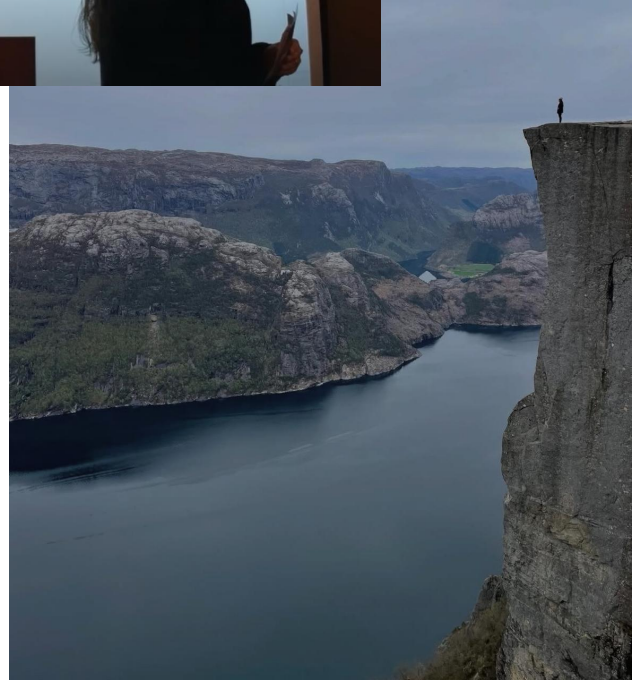
CTRL+Z for your Database

Mastering Backup and Recovery in PostgreSQL

SAKSHI NASHA
Senior Software Engineer
@ Cohesity

$whoami

- Developer👩‍💻
- OpenSearch Ambassador
- AWS community Builder ☁️
- Innovator : Hackathons
- Athlete at heart : 🏃‍♀️🏀⚽🏸




1/3
GDG Pune


praise me

Why Backup matters?

Why Backup matters?

💥 Data Loss Happens

- **Hardware failure:** disks die, RAID fails
- **Human error:** accidental deletions, wrong commands (`rm -rf /`)
- **Cloud outage:** even “highly available” systems can go down -> AWS us-east-1 region outage

🛡️ Backup = Insurance, Not Luxury

📊 Key Metrics

- **RPO (Recovery Point Objective):**
  → *How much data can we afford to lose?* (e.g., last 15 mins of transactions)

- **RTO (Recovery Time Objective):**
  → *How long can we afford to be down?* (e.g., 1 hour max downtime)


WHEN YOU REALIZE
YOUR LAST BACKUP W
6 MONTHS AGO
CHEERS... TO THE
GHOST OF MY DATA


If something can go wrong, it will.
Murphy's Law
quotefancy
COLOR SLIDES

“When was the last time you tested restoring your backup?”

“How confident are you that your backups actually restore?

100% confident ,
50% confident

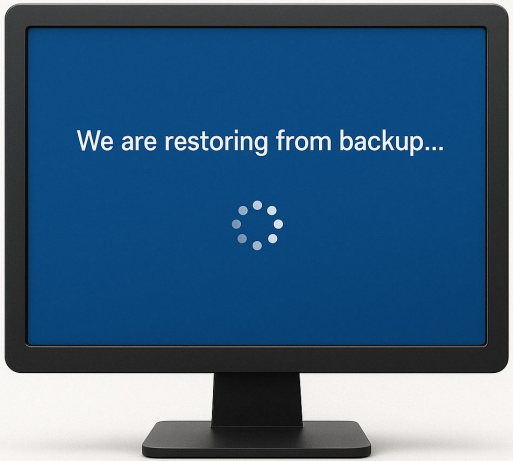
We’ll find out when disaster strikes 🔥

“A single `DROP DATABASE` can cost millions.”

GitLab famously lost production data in 2017 due to a failed `rm -rf` and incomplete backups.

A developer accidentally deleted the production database while attempting to troubleshoot a high database load caused by spam and a background job.

GitLab Incident, 2017.

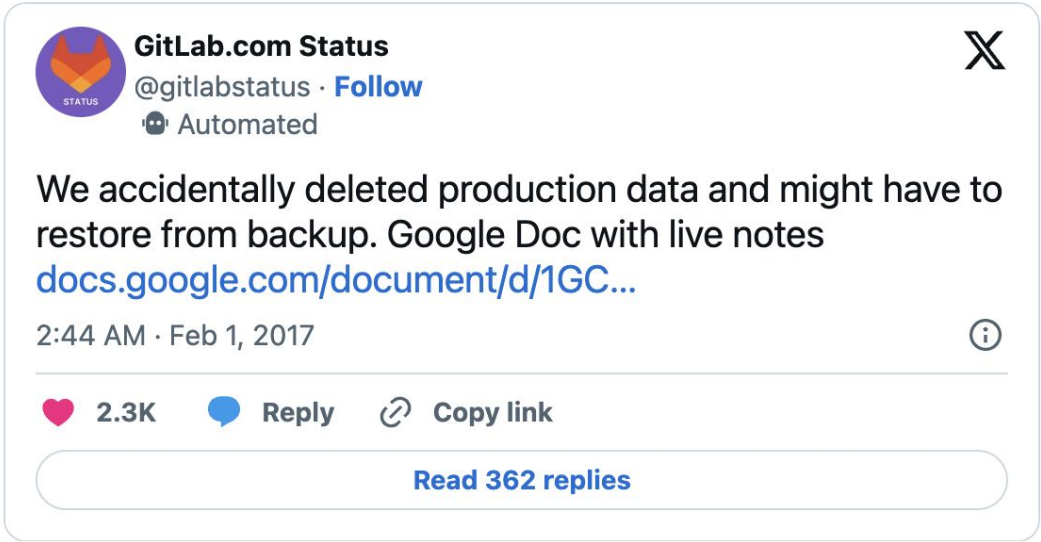


Famous last words.

The outage resulted in data loss and took **18 hours to recover (RTO)**

Recovery

We're working on recovering right now by using a backup of the database from a staging database.



When do you usually test your backups?”

🅐 After someone deletes production data
🅑 During our annual ‘panic testing day’
🅒 Every week like responsible adults
🅓 Testing? What’s that?

Few challenges these days?

1. 💣 **“We Have Backups”… But Never Test Them**
    a. A backup is only as good as its *restore* test.
    b. **Our backup isn’t a backup until you’ve restored it successfully.**
2. Rising Data Volumes & Complex Architectures
    a. Databases are **larger and distributed** (microservices, sharding, multi-region).
    b. Full backups take too long
3. Human Error & Process Gaps
    a. Accidental DROP TABLE, wrong schema migration, or cron misconfigurations.
    b. Missing role-based access controls or versioned backup policies.
4. Cloud snapshots ≠ full backups.
    a. Provider outages (AWS, GCP) or account misconfigurations can cause loss.
    b. Need **cross-region**, **cross-provider** copies and verified restore paths.
5. Security & Compliance
    a. Backups often stored unencrypted or accessible by too many users.
    b. Regulations (GDPR, HIPAA) demand secure, auditable, *restorable* backups.
    c. Need **encryption-at-rest**, **key rotation**, and **immutable backup storage**.

Let's get to know your 🔙 UP! Strategy ⁉️

Logical vs Physical : The Two Backup Universes

- A logical backup saves the data and schema in a human-readable form, such as SQL commands
- It extracts the data using SQL queries.
- # Logical backup
  pg_dump *mydb > mydb.sql*
- # Restore
  psql -d mydb -f mydb.sql

Strengths :

- **Portable**: You can restore it on another version or server.
- **Readable**: It's plain text SQL

Weaknesses :

- **Slower** for large databases
- Doesn't include transaction logs (no **PITR**)
- Needs the database to be **consistent** (no partial backups during changes)

- A physical backup copies the actual binary files that PostgreSQL uses to store data
- It's like copying the entire data directory
- # Stop the server and copy data directory
  sudo systemctl stop postgresql
  cp -r /var/lib/postgresql/15/main /backup/
  sudo systemctl start postgresql
- # use pg_basebackup for online backup
  pg_basebackup *-D /backup -F tar -z -P -U replica*

Strengths :

- **Faster** for large databases.
- **Exact copy** of the database
- Supports PITR

Weaknesses :

- **Not portable:** Must be restored on the same version & OS.
- **Large files:** Includes everything
- **More complex restore** process (especially for PITR).

How Logical and Physical Backups Are Used Together 🤝

Imagine you're a **database admin** for an **online shopping website** (say, "ShopZone").
You need to make sure no data is lost even if a developer deletes a table or the server crashes.

Logical Backup (Export schema + data)

- Once a **week** (say Sunday night), you create a **logical dump**
- This backup contains **SQL statements** that recreate tables and data.

✅ **Purpose:**
If you ever:

- Upgrade PostgreSQL to a newer version,
- Move to a new server,
- Or just need to recover a specific table you can easily restore

⚙️ **WEEKLY**

Physical Backup (Full system snapshot)

- You set up a **nightly physical backup** at 2 AM when traffic is low.
- You have a **complete copy of the database cluster** with **WAL logs**

✅ **Purpose:**
If the database crashes or data gets corrupted, you can restore the exact state **at any moment** before the crash (PITR — Point-In-Time Recovery).

🧩 **DAILY**

Going Deeper : Full vs Incremental

“Don’t start from zero every night”

- Full = complete copy
- pg_basebackup, pg_dump (logical backup per database)
- Incremental = store only changed data since the last full or incremental backup.
- Trade-offs in speed vs complexity
- PostgreSQL 17 (September 26 2024) : pg_basebackup --incremental
- New helper tool: pg_combinebackup (to rebuild a full backup from incremental layers)

Use Case Example:

Take a **weekly full backup**, and **daily incremental backups** — combine them with WAL archiving for full PITR capability.

Write-Ahead Logging (WAL)

“The Database’s Black Box Recorder”

- Every change written to WAL before data files
- Enables crash recovery and Point-in-Time Recovery (PITR)
- At all times, PostgreSQL maintains a write ahead log (WAL) in the pg_wal/ subdirectory of the cluster's data directory.

DROP TABLE users;

Database: I don’t know that shortcut

Interesting Fact!!

- **“In PostgreSQL, your database is writing a mini backup *every second*.”**

  The WAL (Write Ahead Log) is a continuous time machine of your data , most people don’t realize PostgreSQL is already your partial CTRL+Z.

🔷 Quick Check

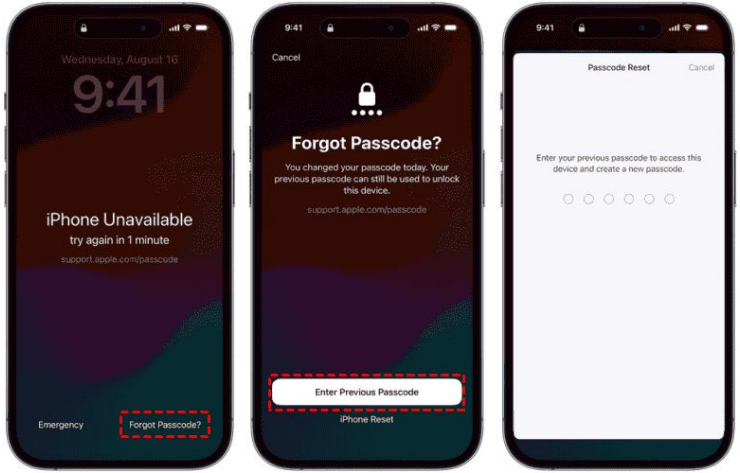
What’s really the difference between Recovery and Restore ?

What’s really the difference between Recovery and Restore ?

- **Recover = get back to working**
- **Restore = bring back what was lost or how it used to be**

- **Recovery:**
  You forgot your password or used incorrect pwd more than 3 times You go into **Recovery Mode** to fix it so you can unlock the phone
  👉 You’re **recovering** the phone : making it functional.

- **Restore:**
  After fixing the phone, you use a backup to **restore** your photos, contacts, and settings.
  👉 You’re **restoring** the old data and setup.



Continuous Archiving & PITR

“Time Travel for Databases”

- Combines a base backup with continuous archiving of WAL (Write-Ahead Log) files.
- Allows restoring the database to any specific point in time.
- Enables crash recovery and Point-in-Time Recovery (PITR)
- Steps to configure
  a. Enable WAL archiving in postgresql.conf:
     wal_level = replica
     archive_mode = on
     archive_command = 'cp %p /var/lib/postgresql/wal_archive/%f'.
  b. Take a base backup:
     pg_basebackup -D /backup -F tar -z …
  c. Restore to a point in time:
     restore_command = 'cp /var/lib/postgresql/wal_archive/%f %p'
     recovery_target_time = '2025-10-23 12:00:00

Continuous Archiving & PITR

“Time Travel for Databases”

Note

pg_dump and pg_dumpall do not produce file-system-level backups and cannot be used as part of a continuous-archiving solution. Such dumps are *logical* and do not contain enough information to be used by WAL replay.

Key Caveats for Continuous Archiving

🧩 1. Template Database Cross-Contamination

- PostgreSQL physically copies data files from the **template database (template1)** when creating a new DB.
- If `CREATE DATABASE` runs **during a base backup** and `template1` is modified mid-way:
  → those changes can leak into the new DB after recovery.
  Result: inconsistent or duplicated data in databases created during backup.

- **Best Practice:**
  🔒 Don't modify `template1` (or any template DB) while a base backup is in progress.
  🧁 *Analogy:* "Never reshape the cookie mold while copying the dough."

🗄️ 2. Tablespace Path Replay Issue

- `CREATE TABLESPACE` is **WAL-logged with an absolute path** (e.g., `/mnt/fastdisk/pg_tblspc`).
- During recovery, WAL replay will **try to recreate that exact path** — even on a different server or data directory.
- **Risks:**
  - Path may not exist → recovery failure.
  - Path exists but points to an old location → data overwrite.

- **Best Practice:**
  🧠 Always take a **new base backup** after creating or dropping tablespaces.
  📦 *Analogy:* "WAL is like a moving truck with a fixed delivery address — if you move but don't update the address, it delivers boxes to the old house."

What’s New in PostgreSQL 17 (September 26 2024)

“Smarter Backups, Easier Recovery”

- Incremental Base Backups with ***pg_basebackup --incremental*** → only changed files since last backup
- **pg_combinebackup** → merge full + incremental into a new baseline
- **pg_dump --filter** → selective backup by table/schema
- Performance: faster restore checksums, parallelism improvements
- Security: client-side encryption for base backups

  👉 Makes PITR and cloud storage backups far more efficient.


BACKUP ADMINS WAITING FOR A RESTORE
imgflip.com

What’s New in PostgreSQL 18 (September 25 2025)

“Resilience & Observability Boosts”

- **pg_basebackup --resume** — continue failed backups
- Improved monitoring (pg_stat_io, WAL stats)
- Checkpointer performance: faster crash recovery
- Data-corruption detection improvements
- Enhanced logical replication: large transactions, DDL support

Backups aren’t just about having copies - they’re about having *recoverable*, *resilient*, and *redundant* copies.

Real-World Backup Strategy

"Layered Protection: The 3-2-1 Rule"

3

1. Production database
2. On-site backup (pg_basebackup + WAL)
3. Off-site/cloud replica or archive.

2

Store copies on **two different kinds of storage** to avoid single point of failure..

1. Local disk + network storage, OR
2. SSD + object storage (S3, GCS, etc.)

1

Keep **one copy in a different physical location** for disaster recovery.

1. Off-site backup server, cloud bucket, or standby in another data center.

Combine logical + physical + PITR

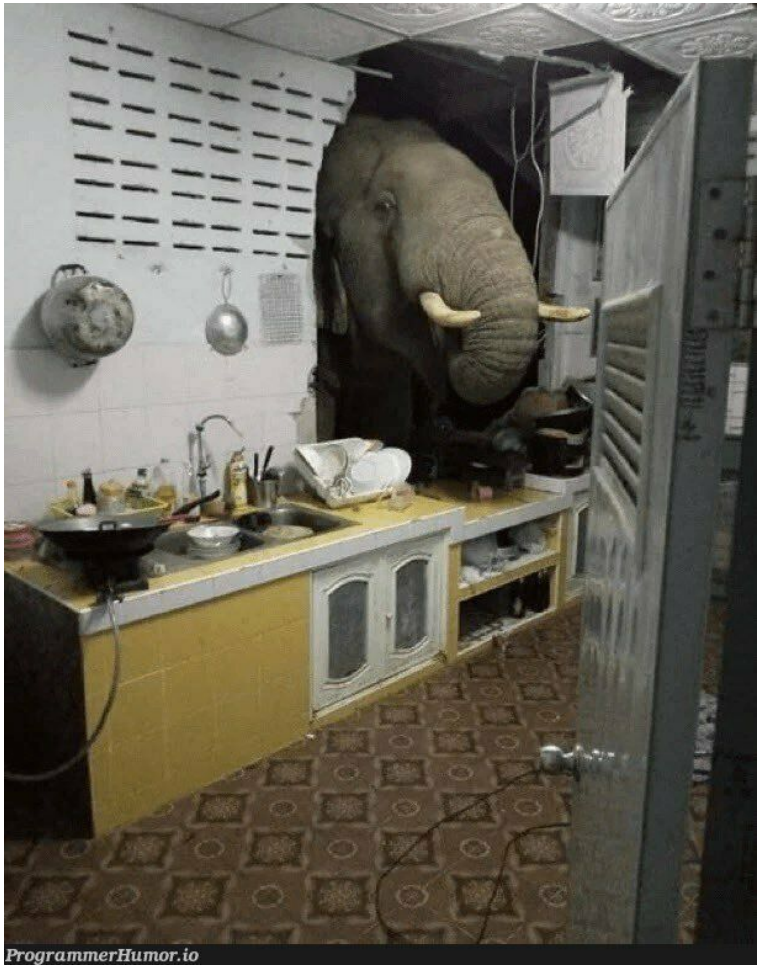
🐘🟰🦸

🔷 CTRL+Z Readiness” list

Can you really undo your next disaster?

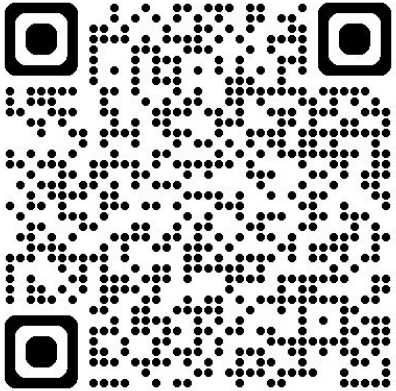
✅ Test Your Backups — Don’t Just Take Them
✅ Automate Daily & Weekly Backups
✅ Layered Protection — The “3-2-1 Rule”
✅ Pick the Right Strategy for Your Use Case
one size doesn’t fit all.
Choose between logical, physical, or incremental depending on RPO/RTO targets

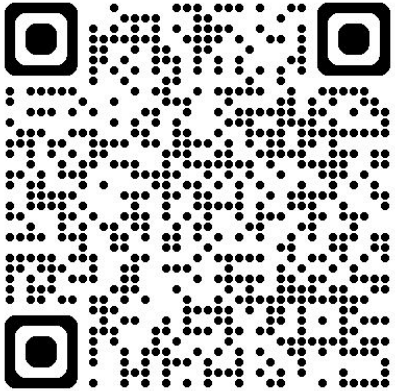
⚠️ Know the Caveats

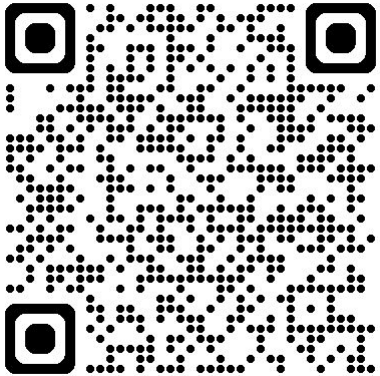


“True CTRL + Z readiness isn’t about having backups -
it’s about knowing they’ll work when you need them.”

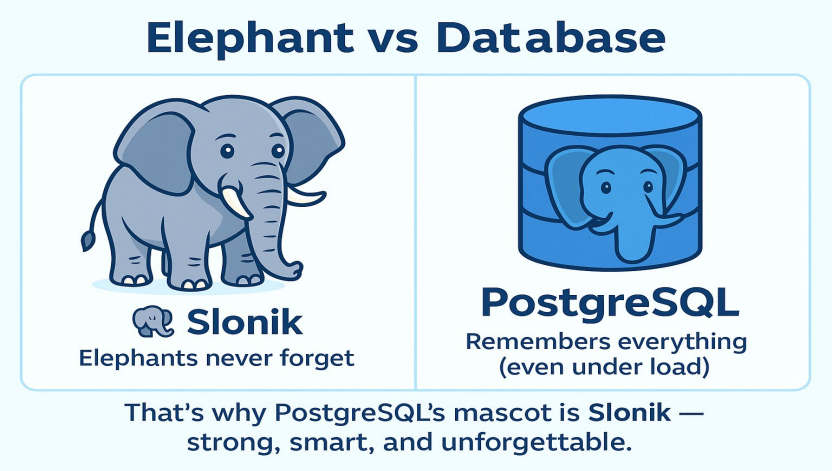
Future References :







“PostgreSQL’s mascot **(Slonik)** is an elephant because it never forgets…



unless you forget to back it up.”

Feedback

Thank you

Connect with me🙋‍♀️