Failover and Switchover Deep Dive with Manual Resolution

2025 David Pech

About Me

David Pech

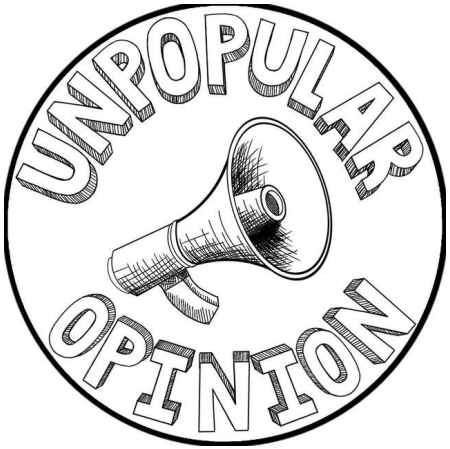
Java

kubestronaut

ORACLE
Cloud Infrastructure

aws

Is Postgres a Distributed System?



- not quite
- "import / export"
- leader (can write) / replica (read-only)
- how to point clients to a leader?

But

- easy to resync data between replicas? (no)
- can we serve only part of dataset? (not with physical)
- multiple leaders? (sure)
- always connect to leader only? (sure - only with libpq)

Postgres as a Product

Linux Kernel

–

SystemD

–-

Postgres (== core)

–

Patroni, RepMgr, PG Kubernetes operator

–

Do-It-Yourself, Helm chart, "distributions" …

–

Web UI, CDC…

Linux Kernel

–

SystemD

–

Apache Kafka *(without ZooKeeper now)*

–

Confluent Platform, Kafka-connect, …

Physical Replication

- Each DB cluster has unique "Database system identifier"
- Different Instances == Clones of the same Cluster
- Replica needs to process exactly the same change data (WAL) as Leader
  - Any difference => not a clone anymore, not a correctly working replica
  - (You can't lose or change any WAL segment)
- Replication slot
  - Just a logical concept for a Leader to track all Replica WAL positions
  - Main objective = coordinate what to send to replicas + not to lose data before the replicas can fetch them

(vs. Logical replication = completely separate PG Clusters)

Log Sequence Number (LSN) & Timeline

- PG instance track its current "writing position" in bytes
  - (== pointer to Write Ahead Log (WAL) position)
  - LSN can be used to calculate "drift" in bytes

- Timeline = each Point-In-Time-Recovery (PITR) event a new timeline is create (+1)
  - .history file is generated to track "branching"

Log Sequence Number (LSN) & Timeline Example

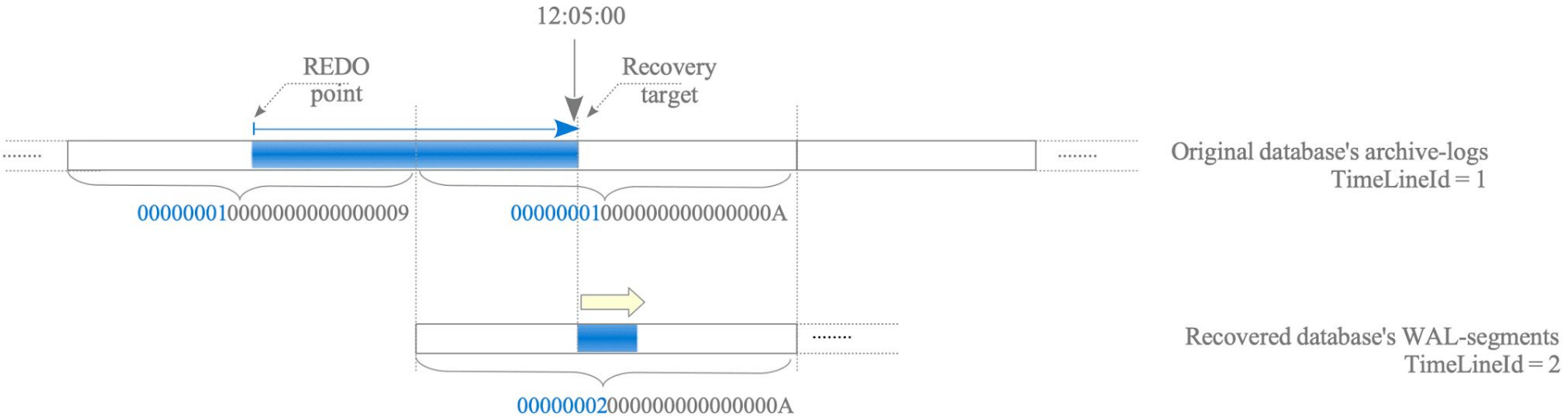


Image: https://www.interdb.jp/pg/pgsql10/03.html

Log Sequence Number (LSN) & Timeline DEMO

```
root=# SELECT pg_current_wal_lsn(), pg_current_wal_insert_lsn();
-[ RECORD 1 ]-------------+-----------
pg_current_wal_lsn        | 0/197C98B8
pg_current_wal_insert_lsn | 0/197C9A20

root=# SELECT pg_current_wal_lsn(), pg_current_wal_insert_lsn();
-[ RECORD 1 ]-------------+-----------
pg_current_wal_lsn        | 0/1996CF98
pg_current_wal_insert_lsn | 0/1996D128

root=# SELECT pg_current_wal_lsn() - '0/0';
-[ RECORD 1 ]-------
?column? | 435596904

root=# SELECT '0/C6F54810'::pg_lsn - '0/BCD270D0'::pg_lsn;
-[ RECORD 1 ]-------
?column? | 170055488

root=# SELECT system_identifier FROM pg_control_system();
-[ RECORD 1 ]-----+--------------------
system_identifier | 7462225020764606494

root=# SELECT timeline_id FROM pg_control_checkpoint();
-[ RECORD 1 ]--
timeline_id | 1

root=# SELECT pg_walfile_name(pg_current_wal_lsn());
-[ RECORD 1 ]---+------------------------
pg_walfile_name | 000000010000000000000001C
```

RPO and RTO

RPO - measured primarily in bytes (replica lag) => LSN calculations

Image: https://www.rubrik.com/insights/rto-rpo-whats-the-difference

Following Examples

- pg-red - Leader
- pg-green - Replica
- pg-blue - Replica

pg_is_in_recovery() - on the replica side

```
root=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 row)

root=# SELECT * FROM pg_stat_wal_receiver ;
-[ RECORD 1 ]---------+------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
pid                   | 29
status                | streaming
receive_start_lsn     | 0/29000000
receive_start_tli     | 1
written_lsn           | 0/2DC0E7A8
flushed_lsn           | 0/2DC0E7A8
received_tli          | 1
last_msg_send_time    | 2025-01-21 04:58:54.180504+00
last_msg_receipt_time | 2025-01-21 04:58:54.180532+00
latest_end_lsn        | 0/2DC0E7A8
latest_end_time       | 2025-01-21 04:58:54.180504+00
slot_name             | blue
sender_host           | pg-red
sender_port           | 5432
conninfo              | user=root passfile=/root/.pgpass channel_binding=prefer dbname=replication host=pg-red port=5432 fallback_applicat
ion_name=walreceiver sslmode=prefer sslnegotiation=postgres sslcompression=0 sslcertmode=allow sslsni=1 ssl_min_protocol_version=TLSv1.2 g
ssencmode=prefer krbsrvname=postgres gssdelegation=0 target_session_attrs=any load_balance_hosts=disable
```

pg_promote()

- If already not in recovery => Error
- Does not require PG instance restart
  - removes standby.signal (typically also postgresql.auto.conf)

```
pg-blue# cat /var/lib/postgresql/data/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=root passfile=''/root/.pgpass'' channel_binding=prefer host=''pg-red'' port=5432 sslmode=prefer sslnegotiation=pd
stgres sslcompression=0 sslcertmode=allow sslsni=1 ssl_min_protocol_version=TLSv1.2 gssencmode=prefer krbsrvname=postgres gssdelegation=0
target_session_attrs=any load_balance_hosts=disable'
primary_slot_name = 'blue'
```

```
root=# select pg_promote();
2025-01-21 05:10:46.716 UTC [28] LOG:  received promote request
2025-01-21 05:10:46.717 UTC [29] FATAL:  terminating walreceiver process due to administrator command
2025-01-21 05:10:46.717 UTC [28] LOG:  invalid record length at 0/4B126028: expected at least 24, got 0
2025-01-21 05:10:46.717 UTC [28] LOG:  redo done at 0/4B125FE8 system usage: CPU: user: 14.70 s, system: 11.37 s, elapsed: 802.38 s
2025-01-21 05:10:46.717 UTC [28] LOG:  last completed transaction was at log time 2025-01-21 05:10:46.714676+00
2025-01-21 05:10:46.721 UTC [28] LOG:  selected new timeline ID: 2
2025-01-21 05:10:46.747 UTC [28] LOG:  archive recovery complete
2025-01-21 05:10:46.753 UTC [25] LOG:  database system is ready to accept connections
 pg_promote
------------
 t
(1 row)
```

pg_demote() ?

- No such function
- Leader needs to be restarted (NO other way)

```
pg-blue# su - postgres -c '/usr/local/bin/pg_ctl -D /var/lib/postgresql/data stop'
waiting for server to shut down....2025-01-21 05:22:37.819 UTC [25] LOG:  received fast shutdown request
2025-01-21 05:22:37.822 UTC [25] LOG:  aborting any active transactions
2025-01-21 05:22:37.824 UTC [25] LOG:  background worker "logical replication launcher" (PID 39) exited with exit code 1
2025-01-21 05:22:37.825 UTC [26] LOG:  shutting down
2025-01-21 05:22:37.825 UTC [26] LOG:  checkpoint starting: shutdown immediate
2025-01-21 05:22:37.830 UTC [26] LOG:  checkpoint complete: wrote 0 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.00
1 s, sync=0.001 s, total=0.006 s; sync files=0, longest=0.000 s, average=0.000 s; distance=0 kB, estimate=317826 kB; lsn=0/4B1261D8, redo
lsn=0/4B1261D8
2025-01-21 05:22:37.841 UTC [25] LOG:  database system is shut down
 done
server stopped
pg-blue# touch /var/lib/postgresql/data/standby.signal
pg-blue# echo >>/var/lib/postgresql/data/postgresql.auto.conf "primary_conninfo = '...CONN INFO...'"
pg-blue# echo >>/var/lib/postgresql/data/postgresql.auto.conf "primary_slot_name = 'green'"
```

Who is the Leader now?

- Single source of truth is needed to make decisions
- Simple example = monitoring VM
- Real examples
  - Patroni utilizes "Distributed Configuration Store" (DCS) - etcd, …
  - PG operators in Kubernetes - kube-apiserver (etcd behind)
  - 3 or 5 nodes are needed for a decision (or just 1)

2 purposes:

- PG cluster Leader election
- Client routing (not covered here)

Creating a PG cluster

- Procedure
  - initdb a first PG instance = Leader
  - Clone the cluster to bootstrap Replicas
  - Make Replicas follow the Leader
- Clone (== must copy the PG datadir/ in some state)
  - Typically = pg_basebackup = copy current datadir/ + WAL
  - Using infrastructure = disk clone + WAL
- Result
  - 1 PG instance as a Leader
  - 2 PG instances as Replicas

Creating a PG cluster - demo

- su - postgres -c '/usr/local/bin/initdb -D /var/lib/postgresql/data/ -k'

```
pg-red# 1-initdb.sh
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with this locale configuration:
  locale provider:   libc
  LC_COLLATE:  C
  LC_CTYPE:    C.UTF-8
  LC_MESSAGES: C
  LC_MONETARY: C
  LC_NUMERIC:  C
  LC_TIME:     C
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are enabled.

fixing permissions on existing directory /var/lib/postgresql/data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default "max_connections" ... 100
selecting default "shared_buffers" ... 128MB
selecting default time zone ... UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... sh: locale: not found
2025-01-21 05:27:22.140 UTC [26] WARNING:  no usable system locales were found
ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host, the next time you run in
itdb.

Success. You can now start the database server using:

    /usr/local/bin/pg_ctl -D /var/lib/postgresql/data/ -l logfile start
```

Creating a Replica - easy way

pg_basebackup -c fast -C -P -v --slot=blue -R -h pg-red -D /var/lib/postgresql/data

```
pg-blue# 1-pg_basebackup.sh
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/3000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created replication slot "blue"
53684/53684 kB (100%), 1/1 tablespace
pg_basebackup: write-ahead log end point: 0/3000120
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed
```

```
pg-blue# ls /var/lib/postgresql/data/
PG_VERSION          pg_commit_ts        pg_multixact        pg_stat             pg_wal
backup_label        pg_dynshmem         pg_notify           pg_stat_tmp         pg_xact
backup_manifest     pg_hba.conf         pg_replslot         pg_subtrans         postgresql.auto.conf
base                pg_ident.conf       pg_serial           pg_tblspc           postgresql.conf
global              pg_logical          pg_snapshots        pg_twophase         standby.signal
```

Creating a Replica - hard way

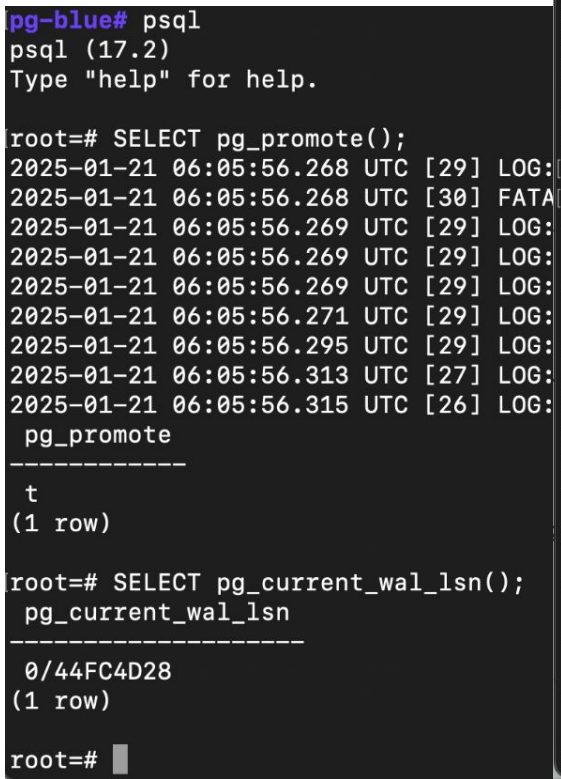
```
pg-green# psql -h pg-red
psql (17.2)
Type "help" for help.

root=# SELECT pg_create_physical_replication_slot('green');
 pg_create_physical_replication_slot
-------------------------------------
 (green,)
(1 row)

root=# SELECT pg_backup_start(label => 'green', fast => true);
 pg_backup_start
-----------------
 0/70000B0
(1 row)

root=# -- DO COPY HERE
root=# SELECT * FROM pg_backup_stop(wait_for_archive =>
    lsn     |                          labelfile
------------+---------------------------------------------------
 0/1B809550 | START WAL LOCATION: 0/70000B0 (file 000000
            | CHECKPOINT LOCATION: 0/7065740
            | BACKUP METHOD: streamed
            | BACKUP FROM: primary
            | START TIME: 2025-01-21 05:46:43 UTC
            | LABEL: green
            | START TIMELINE: 1
            |
(1 row)
```

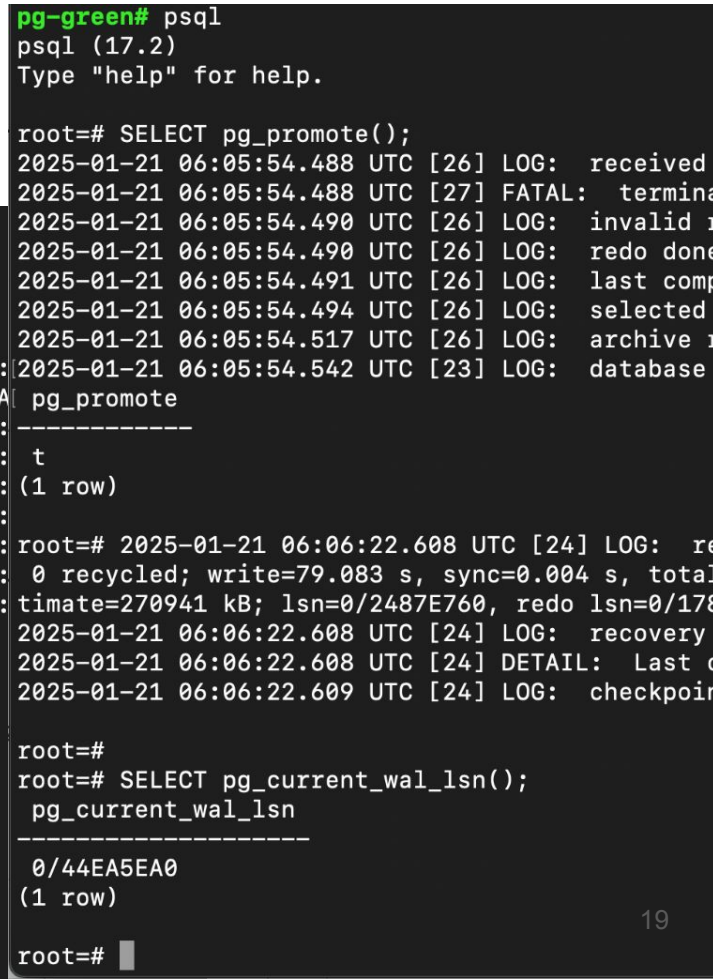
```
pg-green# mv /mnt/backup/* /var/lib/postgresql/data/
pg-green# touch /var/lib/postgresql/data/standby.signal
pg-green# echo >>/var/lib/postgresql/data/postgresql.auto.conf "primary_conninfo = 'user=root
 refer host=''pg-red'' port=5432 sslmode=prefer sslnegotiation=postgres sslcompression=0 sslce
n=TLSv1.2 gssencmode=prefer krbsrvname=postgres gssdelegation=0 target_session_attrs=any load
pg-green# echo >>/var/lib/postgresql/data/postgresql.auto.conf "primary_slot_name = 'green'"
pg-green# su - postgres -c '/usr/local/bin/pg_ctl -D /var/lib/postgresql/data start'
waiting for server to start....2025-01-21 05:51:58.207 UTC [47] LOG:  starting PostgreSQL 17.
gcc (Alpine 14.2.0) 14.2.0, 64-bit
2025-01-21 05:51:58.207 UTC [47] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2025-01-21 05:51:58.207 UTC [47] LOG:  listening on IPv6 address "::", port 5432
2025-01-21 05:51:58.209 UTC [47] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL
2025-01-21 05:51:58.211 UTC [50] LOG:  database system was shut down in recovery at 2025-01-2
2025-01-21 05:51:58.211 UTC [50] LOG:  entering standby mode
2025-01-21 05:51:58.213 UTC [50] LOG:  redo starts at 0/70000B0
.2025-01-21 05:51:59.714 UTC [50] LOG:  consistent recovery state reached at 0/15CA3B58
2025-01-21 05:51:59.715 UTC [47] LOG:  database system is ready to accept read-only connectio
2025-01-21 05:51:59.715 UTC [50] LOG:  invalid record length at 0/15CA3CE8: expected at least
2025-01-21 05:51:59.718 UTC [51] LOG:  started streaming WAL from primary at 0/15000000 on ti
 done
server started
```

Let's Promote Everyone…

Split-Brain scenario

OK, let's get back





walsender + walreceiver processes

- Leader has a walsender process (per Replica)

```
pg-red# ps uax | grep sender
  195 postgres  0:05 postgres: walsender root 172.20.0.3(41048) streaming 0/6766BDB8
```

- Replicas have a walreceiver process

```
pg-blue# ps uax | grep wal
   72 postgres  0:06 postgres: walreceiver streaming 0/66ABD7F8
```

Physical Replication in psql

- Leader has pg_replication_slots + pg_stat_replication
- Replicas have pg_stat_wal_receiver

```
root=# select * from pg_replication_slots where slot_name = 'blue';
-[ RECORD 1 ]-------+-----------
slot_name           | blue
plugin              |
slot_type           | physical
datoid              |
database            |
temporary           | f
active              | t
active_pid          | 195
xmin                |
catalog_xmin        |
restart_lsn         | 0/6FEF5DF0
confirmed_flush_lsn |
wal_status          | reserved
safe_wal_size       |
two_phase           | f
inactive_since      |
conflicting         |
invalidation_reason |
failover            | f
synced              | f
```

```
root=# select * from pg_stat_replication;
-[ RECORD 1 ]----+------------------------------
pid              | 195
usesysid         | 16385
usename          | root
application_name | walreceiver
client_addr      | 172.20.0.3
client_hostname  |
client_port      | 41048
backend_start    | 2025-01-21 06:14:28.084716+00
backend_xmin     |
state            | streaming
sent_lsn         | 0/736F0958
write_lsn        | 0/736F07C8
flush_lsn        | 0/736F07C8
replay_lsn       | 0/736F07C8
write_lag        | 00:00:00.000073
flush_lag        | 00:00:00.000193
replay_lag       | 00:00:00.000559
sync_priority    | 0
sync_state       | async
reply_time       | 2025-01-21 06:19:41.069414+00
```

Physical Replication status

Patroni

```
$ patronictl -c postgres0.yml list batman
+ Cluster: batman (7277694203142172922) -+----------+----+-----------+
| Member      | Host           | Role    | State     | TL | Lag in MB |
+-------------+----------------+---------+-----------+----+-----------+
| postgresql0 | 127.0.0.1:5432 | Leader  | running   |  5 |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming |  5 |         0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming |  5 |         0 |
+-------------+----------------+---------+-----------+----+-----------+
```

CloudNativePG

```
kubectl cnpg status sandbox
```

```
Cluster Summary
Name:                default/sandbox
System ID:           7423474350493388827
PostgreSQL Image:    ghcr.io/cloudnative-pg/postgresql:16.4
Primary instance:    sandbox-1
Primary start time:  2024-10-08 18:31:57 +0000 UTC (uptime 1m14s)
Status:              Cluster in healthy state
Instances:           3
Ready instances:     3
Size:                126M
Current Write LSN:   0/604DE38 (Timeline: 1 - WAL File: 000000010000000000000006)

Continuous Backup status
Not configured

Streaming Replication status
Replication Slots Enabled
Name       Sent LSN   Write LSN  Flush LSN  Replay LSN  Write Lag  Flush Lag  Replay Lag  State
----       --------   ---------  ---------  ----------  ---------  ---------  ----------  -----
sandbox-2  0/604DE38  0/604DE38  0/604DE38  0/604DE38   00:00:00   00:00:00   00:00:00    strea
sandbox-3  0/604DE38  0/604DE38  0/604DE38  0/604DE38   00:00:00   00:00:00   00:00:00    strea

Instances status
Name       Current LSN  Replication role  Status  QoS         Manager Version  Node
----       -----------  ----------------  ------  ---         ---------------  ----
sandbox-1  0/604DE38    Primary           OK      BestEffort  1.25.0           k8s-eu-worker
sandbox-2  0/604DE38    Standby (async)   OK      BestEffort  1.25.0           k8s-eu-worker2
sandbox-3  0/604DE38    Standby (async)   OK      BestEffort  1.25.0           k8s-eu-worker
```

Cluster Status

- Most info can be retrieved from previously mentioned commands
- Status: Cluster is in healthy state

  ~~ Primary ready and no Replica in creation (might be slightly off)

[Bug]: Cluster in healthy state despite "WAL archive check failed" #6137

Open

[Bug]: "Cluster is in healthy state" despite 0 running pods #5150

Open

Replication Conflicts

```
pg-blue# psql mydb
psql (17.2)
Type "help" for help.

mydb=# ALTER SYSTEM SET max_standby_streaming_delay ='100ms';
ALTER SYSTEM
mydb=# SELECT pg_reload_conf();
2025-01-21 06:31:19.419 UTC [68] LOG:  received SIGHUP, reloading configuration files
 pg_reload_conf
----------------
 t
(1 row)

mydb=# 2025-01-21 06:31:19.421 UTC [68] LOG:  parameter "max_standby_streaming_delay" changed to "100ms"

mydb=# begin; select * from pgbench_accounts   order by random(), random(), random();
BEGIN
2025-01-21 06:31:27.980 UTC [83] FATAL:  terminating connection due to conflict with recovery
2025-01-21 06:31:27.980 UTC [83] DETAIL:  User query might have needed to see row versions that must be removed.
2025-01-21 06:31:27.980 UTC [83] HINT:  In a moment you should be able to reconnect to the database and repeat your command.
```

FLUSH_LSN vs. REPLAY_LSN (save to replica's disk vs. visible in replica PG)

Switchover - Happy Path

- Procedure
  - Start with a healthy Leader + 2 Replicas
  - Stop a Leader, let Replicas catch up
  - Choose a New Leader
  - Create replication slots on a new Leader
  - Promote a new Leader
  - Make the Old Leader + 2nd Replica follow the New Leader
- Result
  - Another PG instance is the Leader, 2 Replicas follow
  - We didn't lose ANY data
  - WAL timeline has changed

Switchover - Happy Path - Demo - Stop Old Leader

```
[pg-red# su - postgres -c '/usr/local/bin/pg_ctl -D /var/lib/postgresql/data/ stop -m fast'
waiting for server to shut down....2025-01-21 06:48:38.495 UTC [337] LOG:  received fast shutdown request
2025-01-21 06:48:38.510 UTC [337] LOG:  aborting any active transactions
2025-01-21 06:48:38.513 UTC [338] LOG:  shutting down
2025-01-21 06:48:38.528 UTC [337] LOG:  database system is shut down
 done
server stopped
[pg-red#
[pg-red#
[pg-red# ##### START NEW LEADER HERE
```

Switchover - Happy Path - Demo - Start New Leader

```
pg-green#
pg-green#
pg-green# 2025-01-21 06:44:43.929 UTC [21] LOG:  replication terminated by primary server
2025-01-21 06:44:43.929 UTC [21] DETAIL:  End of WAL reached on timeline 1 at 0/B94CE3E8.
2025-01-21 06:44:43.930 UTC [21] FATAL:  could not send end-of-streaming message to primary: server closed the connection unexpect
edly
                This probably means the server terminated abnormally
                before or while processing the request.
        no COPY in progress
2025-01-21 06:44:43.931 UTC [20] LOG:  invalid record length at 0/B94CE3E8: expected at least 24, got 0
2025-01-21 06:44:43.942 UTC [22] FATAL:  could not connect to the primary server: connection to server at "pg-red" (172.20.0.2), p
ort 5432 failed: server closed the connection unexpectedly
                This probably means the server terminated abnormally
```

```
pg-green# psql
psql (17.2)
Type "help" for help.

root=# \x
Expanded display is on.
root=# SELECT * FROM pg_replication_slots;
(0 rows)

root=# SELECT pg_create_physical_replication_slot('red');
-[ RECORD 1 ]----------------------+-------
pg_create_physical_replication_slot | (red,)

root=# SELECT pg_create_physical_replication_slot('blue');
-[ RECORD 1 ]----------------------+--------
pg_create_physical_replication_slot | (blue,)

root=# SELECT pg_promote();
-[ RECORD 1 ]-
pg_promote | t
```

```
                Is the server running on that host and accepting TCP/IP connecti
2025-01-21 06:46:24.016 UTC [20] LOG:  waiting for WAL to become available at 0/
2025-01-21 06:46:29.016 UTC [51] FATAL:  could not connect to the primary server
ort 5432 failed: Connection refused
                Is the server running on that host and accepting TCP/IP connecti
2025-01-21 06:46:29.018 UTC [20] LOG:  waiting for WAL to become available at 0/
2025-01-21 06:46:32.349 UTC [20] LOG:  received promote request
2025-01-21 06:46:32.349 UTC [20] LOG:  redo done at 0/B94CE370 system usage: CPU
2025-01-21 06:46:32.350 UTC [20] LOG:  last completed transaction was at log tim
2025-01-21 06:46:32.352 UTC [20] LOG:  selected new timeline ID: 2
2025-01-21 06:46:32.371 UTC [20] LOG:  archive recovery complete
2025-01-21 06:46:32.373 UTC [18] LOG:  checkpoint starting: force
2025-01-21 06:46:32.375 UTC [17] LOG:  database system is ready to accept connec
```

Switchover - Happy Path - Demo - Old Leader As Replica

```
pg-red# ##### START NEW LEADER HERE
pg-red#
pg-red# touch /var/lib/postgresql/data/standby.signal
pg-red# echo >>/var/lib/postgresql/data/postgresql.auto.conf "primary_conninfo = 'user=root passfile=''/root/.pgpass'' channel_bin
ding=prefer host=''pg-green'' port=5432 sslmode=prefer sslnegotiation=postgres sslcompression=0 sslcertmode=allow sslsni=1 ssl_min
_protocol_version=TLSv1.2 gssencmode=prefer krbsrvname=postgres gssdelegation=0 target_session_attrs=any load_balance_hosts=disabl
e'"
pg-red# echo >>/var/lib/postgresql/data/postgresql.auto.conf "primary_slot_name = 'red'"
pg-red# su - postgres -c '/usr/local/bin/pg_ctl -D /var/lib/postgresql/data start'
waiting for server to start....2025-01-21 06:49:30.621 UTC [352] LOG:  starting PostgreSQL 17.2 on aarch64-unknown-linux-musl, com
piled by gcc (Alpine 14.2.0) 14.2.0, 64-bit
2025-01-21 06:49:30.621 UTC [352] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2025-01-21 06:49:30.621 UTC [352] LOG:  listening on IPv6 address "::", port 5432
2025-01-21 06:49:30.623 UTC [352] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2025-01-21 06:49:30.626 UTC [355] LOG:  database system was shut down in recovery at 2025-01-21 06:48:38 UTC
2025-01-21 06:49:30.626 UTC [355] LOG:  entering standby mode
2025-01-21 06:49:30.628 UTC [355] LOG:  consistent recovery state reached at 0/B94CE3E8
2025-01-21 06:49:30.628 UTC [355] LOG:  invalid record length at 0/B94CE3E8: expected at least 24, got 0
2025-01-21 06:49:30.628 UTC [352] LOG:  database system is ready to accept read-only connections
2025-01-21 06:49:30.632 UTC [356] LOG:  fetching timeline history file for timeline 2 from primary server
2025-01-21 06:49:30.634 UTC [356] LOG:  started streaming WAL from primary at 0/B9000000 on timeline 1
2025-01-21 06:49:30.638 UTC [356] LOG:  replication terminated by primary server
2025-01-21 06:49:30.638 UTC [356] DETAIL:  End of WAL reached on timeline 1 at 0/B94CE3E8.
2025-01-21 06:49:30.638 UTC [356] FATAL:  terminating walreceiver process due to administrator command
2025-01-21 06:49:30.639 UTC [355] LOG:  new target timeline is 2
2025-01-21 06:49:30.643 UTC [357] LOG:  started streaming WAL from primary at 0/B9000000 on timeline 2
2025-01-21 06:49:30.656 UTC [355] LOG:  redo starts at 0/B94CE3E8
 done
server started
```

Switchover - Happy Path - Demo - Replica Leader Switch

```
pg-blue# psql
psql (17.2)
Type "help" for help.

root=# ALTER SYSTEM SET primary_conninfo = 'user=root passfile=''/root/.pgpass'' channel_binding=prefer host=''pg-green'' port=543
2 ssl
refer krbsrvname=postgres gssdelegation=0 target_session_attrs=any load_balance_hosts=disable'
root-# ;
ALTER SYSTEM
root=# SELECT * FROM pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Condition: same replication slot name

Switchover - Happy Path - Logical Replication Slots

- Problem: Replication Slots from the Old Leader are not transferred
  - Typically a problem for Logical replication, CDC

Patroni:

On replicas that are eligible for a failover, Patroni creates the logical replication slot by copying the slot file from the primary and restarting the replica. In order to copy the slot file Patroni opens a connection to the primary with `rewind` or `superuser` credentials and calls `pg_read_binary_file()` function.

When the logical slot already exists on the replica Patroni periodically calls `pg_replication_slot_advance()` function, which allows moving the slot forward.

CloudNativePG

Standby HA slot: a physical replication slot for a standby whose lifecycle is entirely managed by another standby in the cluster, based on the content of the pg_replication_slots view in the primary, and updated at regular intervals using pg_replication_slot_advance().

Switchover - Waiting for Leader Shutdown

Can be made faster? Postgres will accept any new transactions after shutdown command.

Explicit CHECKPOINT before can shorted down-time period.

Shutdown modes are:

smart	quit after all clients have disconnected
fast	quit directly, with proper shutdown (default)
immediate	quit without complete shutdown; will lead to recovery on restart

CloudNativePG - Smart + Fast shutdown

- .spec.smartShutdownTimeout (no new connections only, but finish existing) + .spec.stopDelay, min 15s for WAL archiving

Leader Selection process

We should NOT promote:

- *replica not catching up WAL*
- *replica not being able to serve traffic as a Leader*
- *…*

Patroni approach - tags: nofailover + **(since 2023)** **failover_priority** : integer, controls the priority that this node should have during failover. Nodes with higher priority will be preferred over lower priority nodes if they received/replayed the same amount of WAL. However, nodes with higher values of receive/replay LSN are preferred regardless of their priority.

CloudNative PG approach

// Set the first pod in the sorted list as the new targetPrimary
return mostAdvancedInstance.Pod.Name, …

Leader Selection - Asymmetrical Instances

Some GUCs (config values) must be at least the same or greater:

max_wal_senders, max_replication_slots…

Used to be a thing in a past + with manual failover

Historical use case = have a small physical replica for backup purposes (that DBA scales manually during an incident)

Switchover - Unhappy Path

- Replica being shut down for a while
- Replica lost / has corrupted WAL files
- …

Options:

- wait (prefer consistency)
- promote = lose data (prefer availability)

(CAP theorem)

Prevention:

- monitoring

Failover - "Happy Path"

"Happy" = we are absolutely sure that Leader is dead and Replicas caught up

- Procedure
  - Start with 1 Leader (DEAD !) + healthy 2 Replicas
  - (!!!) Make sure Old Leader can't start
  - Choose the most advanced (?) Replica as a New Leader
  - Create a replication slot for Replica on a New Leader
  - Promote chosen Replica to a new Leader
  - Make 2nd Replica follow a new Leader
  - Make old Leader follow a new Leader BEFORE it starts (it MUST NOT get any writes)
- Result
  - Another PG instance is the Leader, 1 Replica follow
  - Minimal data loss
  - WAL timeline has changed
  - Leader is fenced, Old Leader can be turned to Replica (no split-brain) without reinit

Failover - "Happy Path" - Demo - Old Leader failure

```
pg-red# rm -rf /var/lib/postgresql/data/base/*
pg-red# 2025-01-21 19:21:46.957 UTC [480] FATAL:  database "template1" does not exist
2025-01-21 19:21:46.957 UTC [480] DETAIL:  The database subdirectory "base/1" is missing.
2025-01-21 19:21:48.177 UTC [471] LOG:  could not receive data from client: Connection reset by peer
2025-01-21 19:21:48.177 UTC [471] LOG:  unexpected EOF on client connection with an open transaction
2025-01-21 19:21:48.366 UTC [483] FATAL:  database "mydb" does not exist
2025-01-21 19:21:48.366 UTC [483] DETAIL:  The database subdirectory "base/16386" is missing.
2025-01-21 19:21:50.261 UTC [486] FATAL:  database "mydb" does not exist
2025-01-21 19:21:50.261 UTC [486] DETAIL:  The database subdirectory "base/16386" is missing.
```

```
pg-red# su - postgres -c '/usr/local/bin/pg_ctl -D /var/lib/postgresql/data stop'
waiting for server to shut down....2025-01-21 19:22:43.419 UTC [492] LOG:  received fast shutdown request
2025-01-21 19:22:43.422 UTC [492] LOG:  aborting any active transactions
2025-01-21 19:22:43.425 UTC [492] LOG:  background worker "logical replication launcher" (PID 498) exited with exit code 1
2025-01-21 19:22:43.427 UTC [493] LOG:  shutting down
2025-01-21 19:22:43.438 UTC [493] LOG:  checkpoint starting: shutdown immediate
2025-01-21 19:22:43.443 UTC [493] LOG:  checkpoint complete: wrote 0 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.001 s, sync
=0.001 s, total=0.006 s; sync files=0, longest=0.000 s, average=0.000 s; distance=0 kB, estimate=78502 kB; lsn=0/D52E3E8, redo lsn=0/D52E3E8
2025-01-21 19:22:43.456 UTC [492] LOG:  database system is shut down
 done
server stopped
pg-red#
pg-red#
pg-red# ### MAKE SURE PG CAN'T ACCIDENTALLY START
```

Failover - "Happy Path" - Demo - Inspect Replicas

```
root=#
root=# SELECT pg_current_wal_insert_lsn();
ERROR:  recovery is in progress
HINT:  WAL control functions cannot be executed during recovery.
root=# SELECT pg_current_wal_insert_lsn();
ERROR:  recovery is in progress
HINT:  WAL control functions cannot be executed during recovery.
root=# SELECT pg_is_in_recovery(),pg_is_wal_replay_paused(), pg_last_wal_receive_lsn(), pg_last_wal_replay_lsn(), pg_last_xact_replay_timestamp();
-[ RECORD 1 ]-----------------+------------------------------
pg_is_in_recovery             | t
pg_is_wal_replay_paused       | f
pg_last_wal_receive_lsn       | 0/D52E460
pg_last_wal_replay_lsn        | 0/D52E460
pg_last_xact_replay_timestamp | 2025-01-21 19:21:48.176914+00
```

PG functions to get WAL for Replica are different

Compare the latest receive LSN…

Leader Fencing

- Separating / Self-containing / Blocking PG instance Leader from either clients or other PG instances

- Rerouting clients to other PG instances
- Marking original Leader as "ill" / dead / needs maintenance

Probably the hardest part to get right in PG "product".

Failover - "Happy Path" - Demo - Promote New Leader

```
root=# SELECT pg_create_physical_replication_slot('red');
-[ RECORD 1 ]-------------------------+-------
pg_create_physical_replication_slot | (red,)

root=# SELECT pg_create_physical_replication_slot('green');
-[ RECORD 1 ]-------------------------+---------
pg_create_physical_replication_slot | (green,)

root=# SELECT pg_promote();
-[ RECORD 1 ]-
pg_promote | t
```

```
pg-green# psql
psql (17.2)
Type "help" for help.

root=# ALTER SYSTEM SET primary_conninfo = 'user=root passfile=''/root/.pgpass'' channel_binding=prefer host=''pg-blue'' port=5432 sslmode=prefer ss
lnegotiation=postgres sslcompression=0 sslcertmode=allow sslsni=1 ssl_min_protocol_version=TLSv1.2 gssencmode=prefer krbsrvname=postgres gssdelegati
on=0 target_session_attrs=any load_balance_hosts=disable'
root-# ;
ALTER SYSTEM
root=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Failover - "Unhappy Path"

- Procedure
  - Start with a Old Leader (alive) + healthy 2 Replicas
  - Forget about a Old Leader
  - Choose and promote any Replica to a New Leader
  - Make 2nd Replica follow a new Leader
  - (But some clients are writing to the original Leader at the same time or even maintenance - VACUUM, ANALYZE)
- Result
  - "Split brain" = we have 2 incompatible leaders (and timelines)
    - One needs to be discarded and data will be lost
    - (Don't even think about "merging" data together during such an incident)

Split Brain (pg-blue = Leader, promote pg-green)

- Conditions: Two or more Leaders have diverged in WAL
- Only possible triggers == pg_promote() or other PITR event (new timelineID)

```
pg-green# 2025-01-21 19:49:37.874 UTC [681] LOG:  received promote request
2025-01-21 19:49:37.874 UTC [822] FATAL:  terminating walreceiver process due to administrator command
2025-01-21 19:49:37.874 UTC [681] LOG:  invalid record length at 0/F9A6C78: expected at least 24, got 0
2025-01-21 19:49:37.876 UTC [681] LOG:  redo done at 0/F9A6C50 system usage: CPU: user: 2.20 s, system: 1.76 s, elapsed: 1759.46 s
2025-01-21 19:49:37.876 UTC [681] LOG:  last completed transaction was at log time 2025-01-21 19:49:37.873379+00
2025-01-21 19:49:37.879 UTC [681] LOG:  selected new timeline ID: 3
2025-01-21 19:49:37.929 UTC [681] LOG:  archive recovery complete
2025-01-21 19:49:37.931 UTC [679] LOG:  checkpoint starting: force
2025-01-21 19:49:37.935 UTC [678] LOG:  database system is ready to accept connections
```

pg-blue is still a Leader

```
root=# SELECT pg_current_wal_insert_lsn(), pg_walfile_name(pg_current_wal_lsn());
-[ RECORD 1 ]------------+------------------------
pg_current_wal_insert_lsn | 0/18365A00
pg_walfile_name           | 000000020000000000000018
```

Split Brain (pg-blue fails to follow pg-green)

Demote pg-blue and make it follow a new leader

```
pg-blue# 0-start-pg.sh
waiting for server to start....2025-01-21 19:54:47.950 UTC [377] LOG:  starting PostgreSQL 17.2 on aarch64-unknown-linux-musl, compiled by gcc (Alpi
ne 14.2.0) 14.2.0, 64-bit
2025-01-21 19:54:47.950 UTC [377] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2025-01-21 19:54:47.950 UTC [377] LOG:  listening on IPv6 address "::", port 5432
2025-01-21 19:54:47.952 UTC [377] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2025-01-21 19:54:47.954 UTC [380] LOG:  database system was shut down at 2025-01-21 19:53:23 UTC
2025-01-21 19:54:47.954 UTC [380] LOG:  entering standby mode
2025-01-21 19:54:47.958 UTC [380] LOG:  consistent recovery state reached at 0/1A7E68D8
2025-01-21 19:54:47.958 UTC [380] LOG:  invalid record length at 0/1A7E68D8: expected at least 24, got 0
2025-01-21 19:54:47.958 UTC [377] LOG:  database system is ready to accept read-only connections
2025-01-21 19:54:47.962 UTC [381] LOG:  fetching timeline history file for timeline 3 from primary server
2025-01-21 19:54:47.963 UTC [381] FATAL:  could not start WAL streaming: ERROR:  requested starting point 0/1A000000 on timeline 2 is not in this se
rver's history
        DETAIL:  This server's history forked from timeline 2 at 0/F9A6C78.
2025-01-21 19:54:47.965 UTC [380] LOG:  new timeline 3 forked off current database system timeline 2 before current recovery point 0/1A7E68D8
2025-01-21 19:54:47.968 UTC [382] FATAL:  could not start WAL streaming: ERROR:  requested starting point 0/1A000000 on timeline 2 is not in this se
rver's history
        DETAIL:  This server's history forked from timeline 2 at 0/F9A6C78.
2025-01-21 19:54:47.968 UTC [380] LOG:  new timeline 3 forked off current database system timeline 2 before current recovery point 0/1A7E68D8
2025-01-21 19:54:47.968 UTC [380] LOG:  waiting for WAL to become available at 0/1A002000
 done
server started
```

Split Brain Detection

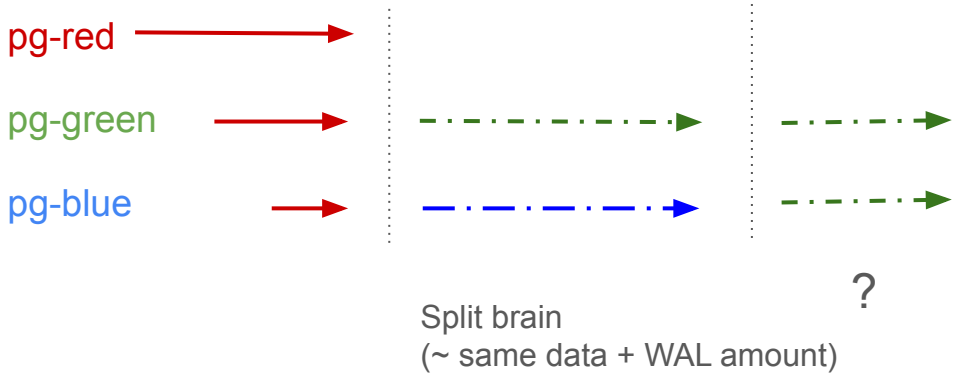
```
1  SELECT * FROM pg_timelines;
```

- Ultimate verification - PG instance can or can't follow the New Leader
- Multiple Leader in the cluster
- Different timelines branching:
  - Old Leader timelineID=x has LSN > branching point of New Leader with timelineID=x+1
  - Two New Leaders with the same timelineID have a different branching point
- last WAL .history file is kept

- What if we branch at the same LSN? (2 caught-up Replicas both pg_promote'd)

Split-Brain at the same LSN Experiment

- Procedure:
  - 1 Leader + 2 Replicas
  - Stop Leader, let both Replicas catch up
  - Promote both Replicas (2nd Leader, 3rd Leader) at the same time (LSN)
  - Do the same operations and ~ WAL amount change on both Replicas (INSERTs)
  - Try to turn 3rd Leader to follow the 2nd Leader




```
2025-01-22 18:33:40.673 UTC [163] LOG:  invalid resource manager ID 86 at 0/22C7E558
2025-01-22 18:33:40.673 UTC [163] LOG:  waiting for WAL to become available at 0/22002000
2025-01-22 18:33:40.674 UTC [163] LOG:  invalid resource manager ID 86 at 0/22C7E558
2025-01-22 18:33:40.674 UTC [163] LOG:  waiting for WAL to become available at 0/22002000
 done
server started
pg-blue# 2025-01-22 18:33:45.684 UTC [163] LOG:  invalid resource manager ID 86 at 0/22C7E558
2025-01-22 18:33:45.684 UTC [163] LOG:  waiting for WAL to become available at 0/22002000
```

Leader Fencing - approaches

- SystemD - block from starting
- Disable VM
- Take VM off the network

Patroni

- Demotes Patroni process + shutdown PG

CloudNativePG

- Different meaning (manual action of keeping "empty Pod" running for inspection)
- "Fencing" during failover - PG shutdown

Rescuing an Old Leader

- Must restore exact clone of New Leader
- Not possible if timelines have diverged?

… or pg_rewind

Patroni

pg_rewind + pg_basebackup + backup tools

CloudNativePG

pg_rewind + pg_basebackup + Barman backup

pg_rewind

- Idea: *"pg_rewind examines the timeline histories of the source and target clusters to determine the point where they diverged, and expects to find WAL in the target cluster's pg_wal directory reaching all the way back to the point of divergence."*

- If pg_rewind fails => pg_basebackup
- Rewinding old Leader to become Replica
  - also - for example resetting a test cluster to some default state quickly

Rescuing an Old Leader - Demo

Procedure:

- Split Brain has occured in "recent" past
- Stop Old Leader
- pg_rewind
- Make Old Leader a Replica

- data_checksums or wal_log_hints required
- Having a WAL archive is likely a must for active workloads

```
pg-blue# su - postgres -c 'pg_rewind -D /var/lib/postgresql/data --source-server=host=pg-green -P'
pg_rewind: connected to server
pg_rewind: servers diverged at WAL location 0/F9A6C78 on timeline 2
pg_rewind: error: could not open file "/var/lib/postgresql/data/pg_wal/0000000200000000000000E": No such file or directory
pg_rewind: error: could not find previous WAL record at 0/EFFFFB8
```

Rescuing an Old Leader - Demo

```
pg-red# su - postgres -c '/usr/local/bin/pg_ctl -D /var/lib/postgresql/data stop'
2025-01-22 07:53:55.836 UTC [100] LOG:  received fast shutdown request
waiting for server to shut down....2025-01-22 07:53:55.838 UTC [100] LOG:  aborting any active transactions
2025-01-22 07:53:55.838 UTC [126] FATAL:  terminating connection due to administrator command
2025-01-22 07:53:55.842 UTC [100] LOG:  background worker "logical replication launcher" (PID 106) exited with exit code 1
2025-01-22 07:53:55.843 UTC [101] LOG:  shutting down
2025-01-22 07:53:55.844 UTC [101] LOG:  checkpoint starting: shutdown immediate
2025-01-22 07:53:55.878 UTC [101] LOG:  checkpoint complete: wrote 3813 buffers (23.3%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.013 s, sync=0.016
 s, total=0.036 s; sync files=28, longest=0.014 s, average=0.001 s; distance=133275 kB, estimate=133275 kB; lsn=0/C226E88, redo lsn=0/C226E88
2025-01-22 07:53:55.885 UTC [100] LOG:  database system is shut down
 done
server stopped
pg-red#
pg-red#
pg-red# su - postgres -c 'pg_rewind -D /var/lib/postgresql/data --source-server=host=pg-green -P'
pg_rewind: connected to server
pg_rewind: servers diverged at WAL location 0/ADC5450 on timeline 1
pg_rewind: rewinding from last common checkpoint at 0/407B948 on timeline 1
pg_rewind: reading source file list
pg_rewind: reading target file list
pg_rewind: reading WAL in target
pg_rewind: need to copy 238 MB (total source directory size is 270 MB)
243961/243961 kB (100%) copied
pg_rewind: creating backup label and updating control file
pg_rewind: syncing target data directory
pg_rewind: Done!
pg-red# ls /var/lib/postgresql/data/
PG_VERSION          global              pg_ident.conf       pg_replslot         pg_stat_tmp         pg_wal
backup_label        pg_commit_ts        pg_logical          pg_serial           pg_subtrans         pg_xact
backup_label.old    pg_dynshmem         pg_multixact        pg_snapshots        pg_tblspc           postgresql.auto.conf
base                pg_hba.conf         pg_notify           pg_stat             pg_twophase         postgresql.conf
```

PG Instance Keeps Same Config After a VM/Pod Restart

- It might continue serving as Leader even another one was already selected
- It might continue to follow previous Leader who has changed since

No good solution without additional tools, some other process needs to check who should be the leader and react to it.

Patroni

- Patroni process is wrapping PG instance and manages its lifecycle fully

CloudNativePG

- Operator starts first and decides what to do and how to start/kill Pods
- Postgres Instance manager process wraps PG instance inside the Pod

Source of Truth (DCS) is Lost / Unable to Write

- Source of Truth will fail sometimes
- Trade-off
  - Either be safe (all nodes as Replicas)
  - or keep the current state (might lead to split brain)

Practical example: maintenance on etcd for Patroni / Kubernetes cluster

Patroni

- Failsafe DCS Mode

CloudNativePG

- Not your problem - without kube-apiserver you can't make any changes to Pods anyway
- Leader election for Controller

Is the Leader healthy? Can we detect a problem fast?

- Low level
  - Is the port opened
  - Can we login
  - VM overloaded
  - Out of disk space

- But also be tolerant to glitches

Patroni

- DCS Lock
- Watchdog

CloudNativePG

- Kubernetes Pod probes
- Controller

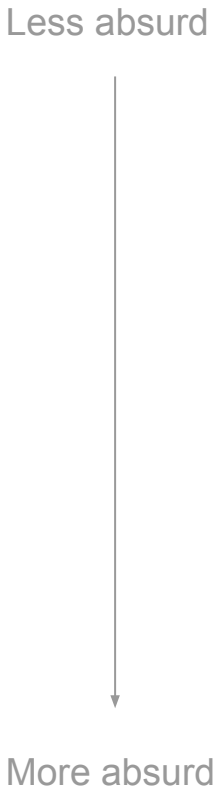
Poor Clients

- With each Leader switch, the clients needs to reconnect
- Good support in libpq, however not with all drivers
- Typically a problem fencing

- Many options, no single winner
  - haproxy
  - pgbouncer
  - Kubernetes networking
  - …

Key Takeaway

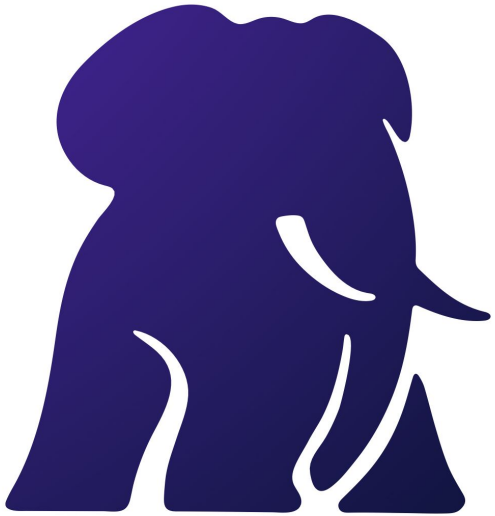
- You have a good failover plan
- You have tested your backups & disaster-recovery plan
- You have a good monitoring, so the replicas are healthy
- You can was up any time and instantly be focused 100%
- You don't do any mistakes
- And you execute the failover perfectly with downtime <100min
- And you execute the failover perfectly with downtime <10min
- And you execute the failover perfectly with downtime <1min



Key Takeaway

Just don't manage PG cluster on you own, use a proven tool.

Apologies, Patroni logo might be incorrect



That's all I had…

GitHub Repo