



INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

Incremental Backup with PostgreSQL 17

Marion Baumgartner

Marion Baumgartner

GIS Software Developer [@Camptocamp](#)



 <https://github.com/marionb>

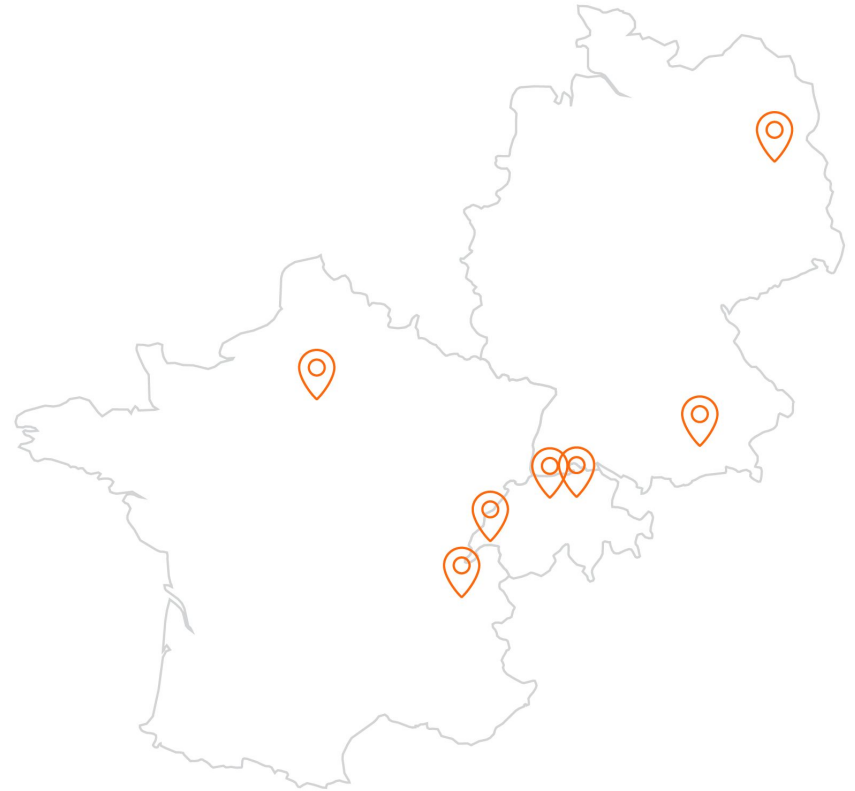
 marion.baumgartner@camptocamp.com

About Camptocamp

Your partner for success



- Founded in **2001**
- **180+** employees
- **3 countries:**
 - Switzerland, France, Germany
- A major European player in **Open Source** technologies
- Independent subsidiary of the **Swisscom Group**





Incremental Backup

What Incremental Backup is and is not



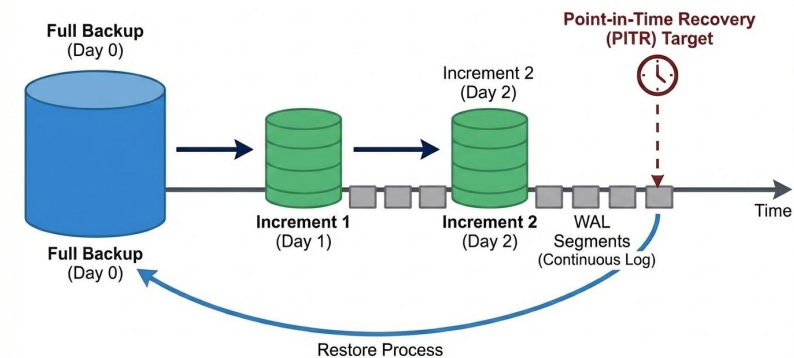
Not the WAL segment , Xlog, Archive,

...

WAL segments are still needed for recovery

This new feature only affect base backup (full backup)

PostgreSQL Incremental Backup & PITR Workflow



Why Incremental Backup



Use case 1 – Reduce Recovery Time Objective (RTO)

RTO: Recovery duration

Typical recovery process

- Restore the last full backup
- Replay WAL logs

Problem: The more transactions you have, the longer the recovery

- Recovery time is proportional to write traffic
- More traffic → more WAL to replay → longer RTO

Why Incremental Backup



Use case 1 – Reduce Recovery Time Objective (RTO)

Naive solution

- Take full backups closer to the RPO (every hour ?)
- Duration and resource usage are proportional to cluster size

Solution: Incremental backups

- Backup only the delta since the previous backup
- Backup duration is bounded, not proportional to data size

Why Incremental Backup



Use case 2 – Large Clusters & Backup Window Constraints

Context

- Very large PostgreSQL cluster
- Nightly full backup does not fit in the night
- Backup overlaps with business hours
- Heavy I/O usage impacts cluster performance

Why Incremental Backup



Use case 2 – Large Clusters & Backup Window Constraints

Naive approach

- Run full backups only on weekends
- ✓ Enough time to complete
- ✗ Recovery Time Objective explodes
 - Restore last full backup
 - Replay a full week of WAL
- Backup duration is proportional to cluster size.
- RTO is proportional to WAL volume.



Why Incremental Backup

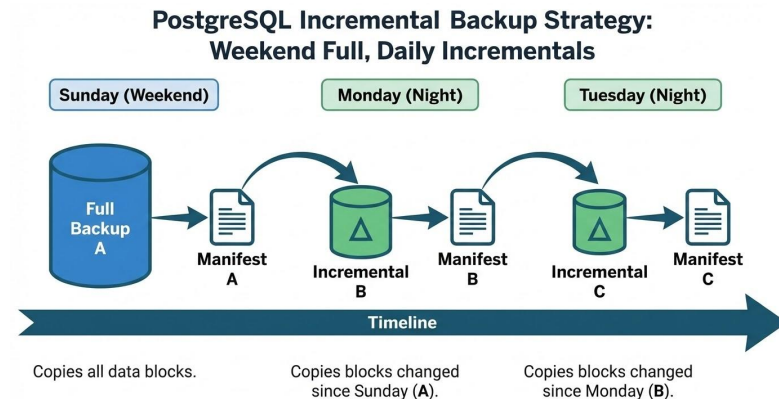
Use case 2 – Large Clusters & Backup Window Constraints

Solution: Incremental backups

- Full backup on the weekend
- Incremental backups every night
- Only changes since the previous backup are copied

Benefits

- Weekday backups fit in the night
- Limited I/O pressure on the cluster
- Recovery time is bounded
- No need to replay a full week of WAL



Why Incremental Backup

Reduce Recovery Time Objective (RTO)



Subsequent backup gets smaller and faster.

Saving only what changed since the previous incremental run.

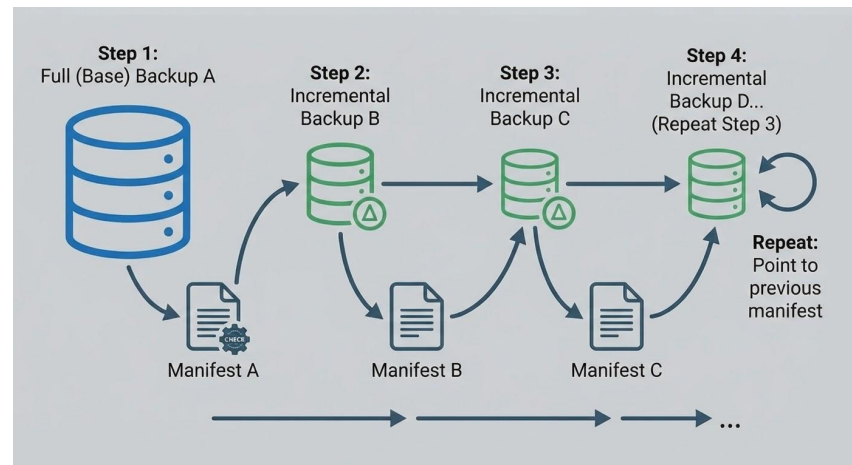


How it works

How to Backup

The basic idea

1. Take a full backup as a starting point
2. Take an incremental backup
3. Take another incremental backup
4. Repeat step 3



WAL file summarization needs to be enabled

```
summarize_wal = on
```



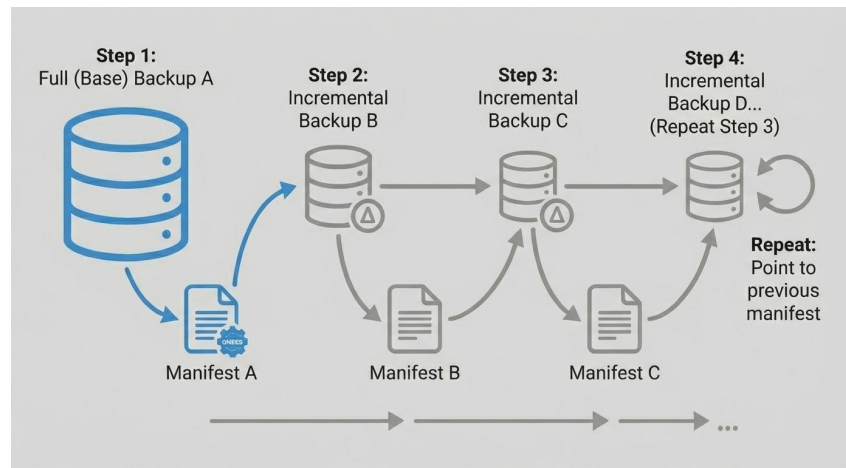
How to Backup

Step 1: The Full Backup

```
pg_basebackup
```

```
--pgdata=/mnt/full_backup
```

- PostgreSQL copies all necessary data files to create a standalone snapshot



How to Backup

pg_basebackup



```
pg_basebackup --pgdata=/mnt/full_backup
```

- `--pgdata` specifies the target directory where the backup will be stored
- Here: the output will be written to `/mnt/full_backup`

How to Backup

backup_label



Indicates the location in the WAL where the backup starts, and so is the checkpoint location

backup_label:

```
START WAL LOCATION: 0/4000028 (file  
00000001000000000000000004)
```

```
CHECKPOINT LOCATION: 0/4000080
```

```
BACKUP METHOD: streamed
```

```
BACKUP FROM: primary
```

```
START TIME: 2025-04-24 09:08:31 UTC
```

```
LABEL: pg_basebackup base backup
```

```
START TIMELINE: 1
```

How to Backup

backup_manifest



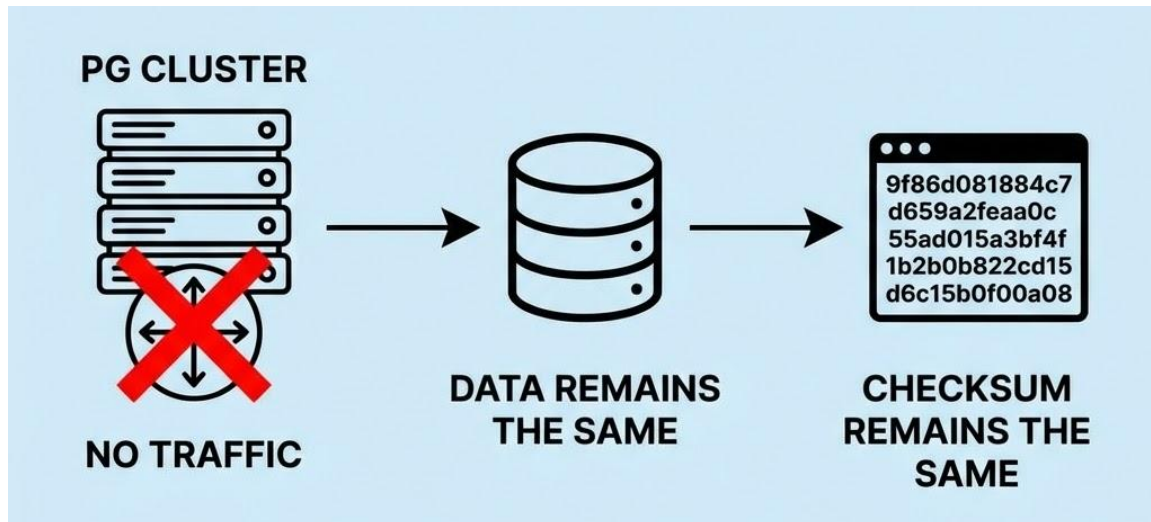
Serves as a reference to determine which files should be included in the incremental backup

backup_manifest:

```
{  
  "Path": "base/16384/3766",  
  "Size": 16384,  
  "Last-Modified": "2025-04-24 08:56:45 GMT",  
  "Checksum-Algorithm": "CRC32C",  
  "Checksum": "3c0ea625"  
},
```

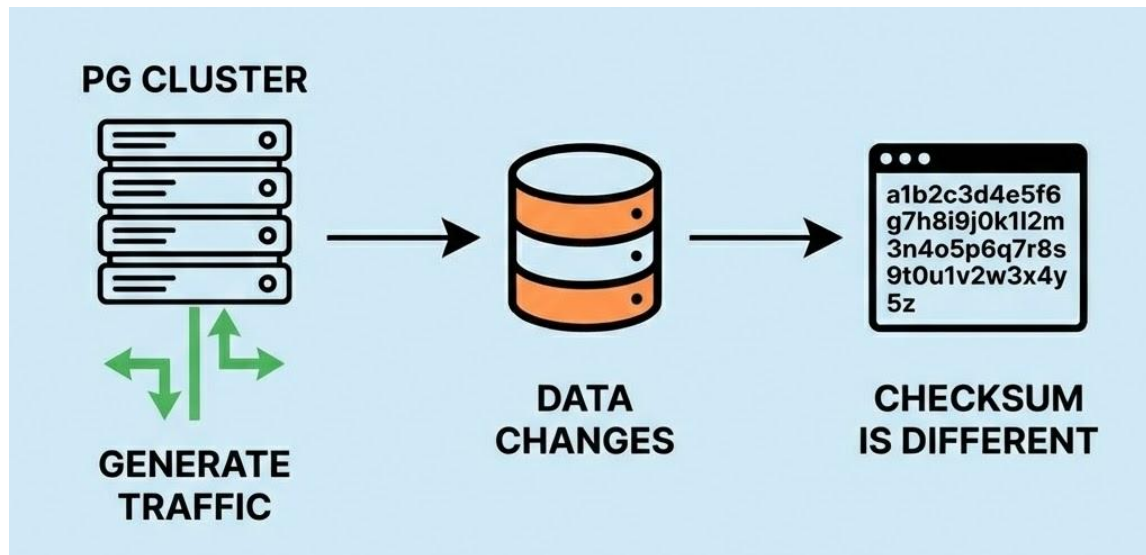
How to Backup

Understanding the Checksum



How to Backup

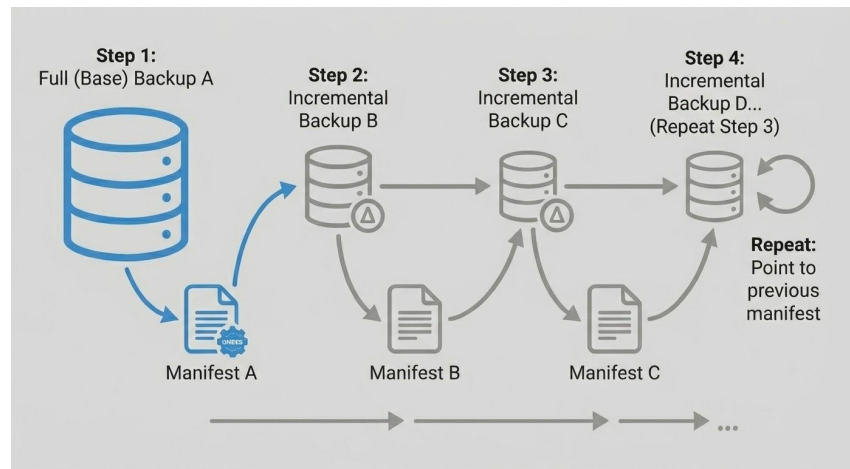
Understanding the Checksum



How to Backup

Step 1: The Full Backup - getting back

```
pg_basebackup  
--pgdata=/mnt/full_backup
```



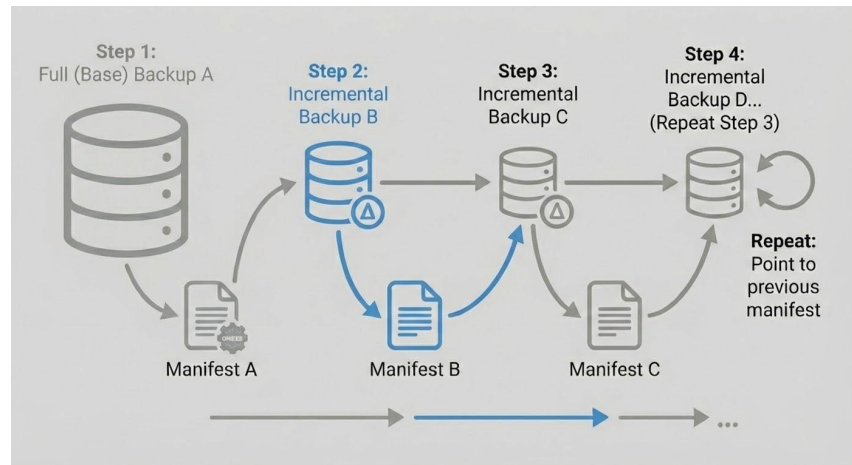
How to Backup



Step 2: The First Incremental

```
pg_basebackup \
--incremental=/mnt/full_backup/backup_manifest \
--pgdata=/mnt/incremental_backup/0/
```

- Depends on the manifest file from the full backup
- Any block that changed since full backup is copied into the new backup.



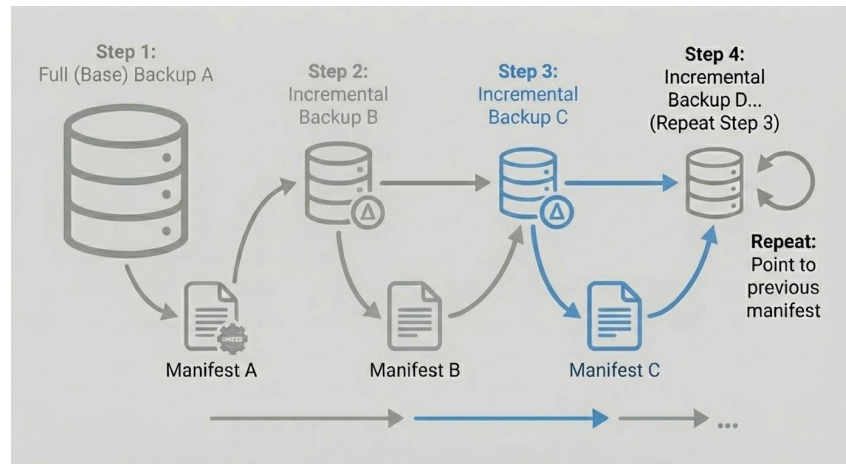
How to Backup



Step 3: The Second Incremental

```
pg_basebackup \  
--incremental=/mnt/0/backup_manifest \  
--pgdata=/mnt/incremental_backup/1/
```

- References the previous increment
- Server looks at the manifest from increment B.
- Copies blocks that have changed between increment B and right now.
- Only contain the diff from the first incremental.

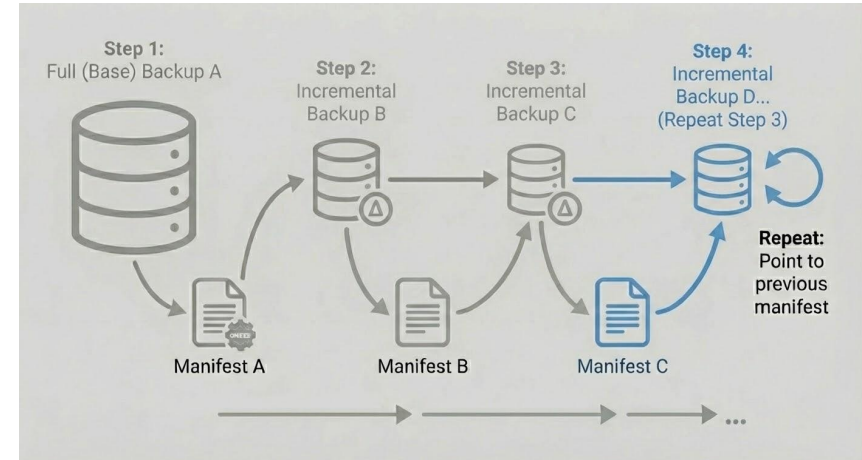


How to Backup

Step 4: Repeat ...



→ Repetition of the incremental backup step ...





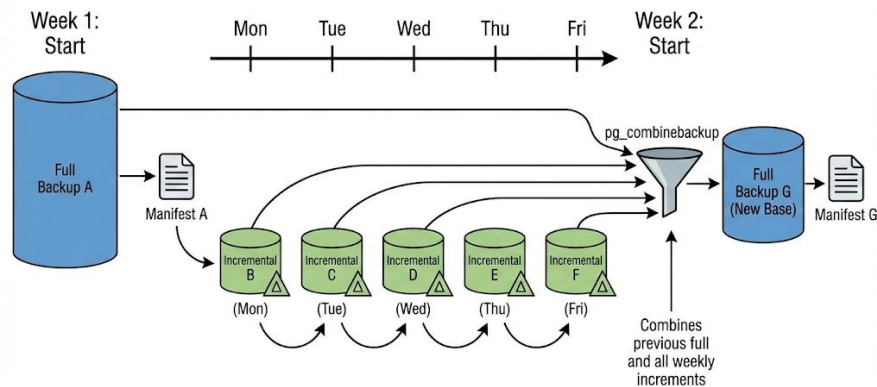
How to Backup

How often is a full backup needed?

Different scenarios:

- One full backup once a week and one incremental backup every day.
- One nightly full backup and incremental backups on an hourly rate.
- Re-create a full backup from the increments periodically and use it as the new full backup

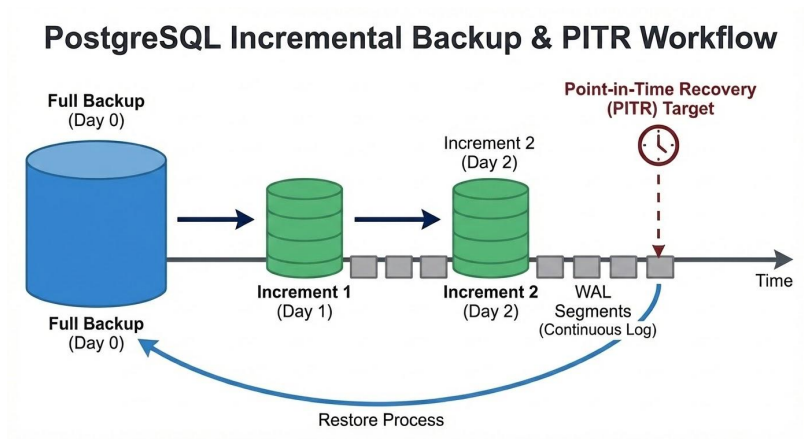
PostgreSQL Weekly Rolling Full Backup Strategy with Increments



How often to create an Increment



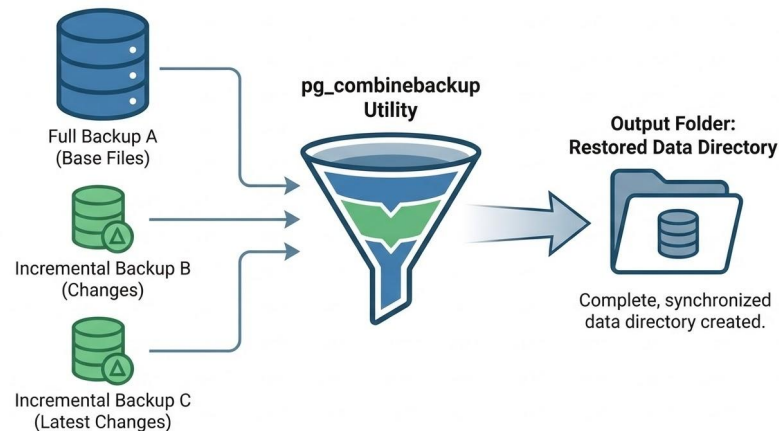
- Reply of WAL file is consuming part:
 - reduce the number of WAL to replay
- One possibility:
 - make the last incremental backup as close as possible to the recovery target.
- Solution with incremental backups:
 - just the tables that contain modification in comparison to the last incremental backup.



How to Restore

The basic idea

1. `pg_combinebackup`
2. take the base files from A (full)
3. apply the changes from B
4. apply the changes from C
5. create a data directory in the output folder.



How to Restore

Combine the Increments



```
pg_combinebackup -d -o /mnt/data/restore \  
/mnt/full_backup /mnt/incremental_backup/0 \  
/mnt/incremental_backup/1
```

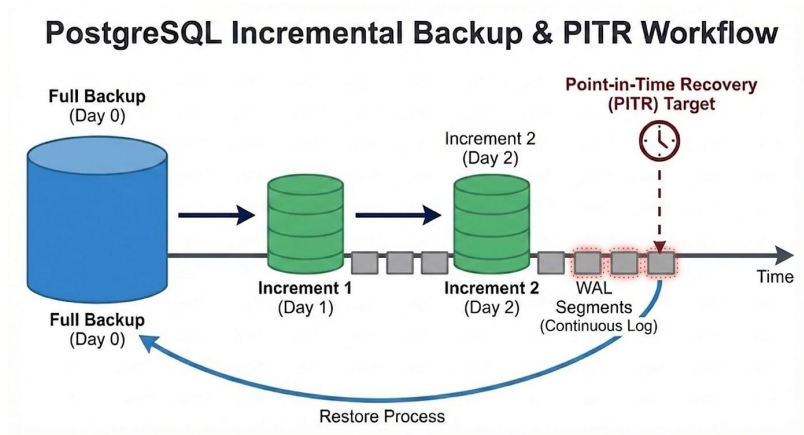
- follow the correct backup sequence:
 - start with the full backup,
 - followed by incremental backups in order.
- will give us the full backup in `/mnt/data/restore`
- Can be used to restore the DB

How to Restore WAL segments



Steps:

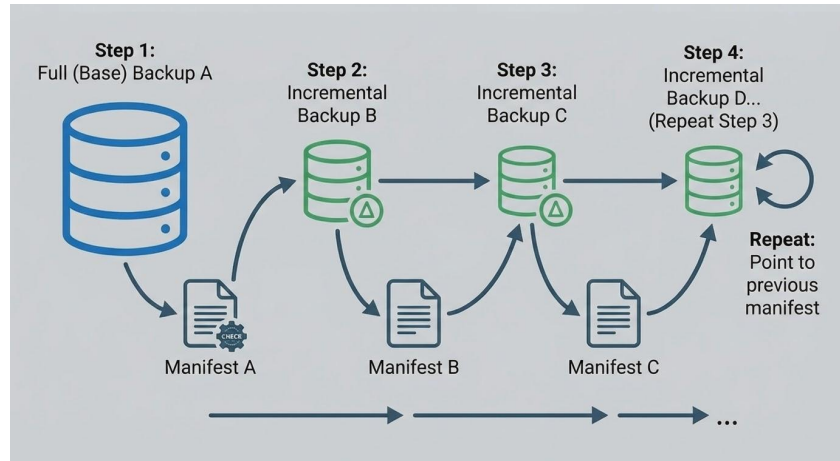
- Launch postgres server
- Configure `restore_command`
- Replay Wal segment



Summary



- Each subsequent backup gets smaller
- Only what changed since the last incremental run is saved





Pros

- Smaller size of backups
- pg_basebackup is a core utility
- Only the final state of the block at the time of the backup is considered

Cons

- Increments are not standalone
 - The whole chain is needed to be available
 - Additional complexity in combining the increments
- Resources are needed to track the changed block → small but it exists
 - Checksums need to be active
- Needs database user with REPLICATION permission or is superuser

Sources and Links



- <https://www.postgresql.org/docs/18/app-pgbasebackup.html>
- <https://www.postgresql.org/docs/18/app-pgcombinebackup.html>
- <https://sqlbak.com/blog/incremental-postgresql-backup-step-by-step-guide/>
- Based on a POC done in sprint 2025 with my colleagues Julien Acroute and Julian Waddle on the incremental backup feature developed by Robert Haas
- Images used in the slides were generated using Google Gemini

Thanks for your attention.

Marion Baumgartner
marion.baumgartner@camptocamp.com

camptocamp[®]

INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS