkey"  
pg\_restore: creating CONSTRAINT "public.city city\_pkey"  
pg\_restore: creating CONSTRAINT "public.country country\_pkey"  
pg\_restore: creating CONSTRAINT "public.customer customer\_pkey"  
pg\_restore: creating CONSTRAINT "public.film\_actor film\_actor\_pkey"  
pg\_restore: creating CONSTRAINT "public.film\_category film\_category\_pkey"  
pg\_restore: creating CONSTRAINT "public.film film\_pkey"  
pg\_restore: creating CONSTRAINT "public.inventory inventory\_pkey"  
pg\_restore: creating CONSTRAINT "public.language language\_pkey"  
pg\_restore: creating CONSTRAINT "public.payment payment\_pkey"  
pg\_restore: creating CONSTRAINT "public.rental rental\_pkey"  
pg\_restore: creating CONSTRAINT "public.staff staff\_pkey"  
pg\_restore: creating CONSTRAINT "public.store store\_pkey"  
pg\_restore: creating INDEX "public.film\_fulltext\_idx"  
pg\_restore: creating INDEX "public.idx\_actor\_last\_name"  
pg\_restore: creating INDEX "public.idx\_fk\_address\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_city\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_country\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_customer\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_film\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_inventory\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_language\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_rental\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_staff\_id"  
pg\_restore: creating INDEX "public.idx\_fk\_store\_id"  
pg\_restore: creating INDEX "public.idx\_last\_name"  
pg\_restore: creating INDEX "public.idx\_store\_id\_film\_id"  
pg\_restore: creating INDEX "public.idx\_title"  
pg\_restore: creating INDEX "public.idx\_unq\_manager\_staff\_id"  
pg\_restore: creating INDEX "public.idx\_unq\_rental\_rental\_date\_inventory\_id\_customer\_id"  
pg\_restore: creating TRIGGER "public.film film\_fulltext\_trigger"  
pg\_restore: creating TRIGGER "public.actor last\_updated"  
pg\_restore: creating TRIGGER "public.address last\_updated"  
pg\_restore: creating TRIGGER "public.category last\_updated"  
pg\_restore: creating TRIGGER "public.city last\_updated"  
pg\_restore: creating TRIGGER "public.country last\_updated"  
pg\_restore: creating TRIGGER "public.customer last\_updated"  
pg\_restore: creating TRIGGER "public.film last\_updated"  
pg\_restore: creating TRIGGER "public.film\_actor last\_updated"  
pg\_restore: creating TRIGGER "public.film\_category last\_updated"  
pg\_restore: creating TRIGGER "public.inventory last\_updated"  
pg\_restore: creating TRIGGER "public.language last\_updated"  
pg\_restore: creating TRIGGER "public.rental last\_updated"  
pg\_restore: creating TRIGGER "public.staff last\_updated"  
pg\_restore: creating TRIGGER "public.store last\_updated"  
pg\_restore: creating FK CONSTRAINT "public.customer customer\_address\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.film\_actor film\_actor\_actor\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.film\_actor film\_actor\_film\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.film\_category film\_category\_category\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.film\_category film\_category\_film\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.film film\_language\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.address fk\_address\_city"  
pg\_restore: creating FK CONSTRAINT "public.city fk\_city"  
pg\_restore: creating FK CONSTRAINT "public.inventory inventory\_film\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.payment payment\_customer\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.payment payment\_rental\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.payment payment\_staff\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.rental rental\_customer\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.rental rental\_inventory\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.rental rental\_staff\_id\_key"  
pg\_restore: creating FK CONSTRAINT "public.staff staff\_address\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.store store\_address\_id\_fkey"  
pg\_restore: creating FK CONSTRAINT "public.store store\_manager\_staff\_id\_fkey"  
[postgres@ip-172-31-20-155 ~]$

## **✅ Linux Host Access (e.g., EC2, Localhost)**

This tutorial assumes:

* You’re using a ****Linux-based server****
* You’re logged in as or have sudo access to the ****postgres**** system user
* You have access to the pg\_basebackup binary

Confirm it’s installed:

which pg\_basebackup

Output example:

[postgres@ip-172-31-20-155 ~]$ which pg\_basebackup  
/usr/bin/pg\_basebackup  
[postgres@ip-172-31-20-155 ~]$

## **✅ PostgreSQL Superuser or Replication Role**

The user running pg\_basebackup must:

* Have REPLICATION privilege in PostgreSQL
* Be able to connect to the local server via TCP/IP or socket

You can check user roles:

\du

And the role should include:

Role name | Attributes  
----------+-------------------------  
replicator | Replication, Login

Make sure the role can connect by setting the appropriate entries in pg\_hba.conf:

# Allow local replication connections  
local replication all trust

Reload PostgreSQL after changes:

sudo systemctl reload postgresql

With all prerequisites checked, you’re now ready to ****perform a full backup and restore**** using pg\_basebackup, which we'll cover in the next step.

💡 Tip: Use this setup not just for disaster recovery, but also to clone databases for development, testing, or analytics workloads — all without impacting the production instance.

## **🛠 Step-by-Step Backup & Restore Using**pg\_basebackup**in PostgreSQL 17**

When it comes to ensuring PostgreSQL is both ****resilient and recoverable****, pg\_basebackup is the default tool DBAs trust for full, binary-level backups. To use it effectively, especially for ****point-in-time recovery (PITR)**** or setting up a ****replica****, it's essential to configure ****WAL archiving**** properly.

Let’s walk through the first two steps to prepare PostgreSQL 17 for safe and restorable backups.

## **✅ Step 1: Verify Your PostgreSQL 17 Environment**

Before taking any backup, it’s good practice to confirm your database configuration and understand your WAL (Write-Ahead Logging) settings.

## **🔍 Connect to the PostgreSQL Instance**

First, switch to the PostgreSQL system user and open a psql session:

sudo su - postgres  
psql

Once you’re inside the psql shell, run the following diagnostics:

## **🔸 View the Current Data Directory**

SHOW data\_directory;

This displays the full path where PostgreSQL stores all its data files. For version 17, it typically looks like:

/var/lib/pgsql/17/data

This directory will later be your ****source**** for backup, and understanding its location is critical before using pg\_basebackup.

## **🔸 List Databases**

\l

This shows a list of databases present in the cluster. Make sure your production or sample databases (like dvdrental) are active and online.

## **🔸 Check Tablespaces**

\db+

Tablespaces can point to additional physical locations outside the default data\_directory. If you use tablespaces, pg\_basebackup will back them up too — but you must ensure those paths are accessible and writable.

## **🔸 Review Archive-Related Settings**

SELECT name, setting, unit  
FROM pg\_settings  
WHERE name LIKE 'archive%';

This outputs values like:

archive\_mode | off |  
archive\_command | (empty) |  
archive\_timeout | 0 |

If archive\_mode is off, and archive\_command is empty — ****your database isn’t ready for WAL archiving****, and backups won’t support PITR or replica creation.

Let’s fix that next.

## **🔧 Step 2: Enable WAL Archiving in PostgreSQL 17**

WAL (Write-Ahead Logging) is the heart of PostgreSQL’s durability and crash recovery. To back up the database ****in a way that allows replaying changes****, WAL segments must be archived externally using an archive command.

## **📝 Edit the PostgreSQL Configuration File**

Exit psql, and edit the postgresql.conf file located inside your PostgreSQL 17 data directory:

sudo vi /var/lib/pgsql/17/data/postgresql.conf

Locate or add the following lines:

archive\_mode = on  
archive\_timeout = 300  
archive\_command = 'test ! -f /var/lib/pgsql/17/data/pg\_wal/%f && cp %p /var/lib/pgsql/17/backups/pg\_wal/%f'

Let’s break it down:

* ****archive\_mode = on****  
  Activates the archiving process, instructing PostgreSQL to allow archive\_command execution.
* ****archive\_timeout = 300****  
  This forces PostgreSQL to switch and archive WAL files every 5 minutes, even if they're not full. It’s useful for ensuring WAL availability in low-traffic systems.
* ****archive\_command = '...'****  
  Defines the shell command used to copy completed WAL segments.
* %p = path to the WAL file to be archived
* %f = file name only  
  The test ! -f part prevents overwriting if the WAL file already exists in the target directory.

## **📁 Create the Archive Directory**

You must ensure the archive destination exists and has proper permissions:

sudo mkdir -p /var/lib/pgsql/17/backups/pg\_wal  
sudo chown postgres:postgres /var/lib/pgsql/17/backups/pg\_wal

[root@ip-172-31-92-215 ~]# sudo mkdir -p /var/lib/pgsql/17/backups/pg\_wal  
[root@ip-172-31-92-215 ~]#  
[root@ip-172-31-92-215 ~]# sudo chown postgres:postgres /var/lib/pgsql/17/backups/pg\_wal  
[root@ip-172-31-92-215 ~]#

Without this step, archive\_command will silently fail or raise permission errors in the logs.

## **🔄 Restart PostgreSQL 17**

Once the configuration is updated and the archive directory is ready, apply the changes:

sudo systemctl restart postgresql-17

Verify the status:

sudo systemctl status postgresql-17

And tail the PostgreSQL logs to confirm WAL archiving is working:

sudo journalctl -u postgresql-17 -f

You should see log lines such as:

archiver: archived WAL file "0000000100000000000000A1"

## **✅ At This Point**

Your PostgreSQL 17 setup is now ready for:

* Hot backups using pg\_basebackup
* PITR (Point-In-Time Recovery)
* Initializing standby replicas

## **🧰 Step 3: Take a Full Cluster Backup with**pg\_basebackup**in PostgreSQL 17**

Now that WAL archiving is enabled and PostgreSQL 17 is correctly configured, you’re ready to take a ****consistent, binary-level backup**** using pg\_basebackup.

Unlike logical backups (pg\_dump), pg\_basebackup gives you a complete snapshot of the ****entire PostgreSQL data directory****, making it ideal for disaster recovery, cloning, or setting up standby replicas.

## **🎯 What**pg\_basebackup**Does**

* Captures the full data directory, including user data, system catalogs, configuration files, and WAL files.
* Supports ****streaming WAL**** so your backup is consistent even while the server is running.
* Works with both ****compressed and uncompressed**** formats.

Let’s explore both options:

## **🔹 Option 1: Compressed Backup (Tar + Gzip)**

This option is useful when:

* You have limited disk space.
* You want to archive backups offsite (e.g., S3, GCS).
* You prefer smaller files for faster transfer.

## **🧱 Step 1: Create Backup Directory**

sudo mkdir -p /var/lib/pgsql/pg\_backup  
sudo chown postgres:postgres /var/lib/pgsql/pg\_backup

[root@ip-172-31-92-215 ~]# sudo mkdir -p /var/lib/pgsql/pg\_backup  
[root@ip-172-31-92-215 ~]#  
[root@ip-172-31-92-215 ~]# sudo chown postgres:postgres /var/lib/pgsql/pg\_backup  
[root@ip-172-31-92-215 ~]#

This ensures the directory is writable by the postgres user.

## **🚀 Step 2: Run the Compressed Backup Command**

pg\_basebackup \  
 -h localhost \  
 -p 5432 \  
 -U postgres \  
 -D /var/lib/pgsql/pg\_backup \  
 -Ft \  
 -z \  
 -Xs \  
 -P

[postgres@ip-172-31-92-215 ~]$ pg\_basebackup \  
 -h localhost \  
 -p 5432 \  
 -U postgres \  
 -D /var/lib/pgsql/pg\_backup \  
 -Ft \  
 -z \  
 -Xs \  
 -P  
Password:  
38520/38520 kB (100%), 1/1 tablespace  
[postgres@ip-172-31-92-215 ~]$  
[postgres@ip-172-31-92-215 ~]$

Let’s break down what each flag means:

Flag Description -h Host to connect to (localhost since we’re backing up the local DB) -p Port where PostgreSQL is running (default: 5432) -U PostgreSQL user with replication privileges (postgres) -D Destination directory for the backup files -Ft Format as tar archive -z Gzip compress the archive -Xs Stream the WAL files during the backup -P Show progress (interactive view of backup process)

⚠️ If you haven’t configured .pgpass or trust in pg\_hba.conf, this command may prompt for a password.

## **🔹 Option 2: Uncompressed Backup (Tar Format Only)**

This is ideal for:

* Local development.
* When speed matters more than space.
* Testing backup/restore pipelines.

## **🧱 Step 1: Create the Destination Directory**

sudo mkdir -p /var/lib/pgsql/pg\_backup\_uncompressed  
sudo chown postgres:postgres /var/lib/pgsql/pg\_backup\_uncompressed

[root@ip-172-31-92-215 ~]# sudo mkdir -p /var/lib/pgsql/pg\_backup\_uncompressed  
[root@ip-172-31-92-215 ~]#  
[root@ip-172-31-92-215 ~]# sudo chown postgres:postgres /var/lib/pgsql/pg\_backup\_uncompressed  
[root@ip-172-31-92-215 ~]#

## **🚀 Step 2: Run the Uncompressed Backup**

pg\_basebackup \  
 -h localhost \  
 -p 5432 \  
 -U postgres \  
 -D /var/lib/pgsql/pg\_backup\_uncompressed \  
 -Ft \  
 -Xs \  
 -P

[root@ip-172-31-92-215 ~]# pg\_basebackup \  
 -h localhost \  
 -p 5432 \  
 -U postgres \  
 -D /var/lib/pgsql/pg\_backup\_uncompressed \  
 -Ft \  
 -Xs \  
 -P  
Password:  
38523/38523 kB (100%), 1/1 tablespace  
[root@ip-172-31-92-215 ~]#

This command will create .tar files but will ****not compress them**** with gzip. This results in:

* Faster backup times.
* Larger file sizes.
* Easier extraction when restoring.

## **✅ Step 4: Verify Backup Completion**

Whether you created a compressed or uncompressed backup, it’s important to verify that all expected files were written.

## **📂 For Uncompressed Backup:**

ls -lrt /var/lib/pgsql/pg\_backup\_uncompressed

You should see files like:

[root@ip-172-31-92-215 ~]# ls -lrt /var/lib/pgsql/pg\_backup\_uncompressed  
total 55104  
-rw-------. 1 root root 39448064 Jun 30 19:52 base.tar  
-rw-------. 1 root root 194152 Jun 30 19:52 backup\_manifest  
-rw-------. 1 root root 16778752 Jun 30 19:52 pg\_wal.tar  
[root@ip-172-31-92-215 ~]#

These are:

* base.tar: Contains your cluster data (tables, indexes, catalogs).
* pg\_wal.tar: Contains the necessary WAL files to restore the database to a consistent state.

## **📦 For Compressed Backup:**

If you used the -z flag, your output may look like:

[root@ip-172-31-92-215 ~]# ls -ltr /var/lib/pgsql/pg\_backup  
total 5924  
-rw-------. 1 postgres postgres 194152 Jun 30 19:51 backup\_manifest  
-rw-------. 1 postgres postgres 5848048 Jun 30 19:51 base.tar.gz  
-rw-------. 1 postgres postgres 17685 Jun 30 19:51 pg\_wal.tar.gz  
[root@ip-172-31-92-215 ~]#

You can extract them later using:

tar -xzf base.tar.gz

or:

gzip -d base.tar.gz  
tar -xf base.tar

## **🛡 Tips for Ensuring a Healthy Backup**

* 🔁 Schedule this backup process regularly via cron.
* 🧪 Periodically test restoring your backups to avoid surprises during a real incident.
* 📦 If backing up to remote storage, validate file integrity using checksums.
* 📉 Monitor WAL size during the backup to avoid filling disk space (especially for large clusters).

## **💥 Step 5: Simulate a Failure (Disaster Recovery Test)**

Once you’ve completed a successful backup using pg\_basebackup, it’s essential to test the ****recovery process****.

Backups are **only useful if they can be restored** — so this step validates your strategy.

We’ll simulate a failure by shutting down PostgreSQL and ****deleting the entire data directory****, as if the server had been corrupted or suffered disk loss.

## **🔻 Stop PostgreSQL 17 Service**

To simulate failure, start by stopping the active PostgreSQL instance. This avoids file corruption during deletion.

sudo systemctl stop postgresql-17

[root@ip-172-31-92-215 ~]#  
[root@ip-172-31-92-215 ~]# sudo systemctl stop postgresql-17  
[root@ip-172-31-92-215 ~]#

You can verify the status using:

sudo systemctl status postgresql-17

[root@ip-172-31-92-215 ~]# sudo systemctl status postgresql-17  
○ postgresql-17.service - PostgreSQL 17 database server  
 Loaded: loaded (/usr/lib/systemd/system/postgresql-17.service; enabled; preset: disabled)  
 Active: inactive (dead) since Mon 2025-06-30 19:56:27 UTC; 2min 34s ago  
 Duration: 8min 37.580s  
 Invocation: ea4788d10d604eb382f91449d80e545d  
 Docs: https://www.postgresql.org/docs/17/static/  
 Process: 20056 ExecStartPre=/usr/pgsql-17/bin/postgresql-17-check-db-dir ${PGDATA} (code=exited, status=0/SUCCESS)  
 Process: 20062 ExecStart=/usr/pgsql-17/bin/postgres -D ${PGDATA} (code=exited, status=0/SUCCESS)  
 Main PID: 20062 (code=exited, status=0/SUCCESS)  
 Mem peak: 83M  
 CPU: 370ms  
  
Jun 30 19:47:46 ip-172-31-92-215.ec2.internal systemd[1]: Starting postgresql-17.service - PostgreSQL 17 database server...  
Jun 30 19:47:46 ip-172-31-92-215.ec2.internal postgres[20062]: 2025-06-30 19:47:46.622 UTC [20062] LOG: redirecting log output to logging collector process  
Jun 30 19:47:46 ip-172-31-92-215.ec2.internal postgres[20062]: 2025-06-30 19:47:46.622 UTC [20062] HINT: Future log output will appear in directory "log".  
Jun 30 19:47:46 ip-172-31-92-215.ec2.internal systemd[1]: Started postgresql-17.service - PostgreSQL 17 database server.  
Jun 30 19:56:24 ip-172-31-92-215.ec2.internal systemd[1]: Stopping postgresql-17.service - PostgreSQL 17 database server...  
Jun 30 19:56:27 ip-172-31-92-215.ec2.internal systemd[1]: postgresql-17.service: Deactivated successfully.  
Jun 30 19:56:27 ip-172-31-92-215.ec2.internal systemd[1]: Stopped postgresql-17.service - PostgreSQL 17 database server.  
Jun 30 19:56:27 ip-172-31-92-215.ec2.internal systemd[1]: postgresql-17.service: Consumed 370ms CPU time, 83M memory peak.  
[root@ip-172-31-92-215 ~]#

*The service should now be inactive (dead state).*

## **🧨 Delete the Existing Data Directory**

Now, delete PostgreSQL’s data\_directory. This mimics a catastrophic event like disk failure or accidental deletion.

⚠️ **Caution:** Only do this in a **test or staging environment** — not production!

sudo rm -rf /var/lib/pgsql/17/data

[root@ip-172-31-92-215 ~]#  
[root@ip-172-31-92-215 ~]# sudo rm -rf /var/lib/pgsql/17/data  
[root@ip-172-31-92-215 ~]#

💡 PostgreSQL stores all critical data in this directory: system catalogs, databases, config files, WAL segments, etc.

At this point, PostgreSQL has ****no idea how to start**** — it’s as if the cluster never existed. Next, let’s restore it.

## **🔄 Step 6: Restore the Backup (Base Files + WAL Files)**

With the original cluster erased, we now ****rebuild the PostgreSQL 17 environment**** using the files generated by pg\_basebackup.

You’ll restore:

* base.tar.gz: Full physical snapshot of your database
* pg\_wal.tar.gz: Write-Ahead Logs required to bring the cluster to a consistent state

## **📁 Step 6.1: Restore the Base Backup**

Let’s restore the snapshot that holds all database data.

### **📌 Step 1: Go to the Backup Location**

Navigate to the location where you saved your backup:

cd /var/lib/pgsql/pg\_backup

[root@ip-172-31-92-215 ~]# cd /var/lib/pgsql/pg\_backup  
[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]#

Your backup directory should contain base.tar.gz and pg\_wal.tar.gz.

### **📌 Step 2: Recreate the Data Directory**

Now we recreate the PostgreSQL 17 data directory from scratch:

sudo mkdir -p /var/lib/pgsql/17/data  
sudo chown postgres:postgres /var/lib/pgsql/17/data

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# sudo mkdir -p /var/lib/pgsql/17/data  
[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# sudo chown postgres:postgres /var/lib/pgsql/17/data  
[root@ip-172-31-92-215 pg\_backup]#

This prepares the environment for extraction.

### **📌 Step 3: Extract the Base Backup**

If your backup is compressed (base.tar.gz), first unzip it:

gunzip base.tar.gz

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# gunzip base.tar.gz  
[root@ip-172-31-92-215 pg\_backup]#

Now extract its contents into the new data directory:

tar -xvf base.tar -C /var/lib/pgsql/17/data

[root@ip-172-31-92-215 pg\_backup]# tar -xvf base.tar -C /var/lib/pgsql/17/data  
backup\_label  
tablespace\_map  
pg\_wal/  
./pg\_wal/archive\_status/  
./pg\_wal/summaries/  
global/  
global/1262  
global/2964  
global/1213  
global/1260  
global/1261  
global/1214  
global/2396  
global/6000  
global/3592  
global/6243  
global/6100  
global/4177  
global/4178  
global/2966  
global/2967  
global/4185  
global/4186  
global/4175  
global/4176  
global/2846  
global/2847  
global/4181  
global/4182  
global/4060  
global/4061  
global/6244  
global/6245  
global/4183  
global/4184  
global/2671  
global/2672  
global/2965

This command restores the ****entire PostgreSQL system****, including:

* base/ → database files
* global/ → system-wide metadata
* pg\_xact/, pg\_multixact/, pg\_stat/ → transaction status
* Config files like postgresql.auto.conf, standby.signal, etc.

At this point, your database structure is rebuilt.

## **🔁 Step 6.2: Restore WAL Segment Files**

WAL files are essential to make the database consistent — they allow recovery up to the moment the backup finished.

### **📌 Step 1: Decompress the WAL Archive**

gunzip pg\_wal.tar.gz

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# gunzip pg\_wal.tar.gz  
[root@ip-172-31-92-215 pg\_backup]#

This creates pg\_wal.tar.

### **📌 Step 2: Create the**pg\_wal**Directory**

If it doesn’t already exist, create the WAL segment directory inside the restored cluster:

mkdir -p /var/lib/pgsql/17/data/pg\_wal

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# mkdir -p /var/lib/pgsql/17/data/pg\_wal  
[root@ip-172-31-92-215 pg\_backup]#

You may need to remove a partially extracted pg\_wal/ folder before restoring, to avoid conflicts.

### **📌 Step 3: Extract the WAL Files**

tar -xvf pg\_wal.tar -C /var/lib/pgsql/17/data/pg\_wal

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# tar -xvf pg\_wal.tar -C /var/lib/pgsql/17/data/pg\_wal  
000000010000000000000002  
archive\_status/000000010000000000000002.done  
[root@ip-172-31-92-215 pg\_backup]#

This places all archived WAL files into the correct location, enabling PostgreSQL to ****replay transactions and finalize recovery**** when started.

## **🔍 Step 6.3: Verify the Restored Structure**

Before proceeding, quickly check the directory structure:

ls -lh /var/lib/pgsql/17/data  
ls -lh /var/lib/pgsql/17/data/pg\_wal

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# ls -lh /var/lib/pgsql/17/data  
total 64K  
-rw-------. 1 postgres postgres 3 Jun 30 19:36 PG\_VERSION  
-rw-------. 1 postgres postgres 225 Jun 30 19:51 backup\_label  
drwx------. 6 postgres postgres 46 Jun 30 19:51 base  
-rw-------. 1 postgres postgres 30 Jun 30 19:47 current\_logfiles  
drwx------. 2 postgres postgres 4.0K Jun 30 20:03 global  
drwx------. 2 postgres postgres 32 Jun 30 19:36 log  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_commit\_ts  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_dynshmem  
-rw-------. 1 postgres postgres 5.4K Jun 30 19:36 pg\_hba.conf  
-rw-------. 1 postgres postgres 2.6K Jun 30 19:36 pg\_ident.conf  
drwx------. 4 postgres postgres 68 Jun 30 19:51 pg\_logical  
drwx------. 4 postgres postgres 36 Jun 30 19:36 pg\_multixact  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_notify  
drwx------. 2 postgres postgres 6 Jun 30 19:51 pg\_replslot  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_serial  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_snapshots  
drwx------. 2 postgres postgres 6 Jun 30 19:47 pg\_stat  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_stat\_tmp  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_subtrans  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_tblspc  
drwx------. 2 postgres postgres 6 Jun 30 19:36 pg\_twophase  
drwx------. 4 postgres postgres 77 Jun 30 20:06 pg\_wal  
drwx------. 2 postgres postgres 18 Jun 30 19:36 pg\_xact  
-rw-------. 1 postgres postgres 88 Jun 30 19:36 postgresql.auto.conf  
-rw-------. 1 postgres postgres 31K Jun 30 19:47 postgresql.conf  
-rw-------. 1 postgres postgres 0 Jun 30 19:51 tablespace\_map  
[root@ip-172-31-92-215 pg\_backup]#

[root@ip-172-31-92-215 pg\_backup]# ls -lh /var/lib/pgsql/17/data/pg\_wal  
total 16M  
-rw-------. 1 postgres postgres 16M Jun 30 19:51 000000010000000000000002  
drwx------. 2 postgres postgres 43 Jun 30 20:06 archive\_status  
drwx------. 2 postgres postgres 6 Jun 30 19:47 summaries  
[root@ip-172-31-92-215 pg\_backup]#

You should see PostgreSQL folders like:

* base/, pg\_wal/, pg\_xact/, global/, pg\_stat/
* Files like PG\_VERSION, postgresql.auto.conf, pg\_control

This confirms that your cluster has been physically restored.

## **🔧 Step 7: Update PostgreSQL 17 Restore Configuration**

After restoring the base data and WAL (Write-Ahead Log) files, we must ensure that PostgreSQL knows ****how to locate and restore the archived WAL files**** during startup. This is done via the restore\_command setting in the postgresql.conf file.

## **🔍 What is**restore\_command**?**

The restore\_command tells PostgreSQL how to ****retrieve archived WAL segments**** when starting up after a crash or when performing recovery. It is essential for:

* Crash recovery
* Replication
* Point-in-Time Recovery (PITR)

During recovery, PostgreSQL will:

1. Detect it needs to replay WAL.
2. Request each missing WAL segment by name.
3. Use restore\_command to fetch the segment.

If the file is not found, recovery will ****pause or fail****, which is why this step is critical.

## **🛠️ Editing the Configuration File**

Open the postgresql.conf file in your restored cluster’s data directory:

sudo vi /var/lib/pgsql/17/data/postgresql.conf

Scroll down to the ****archiving and recovery**** section and set the following:

restore\_command = 'cp /var/lib/pgsql/17/backups/pg\_wal/%f /var/lib/pgsql/17/data/pg\_wal/%p'

## **Explanation of this line:**

* %f: The filename of the requested WAL segment (e.g., 00000001000000010000005A)
* %p: The absolute path where PostgreSQL expects the WAL segment
* cp: The command that physically copies the file from the archive to the required location

✅ This command ensures PostgreSQL can retrieve and apply archived WALs during recovery.

## **🔐 Ensure Permissions Are Correct**

Make sure the postgres user has ownership and permissions over the pg\_wal directory and the restored files:

sudo chown -R postgres:postgres /var/lib/pgsql/17/data  
sudo chown -R postgres:postgres /var/lib/pgsql/17/backups

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# sudo chown -R postgres:postgres /var/lib/pgsql/17/data  
[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# sudo chown -R postgres:postgres /var/lib/pgsql/17/backups  
[root@ip-172-31-92-215 pg\_backup]#

## **🚀 Step 8: Restart PostgreSQL 17 and Verify Restoration**

With everything in place — restored files and restore\_command configured — it’s time to bring the PostgreSQL service back online and confirm that everything works.

## **🔄 Restart PostgreSQL 17**

Start the PostgreSQL service:

sudo systemctl start postgresql-17

If you are getting an error like the one below, then you need to troubleshoot it as follows

I have just reproduced the error for your reference

[root@ip-172-31-92-215 pg\_backup]# sudo systemctl start postgresql-17  
Job for postgresql-17.service failed because the control process exited with error code.  
See "systemctl status postgresql-17.service" and "journalctl -xeu postgresql-17.service" for details.  
[root@ip-172-31-92-215 pg\_backup]#

You need to run the following command to check the error details

journalctl -xeu postgresql-17.service

You will see error details like the following:

*****data directory “/var/lib/pgsql/17/data” has invalid permissions*****

*****ETAIL: Permissions should be u=rwx (0700) or u=rwx,g=rx (0750).*****

Run the following journalctl command:

[root@ip-172-31-92-215 pg\_backup]# journalctl -xeu postgresql-17.service  
░░  
░░ A stop job for unit postgresql-17.service has finished.  
░░  
░░ The job identifier is 83856 and the job result is done.  
Jun 30 19:56:27 ip-172-31-92-215.ec2.internal systemd[1]: postgresql-17.service: Consumed 370ms CPU time, 83M memory peak.  
░░ Subject: Resources consumed by unit runtime  
░░ Defined-By: systemd  
░░ Support: https://access.redhat.com/support  
░░  
░░ The unit postgresql-17.service completed and consumed the indicated resources.  
Jun 30 20:09:34 ip-172-31-92-215.ec2.internal systemd[1]: Starting postgresql-17.service - PostgreSQL 17 database server...  
░░ Subject: A start job for unit postgresql-17.service has begun execution  
░░ Defined-By: systemd  
░░ Support: https://access.redhat.com/support  
░░  
░░ A start job for unit postgresql-17.service has begun execution.  
░░  
░░ The job identifier is 84142.  
Jun 30 20:09:34 ip-172-31-92-215.ec2.internal postgres[20483]: 2025-06-30 20:09:34.419 UTC [20483] FATAL: data directory "/var/lib/pgsql/17/data" has invalid permissions  
Jun 30 20:09:34 ip-172-31-92-215.ec2.internal postgres[20483]: 2025-06-30 20:09:34.419 UTC [20483] DETAIL: Permissions should be u=rwx (0700) or u=rwx,g=rx (0750).  
Jun 30 20:09:34 ip-172-31-92-215.ec2.internal systemd[1]: postgresql-17.service: Main process exited, code=exited, status=1/FAILURE  
░░ Subject: Unit process exited  
░░ Defined-By: systemd  
░░ Support: https://access.redhat.com/support  
░░  
░░ An ExecStart= process belonging to unit postgresql-17.service has exited.  
░░  
░░ The process' exit code is 'exited' and its exit status is 1.  
Jun 30 20:09:34 ip-172-31-92-215.ec2.internal systemd[1]: postgresql-17.service: Failed with result 'exit-code'.  
░░ Subject: Unit failed  
░░ Defined-By: systemd  
░░ Support: https://access.redhat.com/support  
░░  
░░ The unit postgresql-17.service has entered the 'failed' state with result 'exit-code'.  
Jun 30 20:09:34 ip-172-31-92-215.ec2.internal systemd[1]: Failed to start postgresql-17.service - PostgreSQL 17 database server.  
░░ Subject: A start job for unit postgresql-17.service has failed  
░░ Defined-By: systemd  
░░ Support: https://access.redhat.com/support  
░░  
░░ A start job for unit postgresql-17.service has finished with a failure.  
░░  
░░ The job identifier is 84142 and the job result is failed.  
[root@ip-172-31-92-215 pg\_backup]#

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# sudo chmod 750 /var/lib/pgsql/17/data  
[root@ip-172-31-92-215 pg\_backup]#

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# sudo systemctl start postgresql-17  
[root@ip-172-31-92-215 pg\_backup]#

Check the status immediately:

sudo systemctl status postgresql-17

[root@ip-172-31-92-215 pg\_backup]#  
[root@ip-172-31-92-215 pg\_backup]# sudo systemctl status postgresql-17  
● postgresql-17.service - PostgreSQL 17 database server  
 Loaded: loaded (/usr/lib/systemd/system/postgresql-17.service; enabled; preset: disabled)  
 Active: active (running) since Mon 2025-06-30 20:16:34 UTC; 27s ago  
 Invocation: c4cd0309e4c84fe8933f472fbb3a3510  
 Docs: https://www.postgresql.org/docs/17/static/  
 Process: 20523 ExecStartPre=/usr/pgsql-17/bin/postgresql-17-check-db-dir ${PGDATA} (code=exited, status=0/SUCCESS)  
 Main PID: 20529 (postgres)  
 Tasks: 8 (limit: 5687)  
 Memory: 34.2M (peak: 34.4M)  
 CPU: 65ms  
 CGroup: /system.slice/postgresql-17.service  
 ├─20529 /usr/pgsql-17/bin/postgres -D /var/lib/pgsql/17/data/  
 ├─20531 "postgres: logger "  
 ├─20532 "postgres: checkpointer "  
 ├─20533 "postgres: background writer "  
 ├─20535 "postgres: walwriter "  
 ├─20536 "postgres: autovacuum launcher "  
 ├─20537 "postgres: archiver "  
 └─20538 "postgres: logical replication launcher "  
  
Jun 30 20:16:34 ip-172-31-92-215.ec2.internal systemd[1]: Starting postgresql-17.service - PostgreSQL 17 database server...  
Jun 30 20:16:34 ip-172-31-92-215.ec2.internal postgres[20529]: 2025-06-30 20:16:34.544 UTC [20529] LOG: redirecting log output to logging collector process  
Jun 30 20:16:34 ip-172-31-92-215.ec2.internal postgres[20529]: 2025-06-30 20:16:34.544 UTC [20529] HINT: Future log output will appear in directory "log".  
Jun 30 20:16:34 ip-172-31-92-215.ec2.internal systemd[1]: Started postgresql-17.service - PostgreSQL 17 database server.  
[root@ip-172-31-92-215 pg\_backup]#

## **Expected output:**

* Status should be active (running)
* No critical errors or failure messages

💡 If anything goes wrong (e.g., WAL not found, permissions denied), check logs with:

sudo journalctl -u postgresql-17 -n 50

## **🧠 What Happens Behind the Scenes?**

Once PostgreSQL starts:

1. It detects that a recovery is needed (based on the presence of backup\_label or standby.signal).
2. It begins ****replaying the WAL logs**** from the pg\_wal directory.
3. If additional segments are needed, it uses restore\_command to fetch them.
4. After reaching consistency, it transitions to the normal operational state and accepts connections.

## **🔑 Connect to PostgreSQL and Validate the Restore**

Switch to the postgres system user:

sudo su - postgres

[root@ip-172-31-92-215 pg\_backup]# sudo su - postgres  
Last login: Mon Jun 30 19:51:00 UTC 2025 on pts/2  
[postgres@ip-172-31-92-215 ~]$  
[postgres@ip-172-31-92-215 ~]$

Then connect to PostgreSQL:

psql

[postgres@ip-172-31-92-215 ~]$  
[postgres@ip-172-31-92-215 ~]$ psql  
psql (17.5)  
Type "help" for help.  
  
postgres=#

Inside the PostgreSQL shell, check the list of databases:

\l+

You should see:

postgres=#  
postgres=# \l+  
 List of databases  
 Name | Owner | Encoding | Locale Provider | Collate | Ctype | Locale | ICU Rules | Access privileges | Size | Tablespace | Description  
-----------+----------+----------+-----------------+---------+---------+--------+-----------+-----------------------+---------+------------+--------------------------------------------  
 dvdrental | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | | 15 MB | pg\_default |  
 postgres | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | | 7515 kB | pg\_default | default administrative connection database  
 template0 | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | =c/postgres +| 7361 kB | pg\_default | unmodifiable empty database  
 | | | | | | | | postgres=CTc/postgres | | |  
 template1 | postgres | UTF8 | libc | C.UTF-8 | C.UTF-8 | | | =c/postgres +| 7433 kB | pg\_default | default template for new databases  
 | | | | | | | | postgres=CTc/postgres | | |  
(4 rows)  
  
postgres=#

✅ These are the exact databases that existed at the time of your backup.

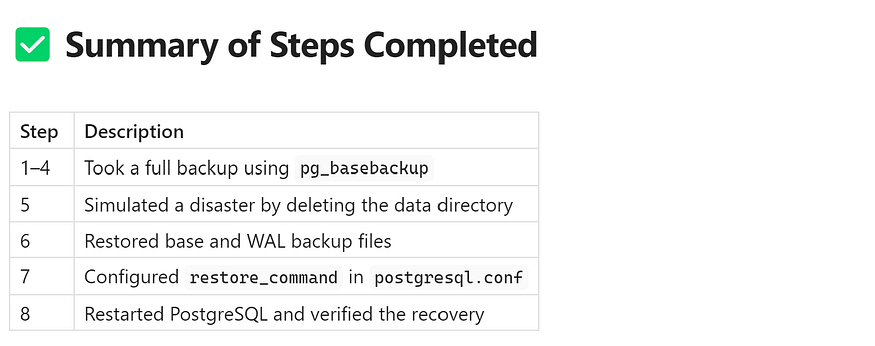
You can explore tables and data:

\c dvdrental  
\dt  
SELECT COUNT(\*) FROM some\_table;

postgres=#  
postgres=# \c dvdrental  
You are now connected to database "dvdrental" as user "postgres".  
dvdrental=#  
dvdrental=# \dt  
 List of relations  
 Schema | Name | Type | Owner  
--------+---------------+-------+----------  
 public | actor | table | postgres  
 public | address | table | postgres  
 public | category | table | postgres  
 public | city | table | postgres  
 public | country | table | postgres  
 public | customer | table | postgres  
 public | film | table | postgres  
 public | film\_actor | table | postgres  
 public | film\_category | table | postgres  
 public | inventory | table | postgres  
 public | language | table | postgres  
 public | payment | table | postgres  
 public | rental | table | postgres  
 public | staff | table | postgres  
 public | store | table | postgres  
(15 rows)  
  
dvdrental=#  
dvdrental=# SELECT COUNT(\*) FROM actor;  
 count  
-------  
 200  
(1 row)  
  
dvdrental=#

This confirms that ****your data, structure, and transactions were fully restored****.

Press enter or click to view image in full size



🎉 ****Congratulations!**** You’ve just walked through a ****realistic PostgreSQL 17 disaster recovery scenario**** using native tools and no third-party plugins.

This setup prepares you for:

* Hardware crashes
* Accidental data deletion
* Cloud VM failures
* Compliance audits (via PITR)

Would you like to continue with:

* ⏱️ Automating this backup/restore process via cron or shell scripts?
* ☁️ Sending WAL backups to AWS S3 or GCS for offsite recovery?
* ⏪ Performing Point-in-Time Recovery (PITR) to a specific timestamp?

## **⚠️ Important Considerations When Using pg\_basebackup in PostgreSQL 17**

Creating a backup is easy. Creating a ****useful**** and ****reliable**** backup that you can confidently restore in a crisis is what truly matters. Let’s take a closer look at some critical considerations to make your PostgreSQL backup strategy production-grade.

## **🗜 Compression Saves Disk Space — But May Increase CPU Load**

When taking a backup using pg\_basebackup, you can enable compression by adding the -z flag (e.g., pg\_basebackup -z ...). This compresses the backup archive using gzip, reducing file size significantly.

### **Benefits:**

* ****Reduces disk space usage****, especially for large databases
* ****Minimizes transfer size**** if you’re uploading the backup to cloud storage
* Speeds up backups over the network

### **Tradeoffs:**

* ****Compression consumes CPU****. On busy or resource-constrained PostgreSQL servers, it may temporarily impact database performance during the backup window.
* Compression may ****slow down backup speed**** slightly depending on the server’s I/O and CPU.

🧠 **Best Practice**: Use compression on staging environments or schedule compressed backups during off-peak hours. Alternatively, perform the backup from a replica server to avoid impacting the primary database.

## **🧾 WAL Logs Are Not Optional — They Are Critical**

One of the biggest mistakes novice DBAs make is backing up only the base data directory without including WAL (Write-Ahead Log) segments.

In PostgreSQL, WAL files are essential because:

* They contain ****all changes**** made to the database.
* PostgreSQL uses them to ****replay committed transactions**** that occurred after the base backup was taken.
* WALs are required for ****Point-in-Time Recovery (PITR)**** and ****replication****.

If you lose the WAL logs, you may still have the base backup — but you’ll lose any data created, updated, or deleted after that snapshot.

### **What to Do:**

* Set archive\_mode = on in postgresql.conf
* Configure a reliable archive\_command, such as:

archive\_command = 'cp %p /var/lib/pgsql/17/backups/pg\_wal/%f'

* Store WAL files in durable storage (NFS, S3, GCS, etc.)
* Periodically clean up old WALs with tools like pg\_archivecleanup to avoid disk overflows

🔐 Without WAL backups, **you cannot achieve PITR**, and you risk **partial or corrupted restores**.

## **🔄 Test Your Backups Regularly**

Taking a backup is only half the job — you need to know with certainty that your backups actually work.

### **Risks of Not Testing:**

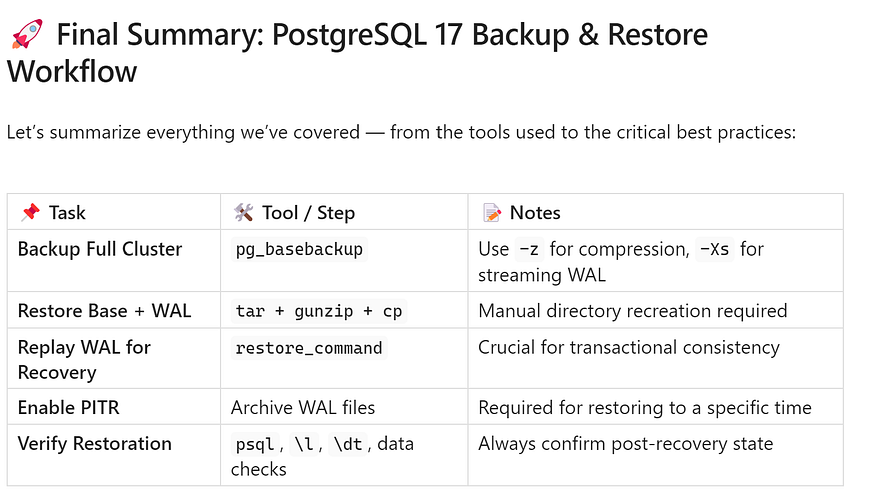
* Backups that ****look complete but are corrupted****
* Archived WAL files ****missing or not properly copied****
* restore\_command not working, breaking recovery
* Inconsistent configurations that silently fail

### **How to Test:**

* Set up a separate PostgreSQL 17 test server or container
* Simulate real disaster scenarios (delete the data directory)
* Restore the base backup and replay WALs
* Validate the data integrity (e.g., row counts, indexes, schema)

🧪 A backup you haven’t tested is just a **false sense of security**.

Press enter or click to view image in full size



## **✅ You’re Now Ready for Real-World PostgreSQL Recovery Scenarios**

By following this workflow, you’ve:

* Used PostgreSQL-native tools (no third-party dependencies)
* Taken a ****hot backup**** without downtime
* Recovered from a ****simulated disaster**** on the same host
* Ensured support for ****WAL-based recovery and PITR****

Whether you’re running PostgreSQL 17 on-prem, on EC2, or inside a Kubernetes cluster, this strategy scales.