PostgreSQL Recovery Targets

EDB™

pgDay Paris 2023

Stefan FERCOT

March 23th, 2023

© Copyright EntrepriseDB Corporation, 2023. All rights reserved.

Who Am I?

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

EDB™

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

EDB™

Restore vs Recovery

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

EDB™

Reminder

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

EDB™

Backup consistency

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

EDB™

What’s your recovery target?

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

EDB™

Consistent state

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

EDB™

Restore point

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

EDB™

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*

EDB™

Transaction ID

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

EDB™

WAL location

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

EDB™

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
--------------------------
 000000010000000200000D3
(1 row)
```

EDB™

WAL filename

- 000000010000000200000003
  - 00000001 : timeline
  - 00000002 : wal
  - 00000003 : segment
- hexadecimal
  - 000000010000000000000001
  - 000000010000000000000FF
  - 00000001000000010000000
  - …

EDB

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)

EDB

Timeline (2)

EDB™

What happens when the target is reached?

- include the target?
- finally, some action!

EDB™

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`

EDB™

Target action

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

EDB™

Target hard to find?

`pg_waldump` is your friend…

EDB™

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'
```

EDB™

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();
```

EDB™

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)
```

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™

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)
```

EDB™

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
```

EDB™

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
```

EDB™

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
```

EDB™

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`

EDB™

Sample recovery outputs

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

EDB™

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™

Don’t forget to practice!

Schrödinger’s Law of Backups

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

EDB™

Quick demo setup

- take a file-system-level copy
  - while `pgbench` is running
- restore the copy
- configure the recovery

EDB™

pg_basebackup

```
$ pg_basebackup -D "/backup_space/backups/$(date +'%F_%T')" \
 --format=plain --wal-method=none --checkpoint=fast --progress
```

EDB™

Configure the recovery

```
$ touch /var/lib/pgsql/15/data/recovery.signal
```

```
# postgresql(.auto).conf
restore_command = 'cp /backup_space/archives/%f %p'
recovery_target = 'immediate'
recovery_target_action = 'promote'
```

```
$ cat /var/lib/pgsql/15/data/backup_label
START WAL LOCATION: 2/5E191470 (file 000000010000000020000005E)
...
```

EDB™

Look at the logs

```
LOG:  starting point-in-time recovery to earliest consistent point
LOG:  restored log file "000000010000000020000005E" from archive
LOG:  redo starts at 2/5E191470
LOG:  restored log file "..." from archive
...
LOG:  consistent recovery state reached at 2/644EC6C8
LOG:  recovery stopping after reaching consistency
```

EDB™

What if 000000010000000200000062 is missing?

```
LOG:  missing contrecord at 2/61FFE470
LOG:  redo done at 2/61FFC420 ...
FATAL:  recovery ended before configured recovery target was reached
```

EDB™

What changed lately?

Quick look inside the PostgreSQL releases notes

EDB™

v12

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

EDB™

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!

EDB™

v15

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

EDB™

pg_walinspect

```
postgres=# SELECT pg_current_wal_lsn(),now();
 pg_current_wal_lsn |              now
--------------------+-------------------------------
 2/830299F0         | 2023-03-15 07:49:20.529526+00
```

```
postgres=# SELECT start_lsn, end_lsn, xid, record_type, description
           FROM pg_get_wal_records_info_till_end_of_wal('2/830299F0')
           WHERE resource_manager='Transaction' AND record_type='COMMIT'
           ORDER BY start_lsn DESC LIMIT 1;
-[ RECORD 1 ]-------------------------------------------------------
start_lsn   | 2/AFEFA448
end_lsn     | 2/AFEFA4C0
xid         | 96131
record_type | COMMIT
description | 2023-03-15 07:56:17.886734+00; inval msgs: catcache 63...
```

EDB™

FAQ

Frequently asked questions…

EDB™

FAQ (1)

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

EDB™

FAQ (2)

How many backups do I need to take per day?

EDB™

Conclusion

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

EDB™

Next stop?

PG Day France

https://pgday.fr/appel

- french speaking event
- 19-20 June, Strasbourg

EDB™

Questions?

Thank you for your attention!

EDB™