

pgBackRest Advanced Archiving and Recovery Magic You'll Wish You Knew Sooner



data egret

Your remote PostgreSQL DBA team

Stefan FERCOT

contact@dataegret.com

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

- Migration
- DB audit
- Performance optimisation
- Backup & recovery
- Architectural review
- Consulting for data science and analytics teams
- Change Data Capture (CDC)
- Developer training



MITGLIED



EXPERTISE

Senior DBA with 10+ years of experience in PostgreSQL administration.



DEVELOPMENT

Involved in new PostgreSQL feature and extension development.



CUSTOMISATION

Flexible approach and dedicated team focused on success of your project.



COMMUNITY

Recognised as significant contributing sponsor to PostgreSQL.



Scan for Updates



oapg-edu.org

- Independent industry certification for PostgreSQL
- First launch at PGConf.DE 2025
- For more information visit booths of CYBERTEC, Data Egret and Proventa

Stefan Fercot

- Senior PostgreSQL Expert @Data Egret
- 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!*

pgBackRest

- aims to be a simple, reliable backup and **restore** system
- current release: 2.55 (2.55.1 - May 5, 2025)
- local or remote operation (via SSH or TLS server)
- parallel and asynchronous operations
- S3, Azure, and GCS support
- client-side encryption (aes-256-cbc)
- ...

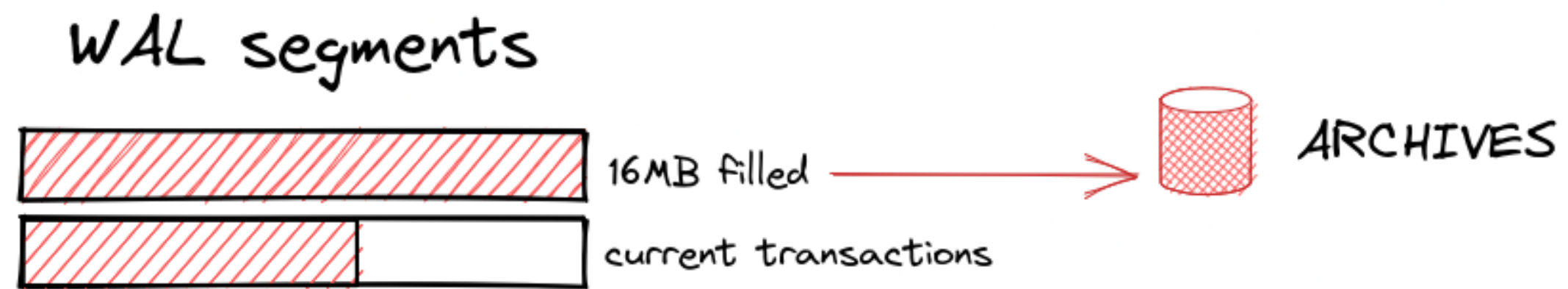
pgBackRest Advanced Archiving and Recovery Magic

- archiving system
- restore possibilities

Archiving system

- `archive_command` vs `pg_receivewal` (not supported)
- how to improve it?
- what can go wrong with archiving?

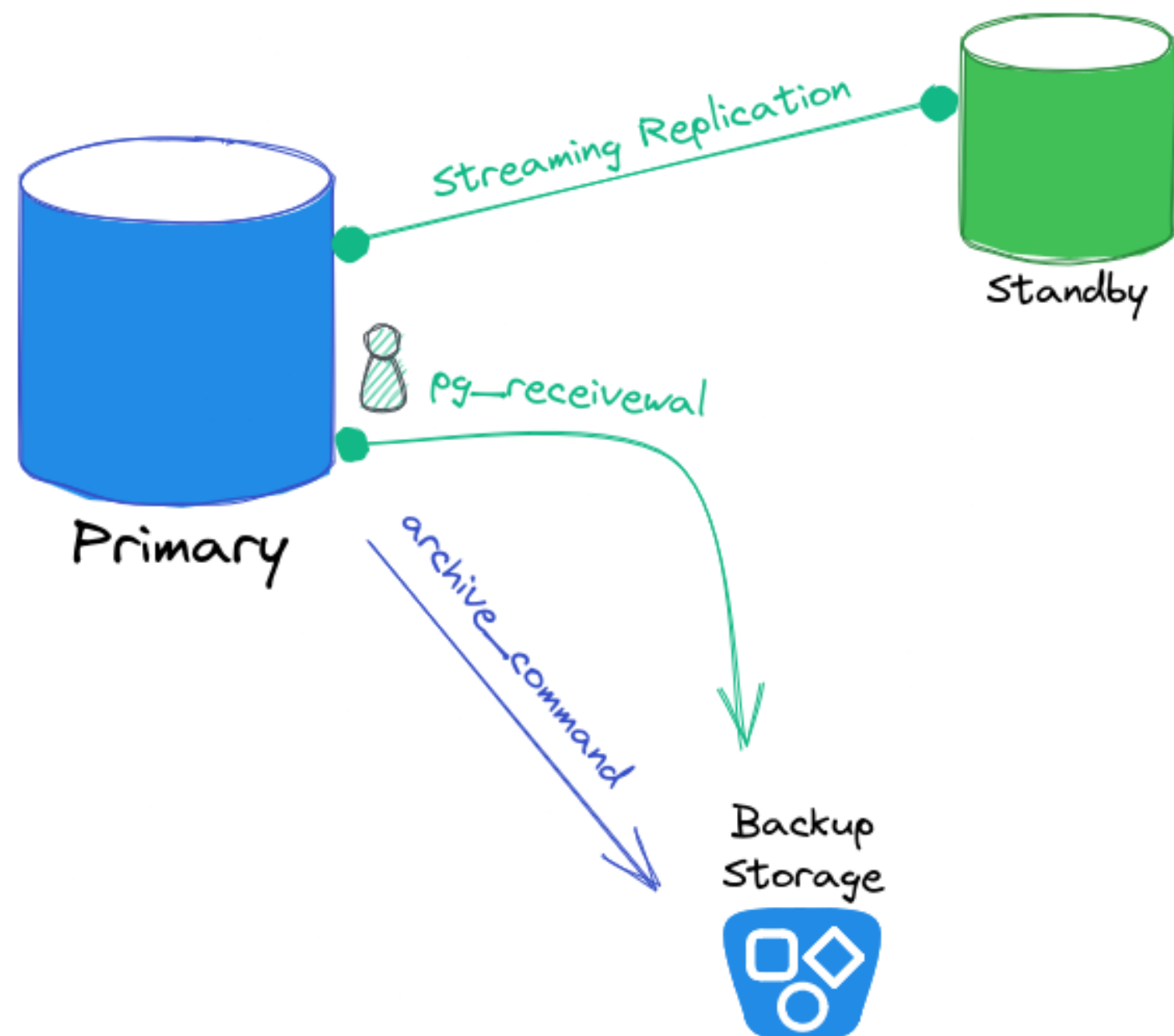
WAL archiving process



`.partial` WAL file

- usually < 16MB
- `pg_receivewal` = standby without data
- `.partial` is pushed by the standby server at promote time

Example (1)



Example (2)

```
archive_command = 'cp %p /shared/archives/%f'
```

```
pg_receivewal -D /shared/receivewal -v
```

```
/usr/pgsql-17/bin/pgbench -i -s 65
```

Example (3)

```
$ ps -o pid,cmd fx
  PID  CMD
27133  /usr/pgsql-17/bin/postgres -D /var/lib/pgsql/17/data/
...
27867  \_ postgres: walsender ... streaming 0/34EA7C80
27881  \_ postgres: walsender ... streaming 0/34EA7C80
```

```
$ ls /shared/archives
...
00000001000000000000000033
```

```
$ ls /shared/receivewal
...
00000001000000000000000033
00000001000000000000000034.partial
```

Example (4)

```
$ psql -c "SELECT pg_promote();" 
```

```
$ ls /shared/archives/  
...  
0000000100000000000000034.partial  
00000002.history
```

`pg_receivewal` still points to the old primary!

Timelines

A correct restore from backup, PITR or not,...
...always involves a timeline switch!

- archive recovery complete or standby promoted -> new timeline
 - part of WAL segment file names
 - to identify the series of WAL records generated after that recover
 - `.history` files

Faster archiving?

- async archiving
- compression types
- compression level

Async archiving

- using `archive-async=y`
 - temporary data (acknowledgments) stored into the `spool-path`
 - early archiving using `process-max` processes

Compression types

- file compression types supported (`compress-type`):
 - `bz2` - bzip2
 - `gz` - gzip (default)
 - `lz4`
 - `zst` - Zstandard

Compression level

- when `compress-level` is not specified
 - defaults levels based on `compress-type`
 - bz2 - 9
 - gz - 6
 - lz4 - 1
 - zst - 3

Example (1) - initial state

```
SELECT application_name, sent_lsn, write_lsn, flush_lsn FROM pg_stat_replication;
```

application_name	sent_lsn	write_lsn	flush_lsn
walreceiver	0/35000060	0/35000060	0/35000060
pg_receivewal	0/35000060	0/35000060	0/35000000

(2 rows)

```
SELECT last_archived_wal FROM pg_stat_archiver;
```

last_archived_wal
0000000100000000000000034

(1 row)

Example (2) - archiving too slow

```
/usr/pgsql-17/bin/pgbench -n -P 1 -T 60 -j 2 -c 50
```

```
SELECT application_name, sent_lsn, write_lsn, flush_lsn FROM pg_stat_replication;
 application_name | sent_lsn | write_lsn | flush_lsn
-----+-----+-----+-----
 walreceiver      | 0/BEF5C8 | 0/BEF5C8 | 0/BEF5C8
 pg_receivewal    | 0/BEF5C8 | 0/B6F04138 | 0/B6000000
(2 rows)
```

```
SELECT last_archived_wal FROM pg_stat_archiver;
 last_archived_wal
-----
 0000000100000000000000087
(1 row)
```

Example (3) - final state

```
SELECT application_name, sent_lsn, write_lsn, flush_lsn FROM pg_stat_replication;
```

application_name	sent_lsn	write_lsn	flush_lsn
walreceiver	0/C5EDB5D8	0/C5EDB5D8	0/C5EDB5D8
pg_receivewal	0/C5EDB5D8	0/C5EDB5D8	0/C5000000

(2 rows)

```
SELECT last_archived_wal FROM pg_stat_archiver;
```

last_archived_wal
000000010000000000000000C4

(1 row)

Example (4) - improvements

```
process-max=2
archive-async=y
compress-type=zst
```

```
/usr/pgsql-17/bin/pgbench -n -P 1 -T 60 -j 2 -c 50
```

```
SELECT application_name, sent_lsn, write_lsn, flush_lsn FROM pg_stat_replication;
 application_name | sent_lsn | write_lsn | flush_lsn
-----+-----+-----+-----
 walreceiver      | 1/270B8AA8 | 1/270B87E8 | 1/270B6670
 pg_receivewal    | 1/270440B0 | 1/21C15D60 | 1/21000000
(2 rows)
```

```
SELECT last_archived_wal FROM pg_stat_archiver;
 last_archived_wal
-----
 000000010000000100000025
(1 row)
```

What can go wrong with archiving?

Things can get worst... and they will! (Laetitia Avrot)

Archiving fails...

```
ERROR: [082]: WAL segment ... was not archived before the 60000ms timeout  
HINT: check the archive_command to ensure that all options are correct  
HINT: check the PostgreSQL server log for errors
```

`archive-push` console output goes into the PostgreSQL logs!

Error example

```
ERROR: [103]: unable to find a valid repository:
repo1: [FileNotFoundError] unable to load info file '.../archive/demo/archive.info'
or '.../archive/demo/archive.info.copy':
FileNotFoundError: unable to open missing file '.../archive/demo/archive.info' for read
FileNotFoundError: unable to open missing file '.../archive/demo/archive.info.copy' for read
HINT: archive.info cannot be opened but is required to push/get WAL segments.
HINT: is archive_command configured correctly in postgresql.conf?
HINT: has a stanza-create been performed?
```

WAL segments piling up...

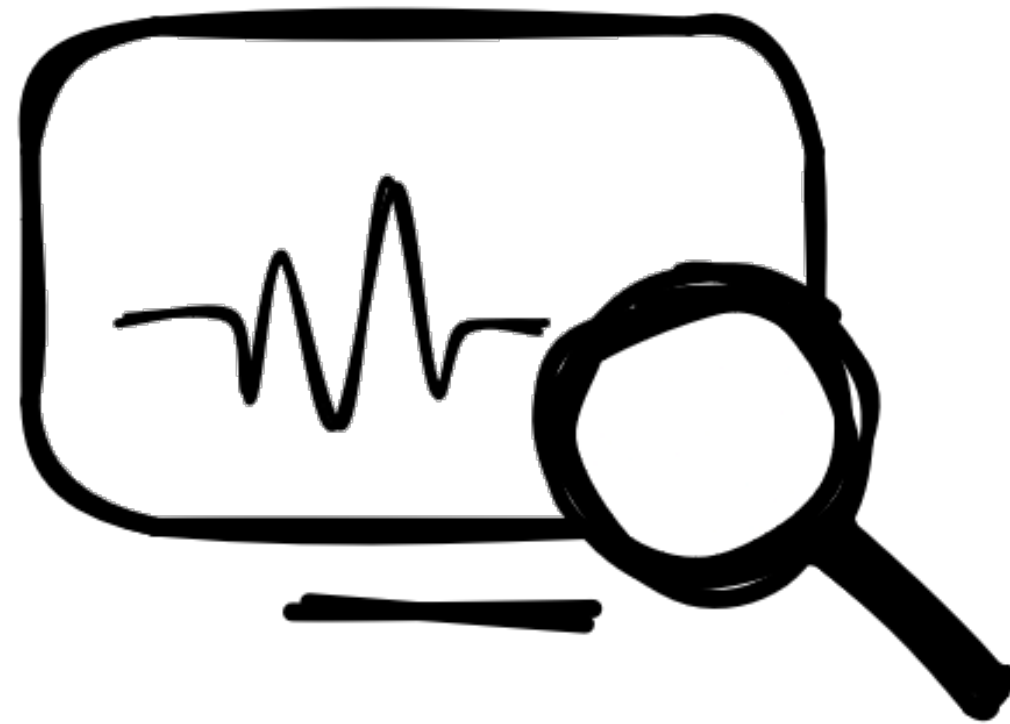
an error prevent PostgreSQL to remove/recycle the WAL file!

```
$ ls data/pg_wal/archive_status |grep .ready
...
00000001000000020000001B.ready
00000001000000020000001C.ready
00000001000000020000001D.ready
00000001000000020000001E.ready
```

Archiving queue

- `archive-push-queue-max`
 - maximum size of the PostgreSQL archive queue
 - prevent the WAL space from filling up until PostgreSQL stops completely...
 - ...but generate **missing archives!**
- very important to monitor archiving to ensure it continues working

Monitoring



What & How?

Logs

- `log-level-console` (stdout): **warn** by default
- `log-level-file` : **info** by default
- usually interesting to set `log-level-console` to **info** and `log-level-file` to **debug**

Archive-push

```
[demo:archive-push]  
log-level-console=debug
```

info command - json output

```
$ pgbackrest info --stanza=demo --output=json
{"archive":[{"database":{"id":1,"repo-key":1},
{"id":"17-1",
"max":"0000000100000002000000082",
"min":"0000000100000001000000039"}],
"backup":[{"archive":{"start":"0000000100000002000000072","stop":"0000000100000002000000082"}},
...
}
```

Backups

- retention
 - how old is your latest backup?
 - how old is your latest **full** backup?
 - how old is your **oldest** full backup?
 - how many full, diff or incr backups are in the repository?
 - does it meet the retention settings?

Backups - example

```
Long message      : full=1
Long message      : diff=0
Long message      : incr=3
Long message      : latest_bck=20240514-140212F_20240515-083055I
Long message      : latest_bck_type=incr
Long message      : latest_bck_age=3h28m33s
Long message      : latest_full=20240514-140212F
Long message      : latest_full_age=21h57m10s
Long message      : oldest_bck=20240514-140212F
Long message      : oldest_bck_age=21h57m10s
```

example generated with `check_pgbackrest`

Archives

- backup command checks archives needed for the consistency

```
$ pgbackrest --stanza=demo backup --type=full
...
INFO: check archive for segment(s) 000000010000000000000000DC:000000010000000000000000DC
...
```

- info command shows oldest and latest WAL archive in the repository

```
$ pgbackrest --stanza=demo info
...
wal archive min/max (17): 00000001000000000000000099/000000010000000000000000DC
```

Archives - questions

- what if archives needed for the backups consistency get removed after the backup?
- are all the archives between oldest and latest (from info) present in the repository?

```
Long message      : latest_archive_age=2m51s
Long message      : latest_archive=0000000100000000000000E7
Long message      : latest_bck_archive_start=0000000100000000000000DC
Long message      : latest_bck=20240515-122650F
Long message      : oldest_archive=000000010000000000000099
Long message      : oldest_bck_archive_start=000000010000000000000099
```

Archives and timeline switch

- WAL archives on different timelines found...

```
$ pgbackrest --stanza=demo info
...
wal archive min/max (17): 00000001000000000000000099/000000020000000000000000E8
```

- parse `.history` file, find switch point and define boundary WAL

```
$ pgbackrest repo-get --stanza=demo archive/demo/17-1/00000002.history
1 0/E856A0E8 no recovery target specified
```

```
SELECT pg_walfile_name('0/E856A0E8');
       pg_walfile_name
-----
000000010000000000000000E8
```

Restore

pgBackRest handles the restore,
PostgreSQL handles the recovery !

Let's talk about `restore` command and recovery targets...

Restore type?

- `--type` (and `--target` to reach)
 - default - to the end of the archive stream
 - immediate - to backup consistency point
 - lsn - to LSN (Log Sequence Number), `recovery_target_lsn`
 - name - to restore point, `recovery_target_name`
 - xid - to transaction id, `recovery_target_xid`
 - time - to a specific timestamp, `recovery_target_time`
 - ...

Backup set

- `--set`
 - default: latest
 - auto-select for **time** and **lsn** target

Timeline

- `--target-timeline`
 - `recovery_target_timeline`
 - default: `latest` (v12+) or `current` (< v12)

Selective restore

- `--db-include`
 - databases not specifically included will be restored as sparse, zeroed files
 - built-in databases (template0, template1, and postgres) are always restored unless specifically excluded
- `--db-exclude`
 - databases excluded will be restored as sparse, zeroed files
 - with the `--db-include` option, only apply to built-in databases

Asynchronously get WAL segments

- `archive-get` using `archive-async=y`
 - early fetching `archive-get-queue-max` amount of WAL segments to speed up recovery
 - using `process-max` processes
 - stored in the `spool-path`

Summary

- tweak your archiving system (async, compression type,...)
- monitor your backups and archives
- regularly try to restore your backups

Schrödinger's Law of Backups

*The condition/state of any backup is unknown until
a restore is attempted.*

Final Thoughts

- pgBackRest is a powerful and flexible tool
 - with a wide range of features
 - to boost your PostgreSQL backup and recovery strategy
- finding the best setup for your specific use case can be challenging
 - we're here to help!

Thank You!



contact@dataegret.com

