PostgreSQL Recovery Targets

PGSession 15

Stefan FERCOT

Wed Feb 15th, 2023

Who Am I?

- Stefan Fercot
- Database Backup Architect @EDB
- pgBackRest contributor
- aka. pgstef
- https://pgstef.github.io

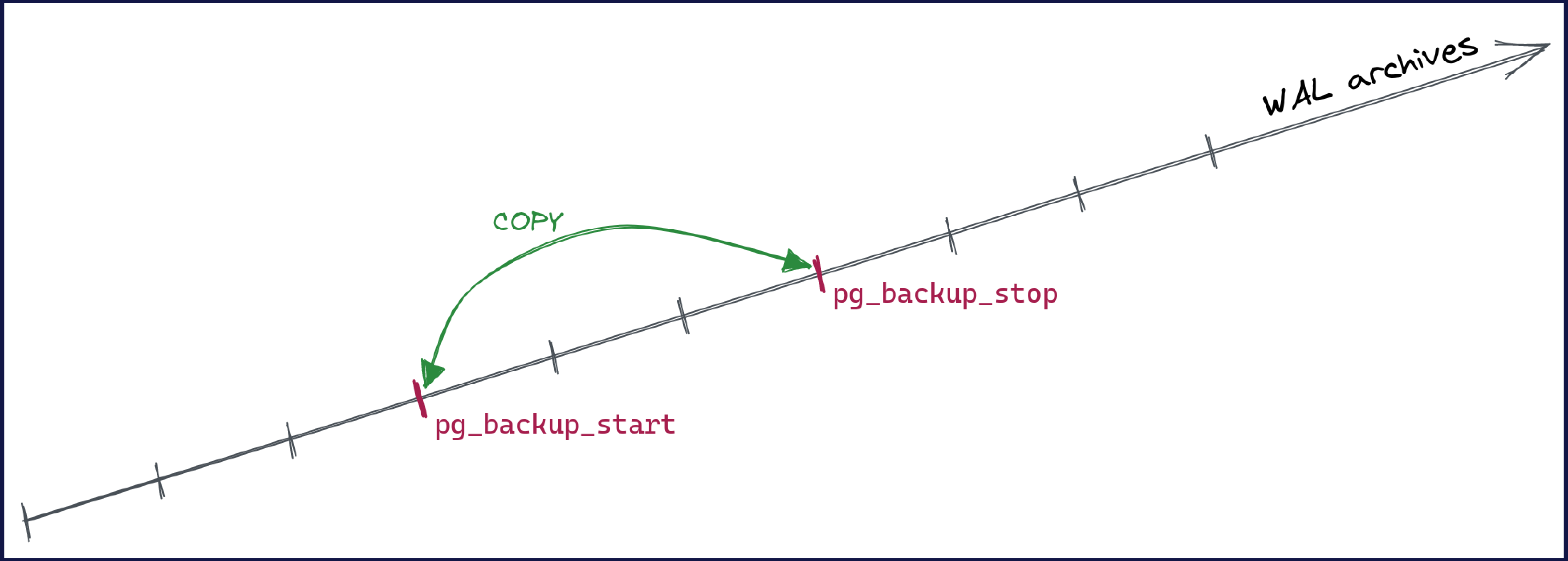
Agenda

- restore vs recovery
- recovery targets
  - what happens when the target is reached?
- how to find an accurate target
  - using pg_waldump
- look at recent PostgreSQL release notes

Restore vs Recovery

- *restore* process handled by community tools…
- *recovery* done by PostgreSQL itself!

Reminder



- file-system-level backup (data files)
- continuous WAL archiving (data modifications)

Backup consistency

- to recover successfully
  - continuous sequence of archived WAL files needed…
  - from backup start to backup stop location

What’s your recovery target?

- by default, recover to the end of the WAL stream
- how to specify an earlier stopping point?

Consistent state

- `recovery_target = 'immediate'`
  - recovery stops when consistent state is reached
  - (i.e. the point where taking the backup ended)

Restore point

- `recovery_target_name`
  - create a named restore point with `pg_create_restore_point()`

Timestamp

- `recovery_target_time`
  - timestamp with time zone format
  - recommended to use a numeric offset from UTC
    - example: `2022-12-15 13:55:18.567762+00`
  - or write a full time zone name, e.g., *Europe/Brussels* not *CET*

Transaction ID

- `recovery_target_xid`
  - transactions committed before (and optionally including) specified xid will be recovered

WAL location

- `recovery_target_lsn`
  - LSN of the write-ahead log location
  - parameter parsed as system data type pg_lsn

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/D3000148
(1 row)

=# SELECT pg_walfile_name(pg_current_wal_lsn());
     pg_walfile_name
--------------------------
 0000000100000002000000D3
(1 row)
```

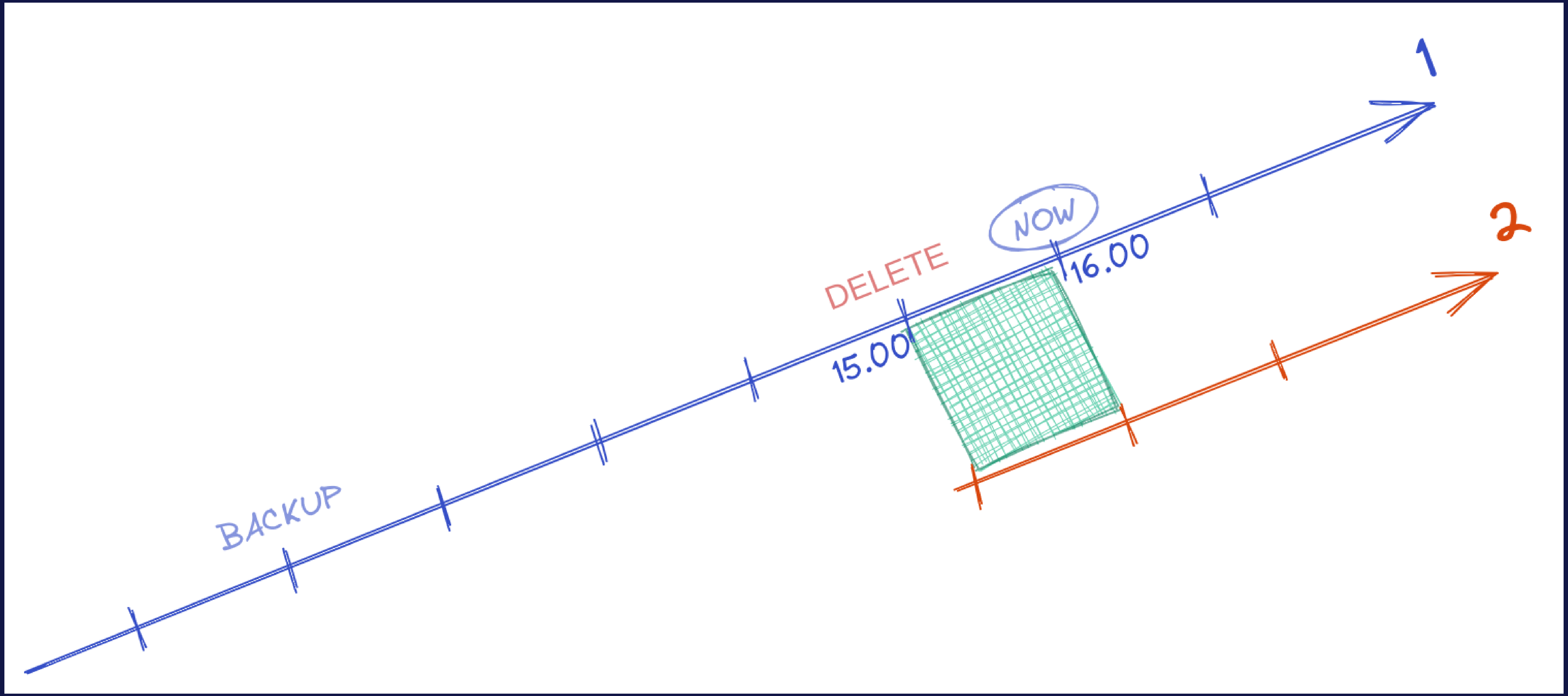
WAL filename

- 000000010000000200000003
  - 00000001 : timeline
  - 00000002 : wal
  - 00000003 : segment
- hexadecimal
  - 000000010000000000000001
  - 0000000100000000000000FF
  - 000000010000000100000000
  - …

Timeline to follow

- 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)

Timeline (2)



What happens when the target is reached?

- include the target?
- finally, some action!

Stop after or before the target

- `recovery_target_inclusive`
  - recovery stops just after recovery target (`on`)…
  - …or just before (`off`)
  - works with LSN, time or xid
  - default is `on`

Target action

- `recovery_target_action`
  - pause (`pg_wal_replay_resume()`)
  - promote
  - shutdown

Target hard to find?

`pg_waldump` is your friend…

Quick demo setup

```
$ createdb pgbench
$ /usr/pgsql-15/bin/pgbench -i -s 600 pgbench
$ /usr/pgsql-15/bin/pgbench -c 4 -j 2 -T 300 pgbench
```

```
archive_mode = on
archive_command = 'test ! -f /backup_space/archives/%f && cp %p /backup_space/archives/%f'
```

Oops time...

```
SELECT pg_create_restore_point('RP1');
BEGIN;
    SELECT pg_current_wal_lsn(), current_timestamp;
    DELETE FROM pgbench_tellers;
COMMIT;
SELECT pg_switch_wal();
```

Useful information from the output

```
pgbench=*# SELECT pg_current_wal_lsn(), current_timestamp;
 pg_current_wal_lsn |        current_timestamp
--------------------+-------------------------------
 2/6987D9E8         | 2023-02-08 14:29:05.703187+00
(1 row)
```

pg_waldump

```
$ /usr/pgsql-15/bin/pg_waldump /backup_space/archives/000000010000000200000069
rmgr: XLOG        len (rec/tot):     98/    98, tx:          0,
    lsn: 2/6987D948, prev 2/6987D910, desc: RESTORE_POINT RP1
...
rmgr: Heap        len (rec/tot):     54/    54, tx:     176701, lsn: 2/6987D9E8,
    prev 2/6987D9B0, desc: DELETE off 2 flags 0x00 KEYS_UPDATED ,
    blkref #0: rel 1663/16388/16404 blk 0
...
rmgr: Transaction len (rec/tot):     34/    34, tx:     176701,
    lsn: 2/698CFE40, prev 2/698CFE08, desc: COMMIT 2023-02-08 14:29:05.708534 UTC
```

EDB™

How to identify our relation?

```
pgbench=# SELECT dattablespace AS tablespace, oid AS database,
          pg_relation_filenode('pgbench_tellers') AS table
          FROM pg_database
          WHERE datname=current_database();

 tablespace | database | table
------------+----------+-------
       1663 |    16388 | 16404
(1 row)
```

Example (1)

```
$ pg_waldump --rmgr=XLOG 000000010000000000000001 000000010000000200000069
...
rmgr: XLOG        len (rec/tot):     98/    98, tx:          0,
    lsn: 2/6987D948, prev 2/6987D910, desc: RESTORE_POINT RP1
rmgr: XLOG        len (rec/tot):     24/    24, tx:          0,
    lsn: 2/698CFEA0, prev 2/698CFE68, desc: SWITCH
```

Example (2)

```
$ pg_waldump --relation=1663/16388/16404 000000010000000000000001 000000010000000200000069
...
rmgr: Heap        len (rec/tot):     54/    54, tx:     176701,
    lsn: 2/698CFD60, prev 2/698CFD28,
    desc: DELETE off 190 flags 0x00 KEYS_UPDATED , blkref #0: rel 1663/16388/16404 blk 56
rmgr: Heap        len (rec/tot):     54/    54, tx:     176701,
    lsn: 2/698CFD98, prev 2/698CFD60,
    desc: DELETE off 198 flags 0x00 KEYS_UPDATED , blkref #0: rel 1663/16388/16404 blk 56
rmgr: Heap        len (rec/tot):     54/    54, tx:     176701,
    lsn: 2/698CFDD0, prev 2/698CFD98,
    desc: DELETE off 205 flags 0x00 KEYS_UPDATED , blkref #0: rel 1663/16388/16404 blk 56
rmgr: Heap        len (rec/tot):     54/    54, tx:     176701,
    lsn: 2/698CFE08, prev 2/698CFDD0,
    desc: DELETE off 212 flags 0x00 KEYS_UPDATED , blkref #0: rel 1663/16388/16404 blk 56
```

Example (3)

```
$ pg_waldump --xid=176701 --rmgr=Transaction 000000010000000200000069
rmgr: Transaction len (rec/tot):     34/    34, tx:     176701,
    lsn: 2/698CFE40, prev 2/698CFE08,
    desc: COMMIT 2023-02-08 14:29:05.708534 UTC
```

Findings...

- name: `RP1`
- lsn: `lsn: 2/6987D9B0` (lsn before the first DELETE)
- xid: `tx: 176701`
- time: `2023-02-08 14:29:05.703187+00`
  - or `COMMIT 2023-02-08 14:29:05.708534 UTC`

What changed lately?

Quick look inside the PostgreSQL releases notes

v12

- move `recovery.conf` settings into `postgresql.conf`
  - `recovery.conf` replaced by `recovery.signal` and `standby.signal`
  - the `standby_mode` setting has been removed

v13

- generate an error if recovery does not reach the specified recovery target
- previously, promote happened upon reaching the end of WAL…
  - even if the target was not reached!

v15

- remove long-deprecated exclusive backup mode
  - `pg_backup_start()` / `pg_backup_stop()` renamed
  - `pg_is_in_backup()` removed

FAQ

Frequently asked questions…

FAQ (1)

I put the backup start time as the recovery target and it didn't work

FAQ (2)

How many backups do I need to take per day?

Conclusion

- tools are helpful
- restore points are easy to use
- as usual, practice is the key to success

Questions?

Thank you for your attention!