Point-in-time Recovery, target 2022

PGConf Belgium 2022

Stefan FERCOT

Thu May 19th, 2022

Who Am I?

- Stefan Fercot
- aka. pgstef
- https://pgstef.github.io
- PostgreSQL user since 2010
- pgBackRest fan & contributor
- Database Backup Architect @EDB

EDB™

Agenda

- What is WAL?
- Point-In-Time Recovery (PITR)
  - WAL archives
  - File-system-level backup
  - Restore
- PITR Tools

EDB™

What is WAL?

- write-ahead log
  - transaction log (aka xlog)
- usually 16 MB (default)
  - `--wal-segsize` *initdb* parameter to change it
- pg_xlog (<= v9.6) -> pg_wal (v10+)
- designed to prevent data loss in most situations

EDB™

Write-Ahead Log (WAL)

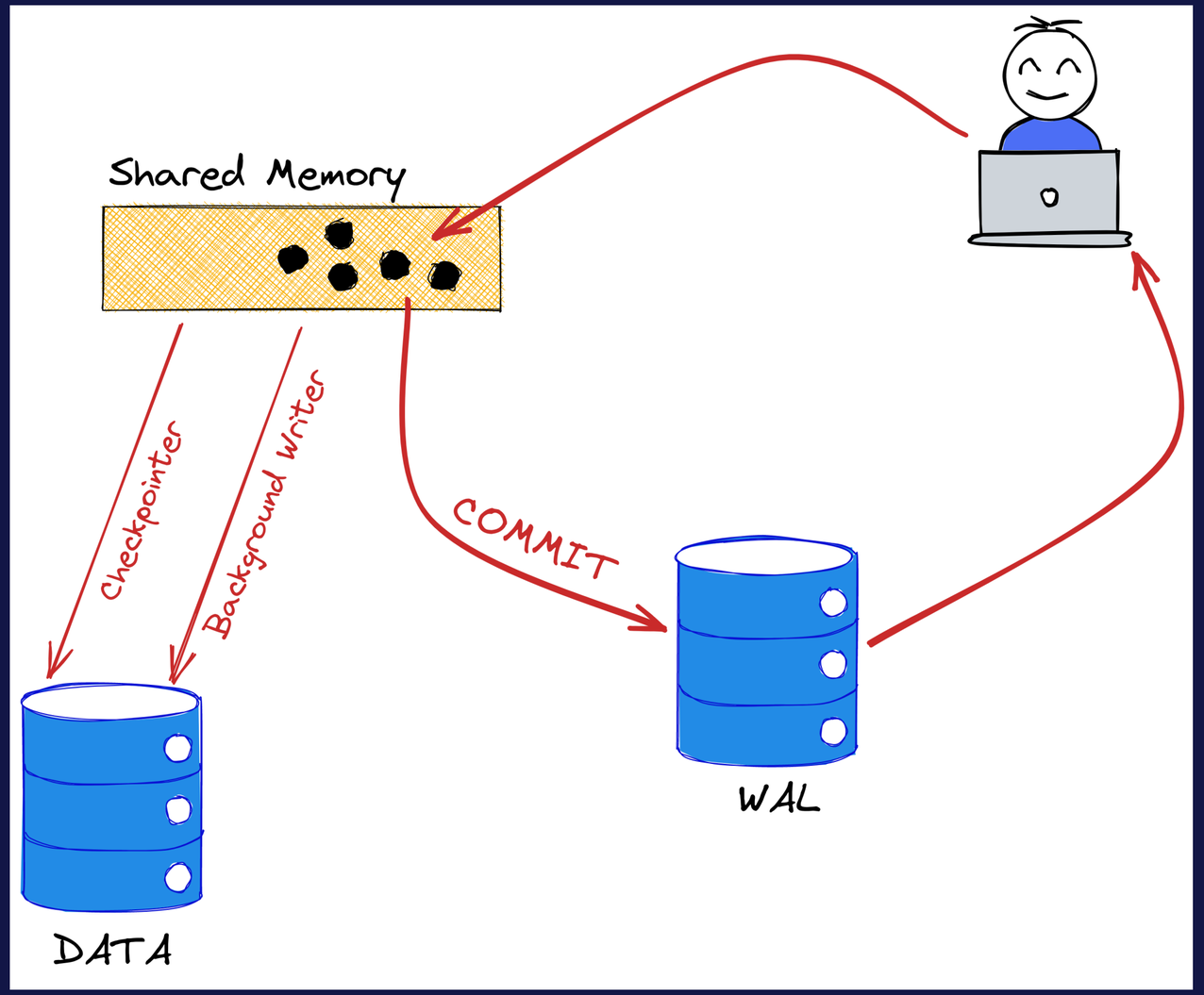
- transactions written sequentially
  - COMMIT when data are flushed to disk
- WAL replay after a crash
  - make the database consistent

EDB™

Data modifications

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

Data modifications (2)



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

EDB™

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

Archive library

- upcoming in v15
- running `archive_command` is slow
  - mostly because of *system()* calls
- `archive_library = 'basic_archive'`
  - option to call a loadable module for each file to be archived
  - rather than running a shell command

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
- more compression types and server side compression
  - upcoming in v15
- no incremental backup (yet)

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

EDB™

Manual steps

- `pg_start_backup()`
- manual file-system-level copy
- `pg_stop_backup()`

`pg_start_backup()`

```
SELECT pg_start_backup (
```

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

```
)
```

EDB™

Exclusive mode

- easy to use but deprecated since 9.6
- `pg_start_backup()`
  - writes `backup_label`, `tablespace_map`
- works only on primary servers

Non-exclusive mode

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

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!

EDB™

`pg_stop_backup()`

```
SELECT * FROM pg_stop_backup (
```

- exclusive
- wait_for_archive

```
)
```

- on primary server
  - automatic switch to the next WAL segment
- on standby server
  - consider using `pg_switch_wal()` on the primary…

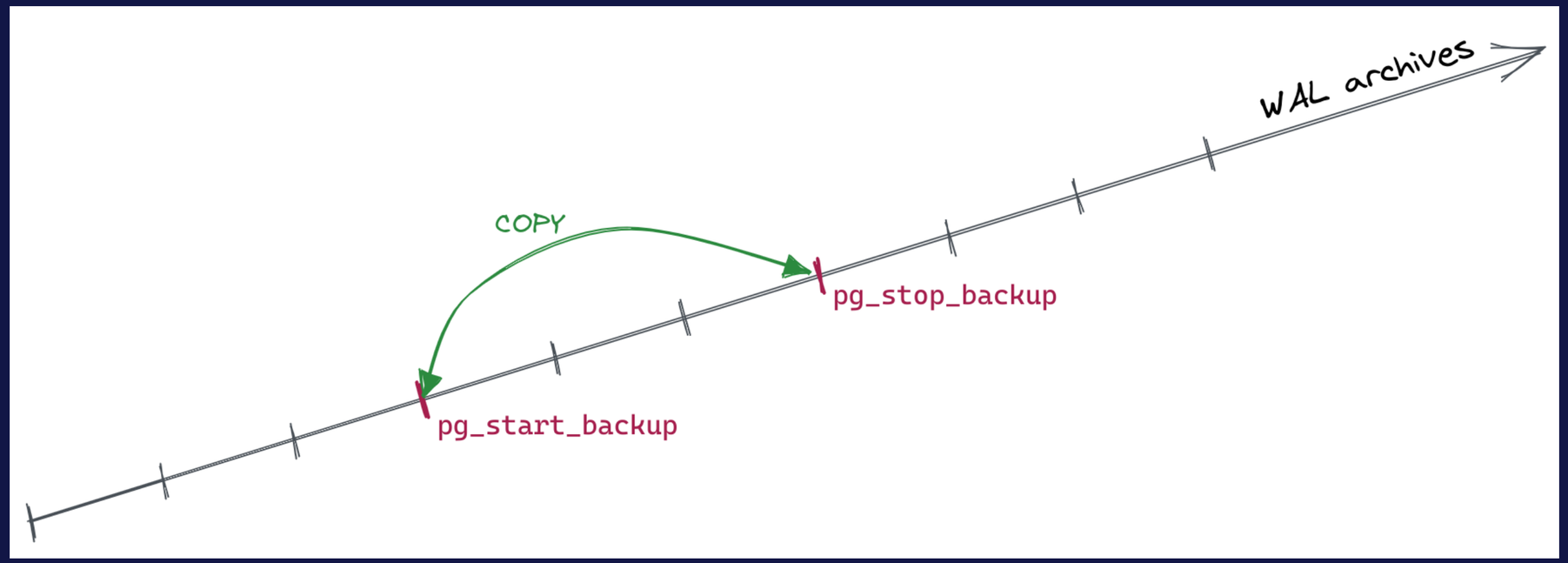
PostgreSQL 15 - exclusive mode removed

#39969e2a1e4d7f5a37f3ef37d53bbfe171e7d77a

- exclusive mode removed
- breaking change
  - `pg_backup_start()`
  - `pg_backup_stop()`

EDB™

Summary



Restore

- recovery procedure is simple but…
  - must be followed carefully!

Recovery steps (1/5)

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

Recovery steps (2/5)

- 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 (3/5)

- configure the recovery…
  - before v12: `recovery.conf`
  - after: `postgresql.conf` + `recovery.signal`
- `restore_command = '... some command ...'`
- prevent ordinary connections in `pg_hba.conf` if needed

> *PostgreSQL 12 integrates recovery.conf into postgresql.conf*

EDB

Recovery steps (4/5)

- recovery target:
  - `recovery_target_name`, `recovery_target_time`
  - `recovery_target_xid`, `recovery_target_lsn`
  - `recovery_target_inclusive`
- timeline to follow:
  - `recovery_target_timeline`
- action once recovery target is reached?
  - `recovery_target_action`

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)
```

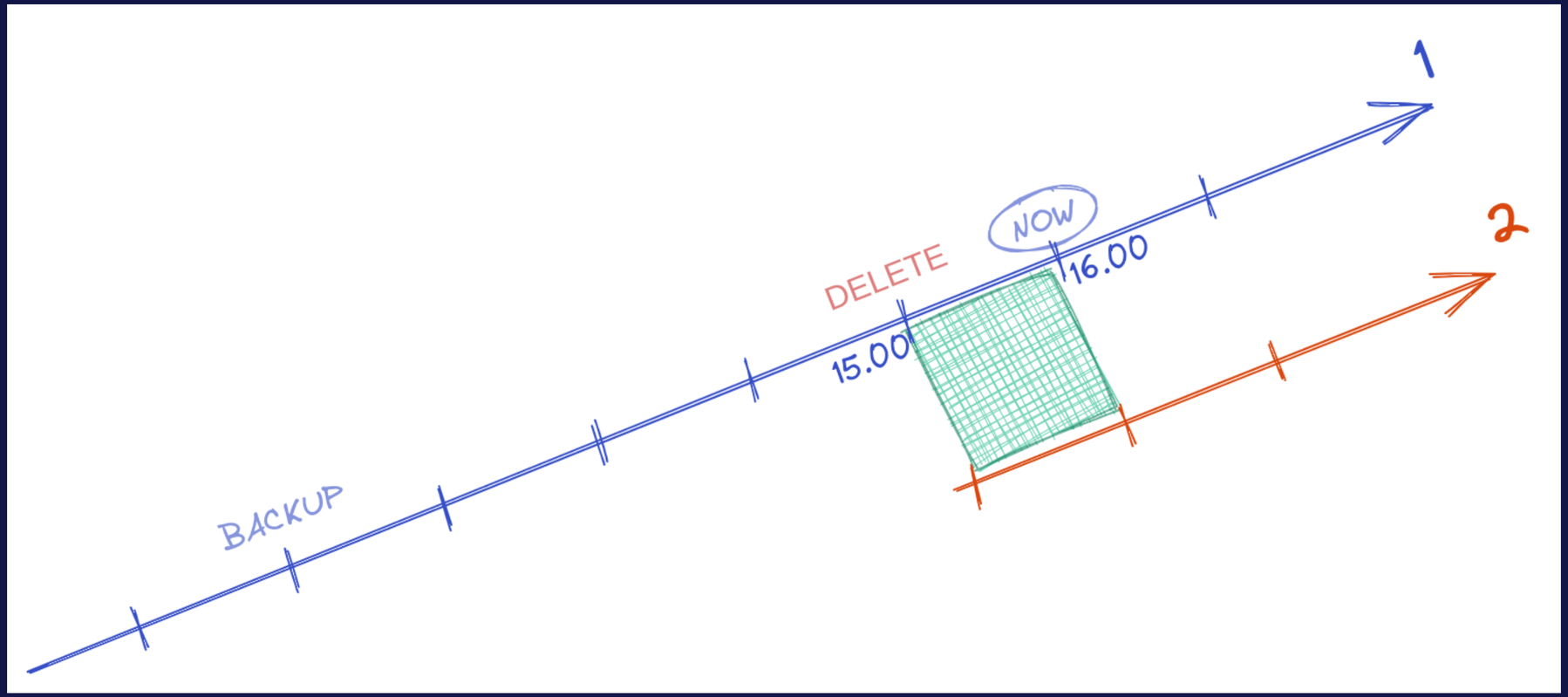
WAL filename

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

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 (2)



Recovery steps (5/5)

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

Recovery target reached?

- `recovery_target_action`
  - *pause*, default: recovery paused
  - *promote*: recovery process will finish and server will accept connections
  - *shutdown*: server stopped
- paused state can be resumed by using `pg_wal_replay_resume()`

EDB™

Sample outputs

- missing `recovery.signal`
- recovery target not found
- recovery target reached
- timeline switch

Missing recovery.signal

- WAL needed for consistency still exists in `pg_wal`?
  - if not, use `restore_command`…
  - …if `recovery.signal` exists!

```
LOG:  invalid checkpoint record
FATAL:  could not locate required checkpoint record
HINT:  If you are restoring from a backup, touch "...data/recovery.signal"
  and add required recovery options.
  If you are not restoring from a backup, try removing the file
  "...data/backup_label".
  Be careful: removing "...data/backup_label" will result in a corrupt cluster if
  restoring from a backup.
```

EDB™

Recovery target not found

```
LOG:  starting point-in-time recovery to "RP1"
LOG:  restored log file "..." from archive
LOG:  redo starts at 0/3000028
LOG:  consistent recovery state reached at 0/3000100
LOG:  database system is ready to accept read-only connections
LOG:  restored log file "..." from archive
FATAL:  recovery ended before configured recovery target was reached
...
LOG:  database system is shut down
```

EDB™

Recovery target reached

```
LOG:  recovery stopping at restore point "RP1", time ...
LOG:  pausing at the end of recovery
HINT:  Execute pg_wal_replay_resume() to promote.

LOG:  selected new timeline ID: 2
LOG:  archive recovery complete
LOG:  database system is ready to accept connections
```

EDB™

Timeline switch

A correct restore from backup, PITR or not,…

…always involves a timeline switch!

```
LOG:  consistent recovery state reached at ...
LOG:  database system is ready to accept read only connections
LOG:  restored log file "..." from archive
...
LOG:  selected new timeline ID: 2
LOG:  archive recovery complete
LOG:  database system is ready to accept connections
```

EDB™

PITR Tools

- tools make life easier
  - pgBackRest
  - Barman
  - …
- providing
  - backup, restore, purge methods
  - archiving commands

EDB™

pgBackRest

- written in C
- local or remote operation (via SSH or TLS server)
- full/differential/incremental backup
- parallel and asynchronous operations
- S3, Azure, and GCS support
- …

EDB™

Barman

- written in Python
- remote backup and restore with rsync (via SSH)
  - or *Streaming Replication* protocol
- file level incremental backups with rsync
- `pg_receivewal` & `pg_basebackup` support
- `barman-cli-cloud` for S3, Azure and GCS access

What is a good backup tool?

- usable
  - documentation & support
  - out-of-box automation of various routines
- scalable
  - parallel execution
  - compression
  - incremental & differential backups
- reliable

EDB™

Key features comparison

	pgBackRest	Barman
archive_command	YES `archive-async`	YES
`pg_receivewal`	NO	YES

Incremental backups	YES
	`--type=incr`
	`--type=diff`
		YES
		rsync (hardlinks)

WAL archive compression	YES	YES
Backup compression	YES	NO

Key features comparison (2)

	pgBackRest	Barman
Symmetric encryption	YES	NO
Parallel backup and restore	YES	YES
Parallel archiving	YES	NO
Partial restore (only selected databases)	YES	NO

EDB™

Conclusion

- PITR is
  - reliable
  - fast[er than `pg_dump`]
  - continuous
- tools make life easier
  - choose wisely…
  - validate your backups!

EDB™

Questions?