PostgreSQL et ses cibles de restauration

PGSession 15

Stefan FERCOT

mer 15 fév 2023

Qui suis-je ?

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

EDB™

Agenda

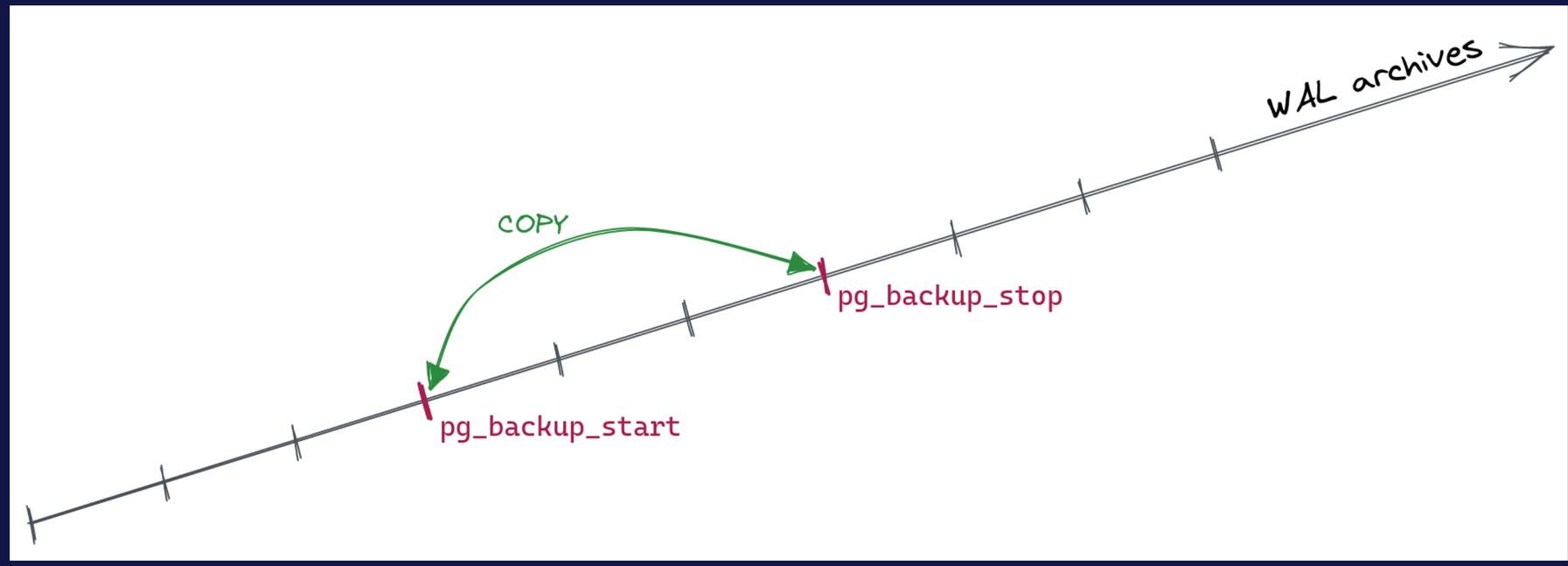
- *restore* vs *recovery*
- cibles de restauration (*recovery targets*)
  - que se passe-t-il lorsque la cible est atteinte ?
- comment trouver une cible précise ?
  - en utilisant *pg_waldump*
- un regard sur les dernières versions de PostgreSQL

EDB™

Restore vs Recovery

- *restore* : processus géré par les outils…
- *recovery* appliqué par PostgreSQL !

Rappel



- copie/sauvegarde des fichiers de données (*pg_data*)
- archivage continu des journaux de transactions WAL

Sauvegarde cohérente

- une restauration correcte nécessite
  - une séquence continue de WAL archivés…
  - du début à la fin de la sauvegarde

Et les cibles de restauration dans tout ça ?

- par défaut, rejeu de toutes les archives WAL existantes
- comment spécifier un point d’arrêt ?

État de cohérence

- `recovery_target = 'immediate'`
  - le rejeu s’arrête lorsque l’état de cohérence des données est atteint
  - (au moment de la fin de la sauvegarde)

Point de restauration

- `recovery_target_name`
  - créer un point de restauration nommé avec `pg_create_restore_point()`

Timestamp

- `recovery_target_time`
  - timestamp with time zone format
  - il est recommandé d'utiliser un décalage numérique à partir de *UTC*
    - exemple: `2022-12-15 13:55:18.567762+00`
  - ou écrire le nom de la zone temporelle en entier : *Europe/Brussels* plutôt que *CET*

EDB™

Identifiant de transaction

- `recovery_target_xid`
  - les transactions commitées avant (ou optionnellement incluant) cet identifiant seront rejouées

Position dans les journaux de transactions

- `recovery_target_lsn`
  - LSN ou position exacte de l'enregistrement dans les journaux de transactions
  - paramètre de type *pg_lsn*

LSN

- *log sequence number*
  - position de l’enregistrement dans les journaux de transactions
  - garant d’unicité

```
=# SELECT pg_current_wal_lsn();
 pg_current_wal_lsn
--------------------
 2/D3000148
(1 row)

=# SELECT pg_walfile_name(pg_current_wal_lsn());
     pg_walfile_name
--------------------------
 000000010000000200000D3
(1 row)
```

EDB™

Nom des fichiers WAL (segments)

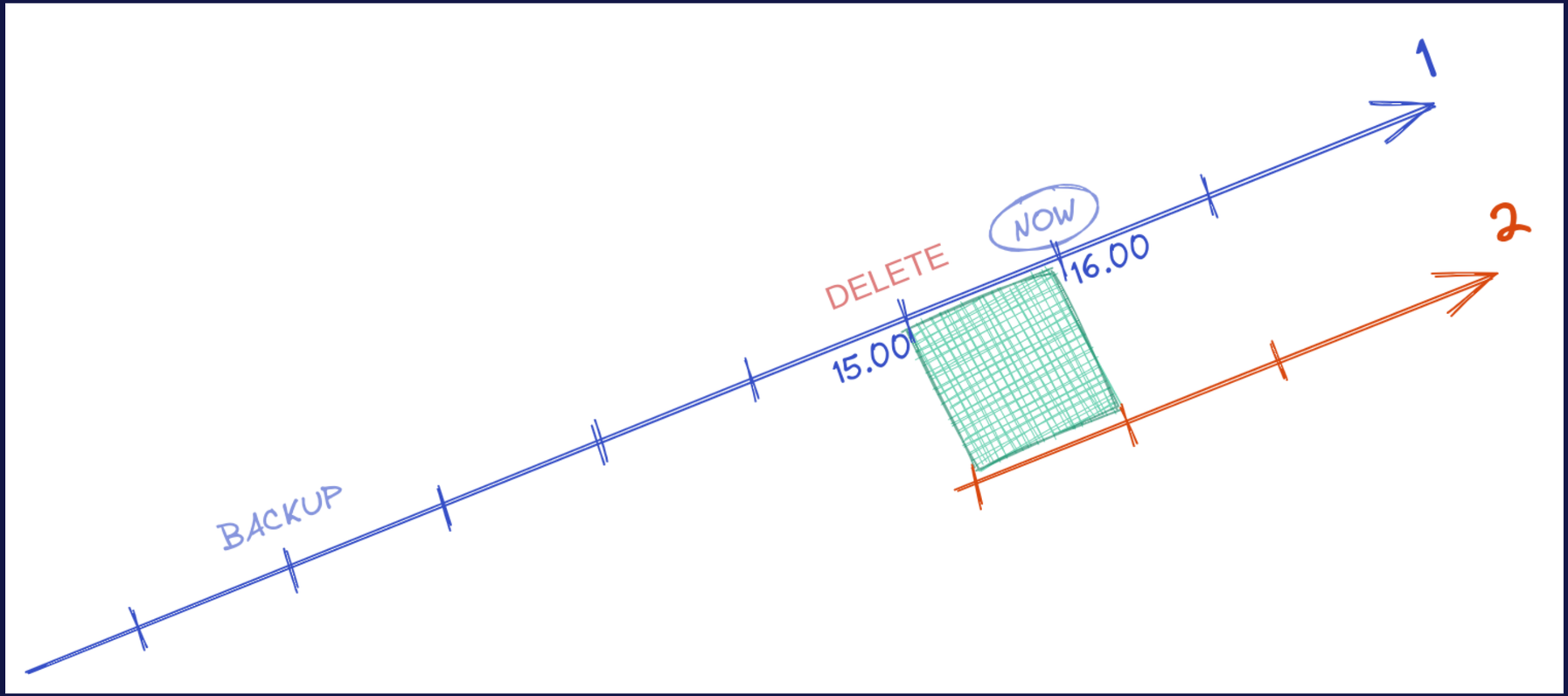
- 000000010000000200000003
  - 00000001 : timeline
  - 00000002 : wal
  - 00000003 : segment
- hexadecimal
  - 000000010000000000000001
  - 0000000100000000000000FF
  - 000000010000000100000000
  - …

Ligne du temps (*timeline*) à suivre

- restauration réussie -> nouvelle ligne du temps
  - reprise dans le nom des fichiers segments WAL
  - pour identifier la série d'enregistrements WAL générés après cette restauration
  - fichiers `.history`
- `recovery_target_timeline`
  - par défaut : `latest` (v12 +) ou `current` (< v12)

EDB™

Ligne du temps (2)



Et quand la cible est atteinte ?

- inclure la cible ?
- enfin un peu d’action !

EDB™

S’arrêter avant ou après la cible

- `recovery_target_inclusive`
  - le rejeu s’arrête juste après (`on`)…
  - …ou juste avant (`off`) la cible de restauration
  - fonctionne avec *LSN*, *time* ou *xid*
  - `on` par défaut

EDB™

Target action

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

EDB™

Difficile de trouver une cible précise ?

`pg_waldump` est votre ami…

EDB™

Démo rapide

```
$ 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...

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

EDB™

Information utile

```
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)
```

EDB™

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™

Comment y retrouver notre table ?

```
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)
```

EDB™

Exemple (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
```

EDB™

Exemple (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
```

EDB™

Exemple (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
```

EDB™

Résumer de nos découvertes…

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

Des changements récents ?

Rapide coup d’œil aux notes de version PostgreSQL récentes

EDB™

v12

- déplacement des paramètres de `recovery.conf` vers `postgresql.conf`
  - `recovery.conf` remplacé par `recovery.signal` et `standby.signal`
  - paramètre `standby_mode` supprimé

EDB™

v13

- génère une erreur si le rejeu (*recovery*) n’atteint pas la cible spécifiée
- précédemment, la promotion avait lieu une fois la fin des journaux de transactions atteinte…
  - même si la cible de restauration demandée n’était pas trouvée

EDB™

v15

- suppression du mode de sauvegarde exclusif (déprécié depuis longtemps)
  - `pg_backup_start()` / `pg_backup_stop()` renommées
  - `pg_is_in_backup()` supprimée

EDB™

FAQ

Frequently asked questions…

EDB™

FAQ (1)

J’ai spécifié l’heure de début de ma sauvegarde comme cible de restauration et ça ne fonctionne pas

FAQ (2)

Combien de sauvegardes par jour dois-je prévoir ?

Conclusion

- les outils sont très utiles (mais pas magiques)
- les points de restauration nommés sont faciles à utiliser
- comme d’habitude, la pratique est la clé du succès

EDB™

Questions ?

Merci pour votre attention !