Introduction to PostgreSQL Backups

data egret

Your remote PostgreSQL DBA team

Stefan FERCOT,
Valeria KAPLAN,
Ilya KOSMODEMIANSKY

contact@dataegret.com

SECURING YOUR DATABASE AVAILABILITY, SO THAT YOUR TEAM CAN FOCUS ON NEW FEATURE DEVELOPMENT.

- Migrations
- DB audit
- Performance optimisation
- Backup & restore
- Architectural review
- Advising Data Science teams
- Developer training

on premise & cloud

EXPERTISE

Senior DBA with **10+ years** of PostgreSQL administration **experience**

DEVELOPMENT

Involved in **new feature and extension development**

TAILORED APPROACH

Felxible approach and **dedicated team** focused on success of your project

COMMUNITY

Contributing Sponsor. Deeply involved in the PostgreSQL community

Data Egret GmbH | 66583 Spiesen-Elversberg | Germany

contact@DataEgret.com

About us

Stefan FERCOT,
Valeria KAPLAN,
Ilya KOSMODEMIANSKY

About the audience

- What’s your favorite backup tool?

Stefan Fercot

- Senior PostgreSQL Expert
- pgBackRest fan & contributor
- aka. pgstef
- https://pgstef.github.io

Need a Disaster and Recovery Plan? ;-)
*Contact **Data Egret** to talk to me about backups and high-availability!*

Agenda

- Why do we need backups?
- What is WAL?
- Point-In-Time Recovery (PITR)
  - WAL archives
  - File-system-level backup
  - Restore key points & timelines

Why do we need backups?

- what could go wrong?
  - infrastructure failure (storage, network)
  - human error
  - system or data corruption

`pg_dump`

- logical export
  - database level
  - object precision: tables, sequences,…
  - consistency across multiple databases is a challenge
- restore
  - clean cluster (no bloat)
  - `CREATE INDEX` and stats collection (`ANALYSE`)!

data size ↗ performance ↘

Speaking about backups…

Why isn’t `pg_dump` enough?

Think beyond backups!

BACKUP STRATEGY + RECOVERY PROCEDURES

=

DATABASE DISASTER RECOVERY PLAN

Recovery Requirements

- **RPO** (*Recovery Point Objective*)
  - maximum amount of data that can be lost
- **RTO** (Recovery Time Objective)
  - maximum acceptable service disruption
- retention period

What do you need to consider?

higher requirements, higher investments

- physical backups and PITR
- *Streaming Replication*
- automated fail-over
- …

Next step?

- physical backups
  - copy of the database files from the file system
  - inconsistency protection using WAL
- Point-in-Time Recovery (PITR)
  - restore the database to a specific moment in time
  - using physical backups and WAL archives!

What is WAL?

- write-ahead log
  - transaction log (aka xlog)
- usually 16 MB (default)
  - `--wal-segsize` *initdb* parameter to change it
- `pg_wal` directory
- designed to prevent data loss in most situations

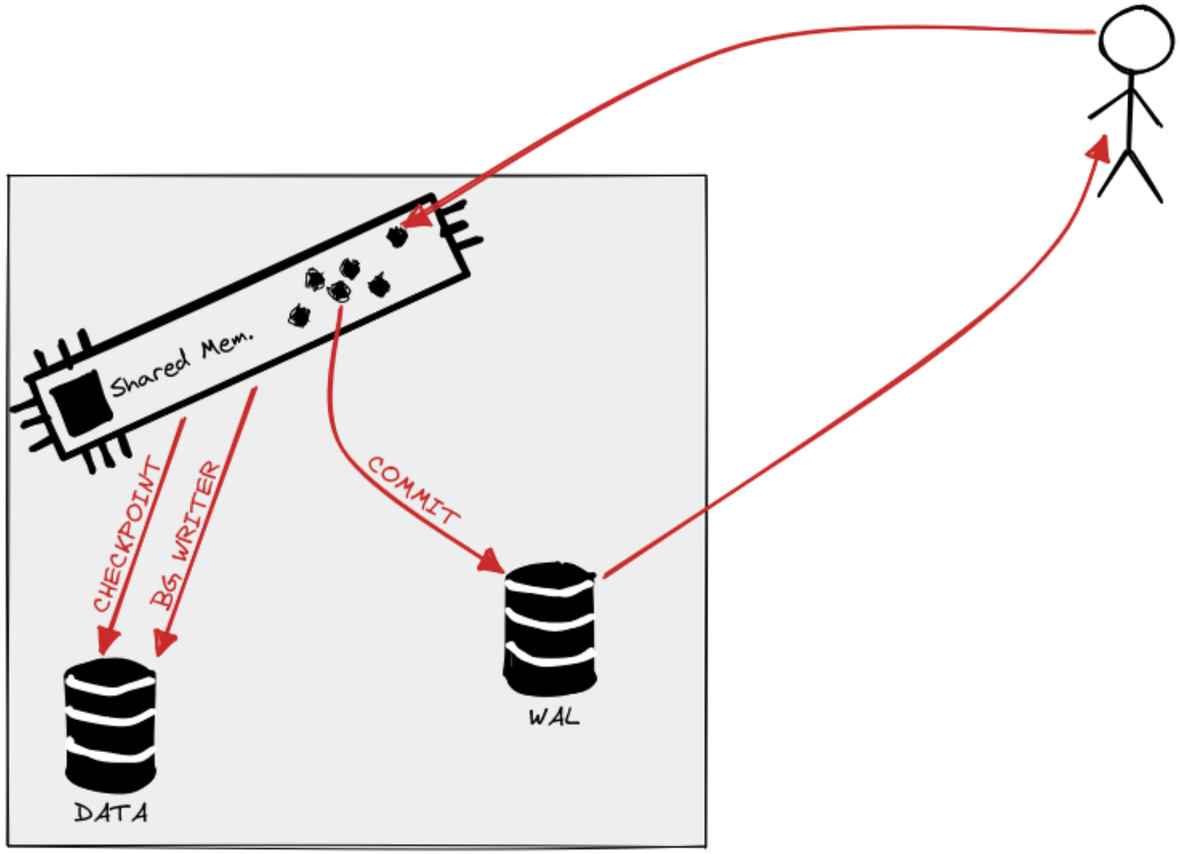
Write-Ahead Log (WAL)

- transactions written sequentially
  - COMMIT when data are flushed to disk
- WAL replay after a crash
  - make the database consistent

Data modifications

- transactions modify data in `shared_buffers`
- checkpoints and background writer…
  - … push all dirty buffers to the storage

Data modifications (2)



LSN

- log sequence number
  - position of the record in WAL file
  - provides uniqueness for each WAL record

```
=# SELECT pg_current_wal_lsn();
 pg_current_wal_lsn
--------------------
 2/3002020
(1 row)

=# SELECT pg_walfile_name(pg_current_wal_lsn());
     pg_walfile_name
--------------------------
 000000010000000200000003
(1 row)
```

WAL filename

- 000000010000000200000003
  - 00000001 : timeline
  - 00000002 : wal
  - 00000003 : segment
- hexadecimal
  - 000000010000000000000001
  - 00000001000000000000FF
  - 000000010000000100000000
  - …

Questions?

Point-In-Time Recovery (PITR)

- combine
  - file-system-level backup
  - continuous archiving of WAL files
- restore the file-system-level backup and replay archived WAL files

Benefits

- live backup
- less data-losses
- not mandatory to replay WAL entries all the way to the end

Drawbacks

- complete cluster backup…
  - … and restore
- big storage space (data + WAL archives)
- WAL clean-up blocked if archiving fails
- not as simple as `pg_dump`

WAL archives

- 2 possibilities
  - archiver process
  - `pg_receivewal` (via *Streaming Replication*)

Archiver process

- configuration (`postgresql.conf`)
  - `wal_level = replica`
  - `archive_mode = on` or `always`
  - `archive_command = '... some command ...'`
  - `archive_timeout = 0`
- don’t forget to flush the file on disk!

`pg_receivewal`

- archiving via *Streaming Replication*
- writes locally WAL files
- supposed to get data faster than the archiver process
- replication slot advised!

Benefits and drawbacks

- archiver process
  - easy to setup
  - maximum 1 WAL possible to lose
- `pg_receivewal`
  - more complex implementation
  - only the last transactions are lost

File-system-level backup

- `pg_basebackup`
- manual steps

pg_basebackup

- takes a file-system-level copy
  - using *Streaming Replication* connection(s)
- collects WAL archives during (or after) the copy
- incremental backups should land in v17!

```
$ pg_basebackup --format=tar --wal-method=stream \
 --checkpoint=fast --progress -h HOSTNAME -U NAME \
 -D DIRECTORY
```

Manual steps

- `pg_backup_start()`
- manual file-system-level copy
- `pg_backup_stop()`

`pg_backup_start()`

```
SELECT pg_backup_start (
```

- `label` : arbitrary user-defined text
- `fast` : immediate checkpoint?

```
)
```

Data copy

- copy data files while PostgreSQL is running
  - *PGDATA* directory
  - tablespaces
- inconsistency protection with WAL archives
- ignore
  - `postmaster.pid`, `postmaster.opts`, `pg_internal.init`
  - `log`, `pg_wal`, `pg_replslot`,…
- don’t forget configuration files!

`pg_backup_stop()`

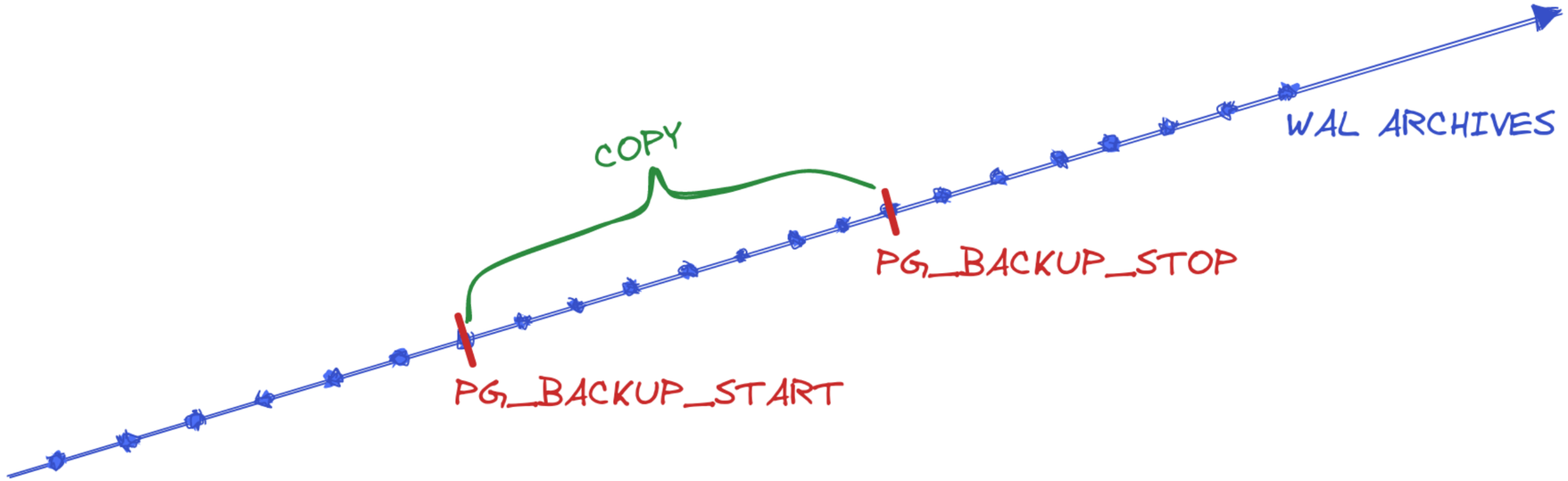
```
SELECT * FROM pg_backup_stop (
```

- wait_for_archive

```
)
```

- executed in the same connection as `pg_backup_start()`!
- returns `backup_label` and `tablespace_map` content

Summary



Caution word

Don’t do it by hand, use backup (and restore) tools !

Questions?

Restore key points

- data files
- recovery configuration

Recovery steps (1)

- stop the server if it’s running
- keep a temporary copy of your PGDATA and tablespaces
  - or at least the `pg_wal` directory

Recovery steps (2)

- restore database files from your file system backup
  - pay attention to ownership and permissions
  - verify tablespaces symbolic links
- remove content of `pg_wal` (if not already the case)
- copy unarchived WAL segment files

Recovery steps (3)

- configure the recovery…
  - `postgresql.conf` + `recovery.signal`
- `restore_command = '... some command ...'`

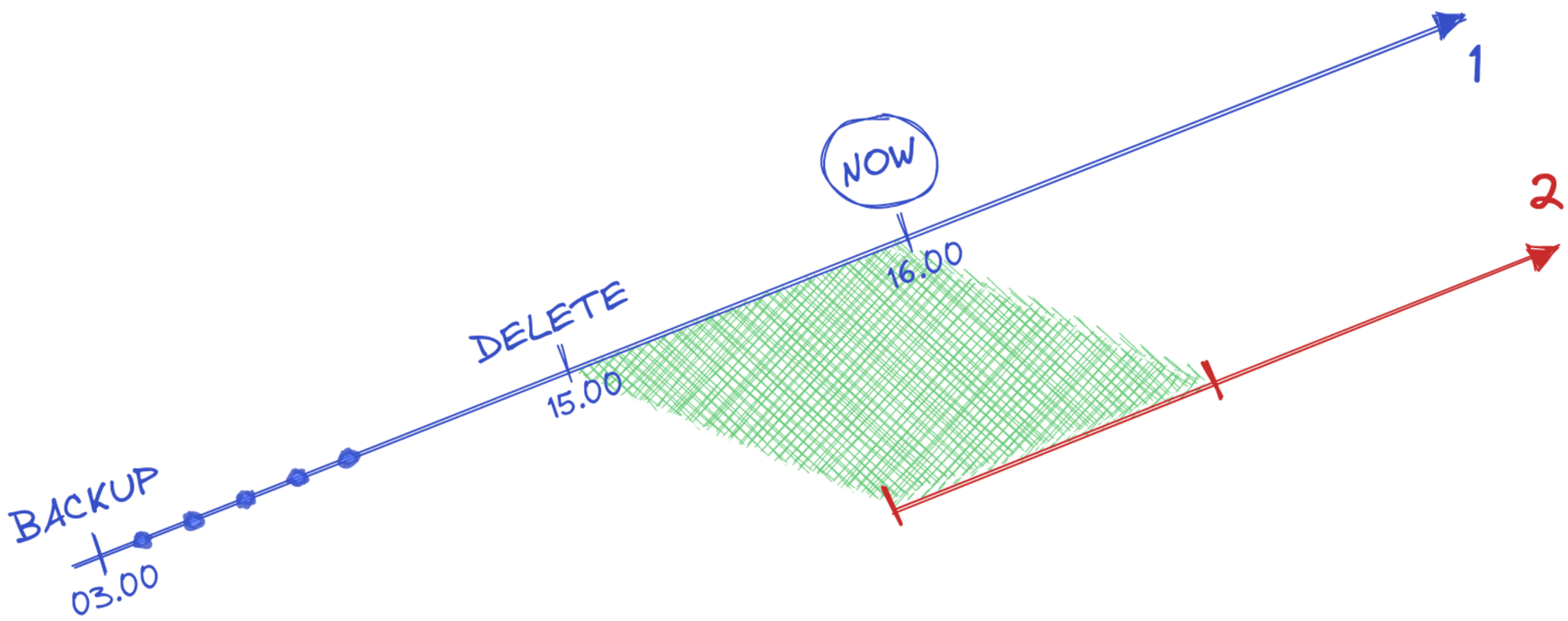
Recovery steps (4)

- start the server
- watch the restore process
  - until consistent recovery state reached

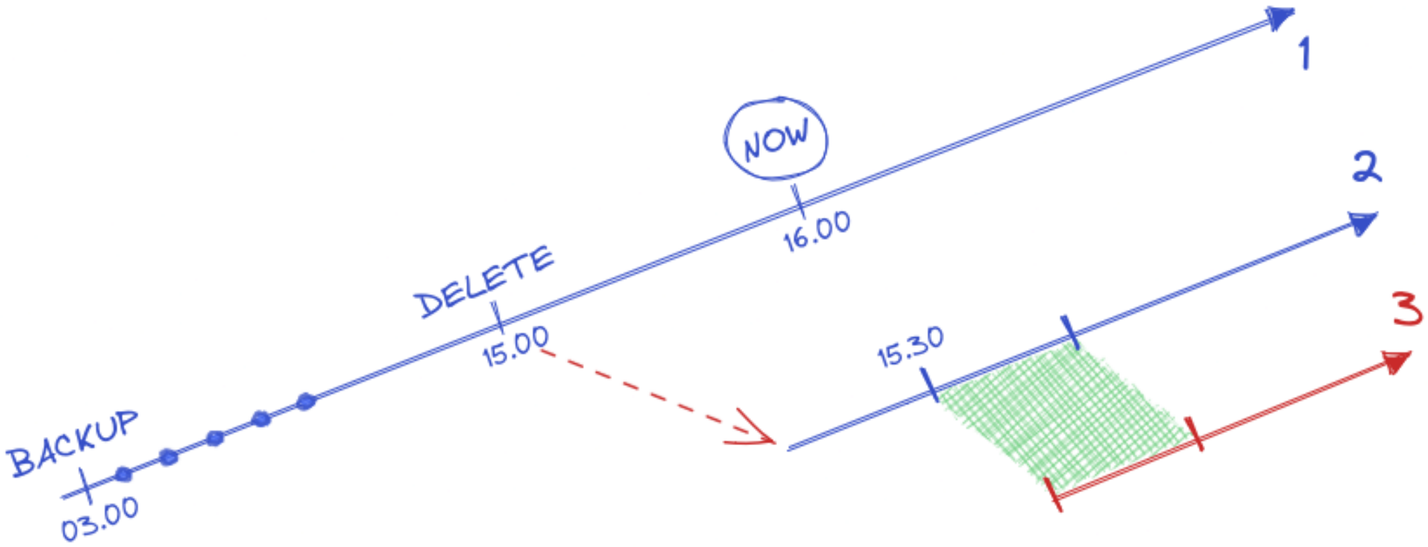
Timelines

- archive recovery complete -> new timeline
  - part of WAL segment file names
  - to identify the series of WAL records generated after that recover
  - `.history` files
- `recovery_target_timeline`
  - default: `latest` (v12+) or `current` (< v12)

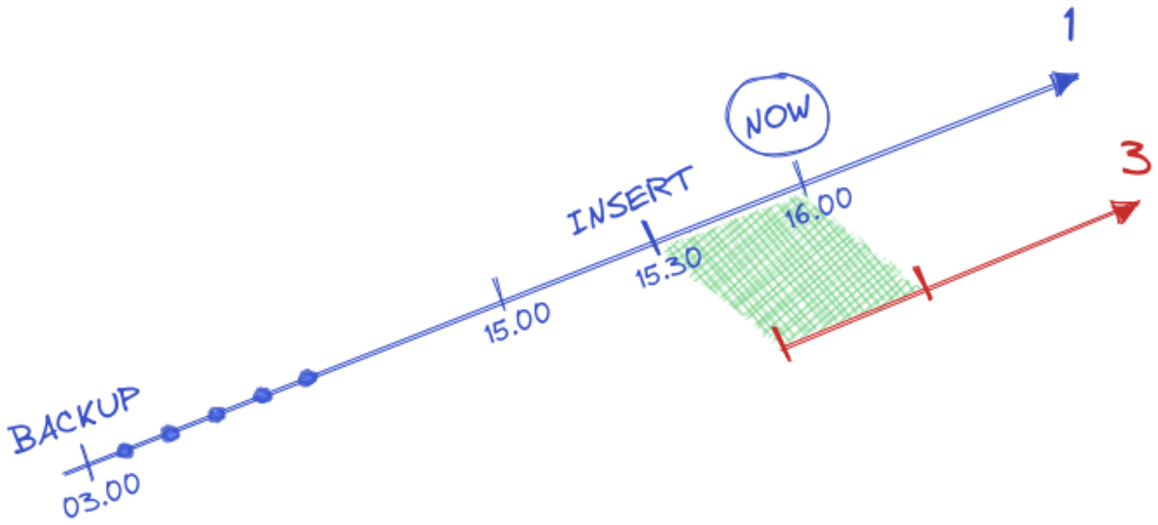
Timelines explanation



Timelines explanation (2)



Timelines explanation (3)



Conclusion

- PITR is
  - reliable
  - fast[er than `pg_dump`]
  - continuous
- the answer is in the PostgreSQL logs!
- tools make life easier…

SECURING YOUR DATABASE AVAILABILITY, SO THAT YOUR TEAM CAN FOCUS ON NEW FEATURE DEVELOPMENT.

- Migrations
- DB audit
- Performance optimisation
- Backup & restore
- Architectural review
- Advising Data Science teams
- Developer training

on premise & cloud

EXPERTISE

Senior DBA with **10+ years** of PostgreSQL administration **experience**

DEVELOPMENT

Involved in **new feature and extension development**

TAILORED APPROACH

Felxible approach and **dedicated team** focused on success of your project

COMMUNITY

Contributing Sponsor. Deeply involved in the PostgreSQL community

Data Egret GmbH | 66583 Spiesen-Elversberg | Germany

contact@DataEgret.com