Tips and tools for minimal downtime in PostgreSQL upgrades

data egret
Your remote PostgreSQL DBA team

Stefan FERCOT

stefan.fercot@dataegret.com

Securing your PostgreSQL database availability and high performance.

- Performance audit
- Backup & restore
- Migration
- Cloud Cost Management
- Architecture review
- DataOps/ CDC projects
- 24/7 Incident support

● ● ●

on premises & cloud

EXPERTISE

Senior DBA team with **10+ years of PostgreSQL experience** each.

DEVELOPMENT

Involved in **new feature and extension development**.

TAILORED APPROACH

Dedicated DBA team that focused on success of your project.

COMMUNITY

Recognised significant **contributing sponsor to PostgreSQL**.

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 <u>backups</u> and <u>high-availability</u>!*

How to achieve minimal downtime in PostgreSQL upgrades?

How it started…

Wednesday, April 3rd 2024

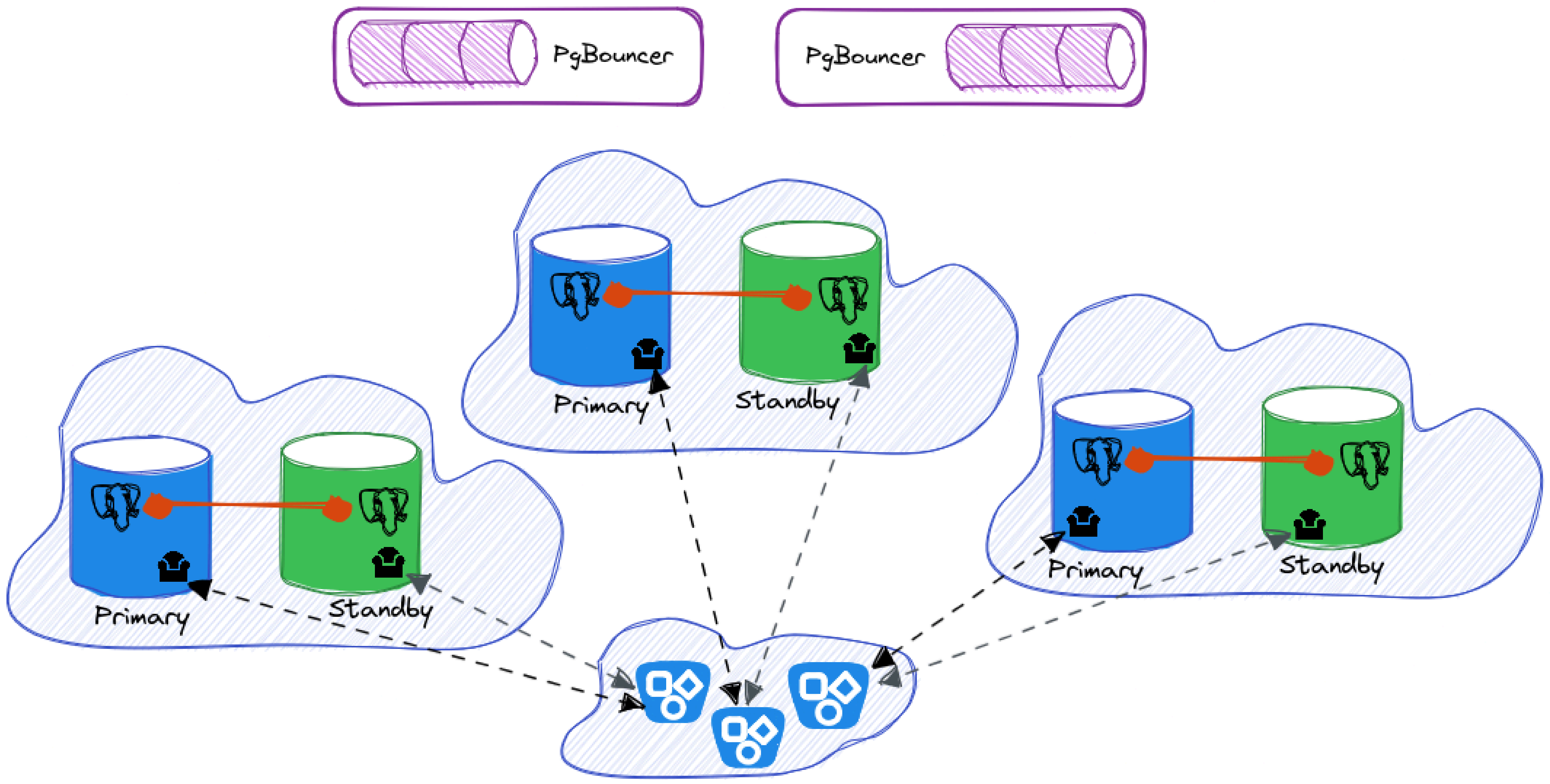
Hey guys, we think about moving to postgres 16 on production. We upgraded postgres on stage and in our tests, everything goes fine so far. Can we schedule the upgrade, please?

Happy customer

Friday, April 4th 2025

Hey folks. What are your thoughts on bumping our DB to PG 17?

Their setup


PgBouncer
PgBouncer
Primary
Standby
Primary
Standby
Primary
Standby

Today’s agenda

- PostgreSQL upgrade strategies
- how we achieved it with minimal downtime
  - implementation details (inc. PgBouncer and pgBackRest)
  - step-by-step example

PostgreSQL upgrade strategies

https://www.postgresql.org/docs/current/upgrading.html

- upgrading data via `pg_dumpall`
- upgrading data via `pg_upgrade`
- upgrading data via logical replication

Logical export

The traditional method for moving data to a new major version is to dump and restore the database, though this can be slow.

`pg_upgrade`

https://www.postgresql.org/docs/current/pgupgrade.html

- in-place migration = both version needs to be installed on the server
  - `--copy` : copy files to the new cluster
  - `--link` : use hard links instead of copying files
  - `--clone` : use efficient file cloning (aka. “reflinks”)
  - (`--swap` : new in PostgreSQL 18!)
- can be performed very fast, particularly with `--link` mode

`pg_upgrade --link`

If you did start the new cluster, it has written to shared files and it is unsafe to use the old cluster. The old cluster will need to be restored from backup in this case.

Logical replication

- setup logical replication from old cluster to new one
  - and then perform the switch-over
- flexible but complex setup
  - initial sync could take a long time
- logical replication limitations also apply

Short summary

Method	Downtime	Extra disk space	Complexity	Riskiness
dump/restore	high	double	low	low
pg_upgrade (copy)	high	double	medium	low
pg_upgrade (link)	low	low	medium	high
logical replication	low	double	high	low

Useful resources

- PGConf.DE 2023 - An ultimate guide to upgrading your PostgreSQL installation, by Ilya Kosmodemiansky
- PGConf India 2024 - pg_upgrade like a boss!, by Alexander Kukushkin
- https://www.crunchydata.com/blog/examining-postgres-upgrades-with-pg_upgrade
- https://www.cybertec-postgresql.com/en/upgrading-and-updating-postgresql
- https://www.cybertec-postgresql.com/en/upgrading-postgres-major-versions-using-logical-replication

What to choose?

- lower downtime, disk space and complexity
  - `pg_upgrade --link`

Riskiness

Because “link” mode was used, the old cluster cannot be safely started once the new cluster has been started.

- problem in case the upgrade fails
  - use *Streaming Replication*
  - rely on the standby server to mitigate the risk

How to upgrade the replica?

pgupgrade.html#PGUPGRADE-STEP-REPLICAS

```
rsync --archive --delete --hard-links --size-only --no-inc-recursive /opt/PostgreSQL/12 \
      /opt/PostgreSQL/17 standby.example.com:/opt/PostgreSQL
```

- downtime of `pg_upgrade --link` + `rsync` depends
  - only on the number of objects in the cluster
  - not on the total size of data
  - but this needs SSH access between the primary and standby hosts !

How to NOT upgrade the replica?

> *If you do not want to use rsync, or want an easier solution, simply recreate the standby servers once pg_upgrade completes and the new primary is running.*

- use pgBackRest, instead of `pg_basebackup`, to rebuild the standby!

Summary

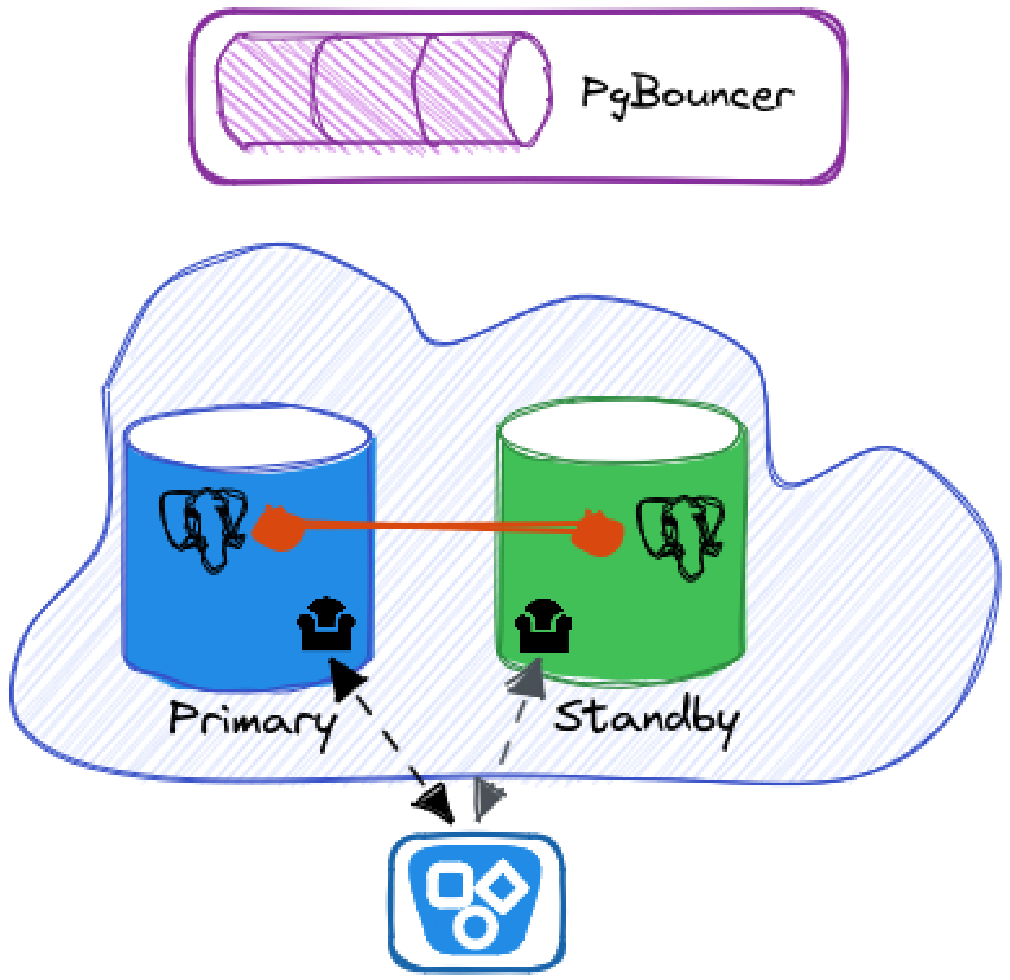
1. shutdown primary and standby
2. run `pg_upgrade --link` on the primary
3. bring primary back on

- if everything’s fine
  - take a fresh backup
  - reconstruct standby
- else
  - promote old standby or restore a backup

Minimizing downtime

- how to route traffic and avoid downtime during the maintenance task?
  - use PgBouncer `PAUSE` and `RESUME`

Implementation details



vagrant-playgrounds

PgBouncer

- use *systemd* `ReusePort=true` and *PgBouncer* `peer_id`
- port **7432**
  - connection url to the **primary** (`host=postgres0 port=5432`)
  - sockets 10001 and 10002
- port **7433**
  - connection url to the **standby** (`host=postgres1 port=5432`)
  - sockets 10101 and 10102

PgBouncer configuration (1)

`/etc/pgbouncer/pgbouncer-primary-10001.ini`

```
[databases]
testdb = host=postgres0 port=5432 dbname=testdb

[peers]
1 = host=/run/postgresql port=10001
2 = host=/run/postgresql port=10002

[pgbouncer]
listen_addr = 0.0.0.0
listen_port = 7432
peer_id = 1
```

PgBouncer configuration (2)

`/etc/pgbouncer/pgbouncer-primary-10002.ini`

```
[databases]
testdb = host=postgres0 port=5432 dbname=testdb

[peers]
1 = host=/run/postgresql port=10001
2 = host=/run/postgresql port=10002

[pgbouncer]
listen_addr = 0.0.0.0
listen_port = 7432
peer_id = 2
```

PgBouncer configuration (3)

`/etc/systemd/system/pgbouncer_primary@.service`

```
[Unit]
Description=connection pooler for PostgreSQL (%i)
After=network.target
Requires=pgbouncer_primary@%i.socket

[Service]
Type=notify
User=postgres
ExecStart=/usr/sbin/pgbouncer /etc/pgbouncer/pgbouncer-primary-%i.ini
ExecReload=/bin/kill -HUP $MAINPID
KillSignal=SIGINT

[Install]
WantedBy=multi-user.target
```

PgBouncer configuration (4)

`/etc/systemd/system/pgbouncer_primary@.socket`

```
[Unit]
Description=sockets (%i) for PgBouncer

[Socket]
ListenStream=0.0.0.0:7432
ListenStream=0.0.0.0:%i
ListenStream=/run/postgresql/.s.PGSQL.%i
ReusePort=true

[Install]
WantedBy=sockets.target
```

PgBouncer sockets

```
$ sudo systemctl enable --now pgbouncer_primary@10001.socket
$ sudo systemctl enable --now pgbouncer_primary@10002.socket
$ sudo systemctl enable --now pgbouncer_standby@10101.socket
$ sudo systemctl enable --now pgbouncer_standby@10102.socket
```

```
$ sudo systemctl list-sockets | grep pgbouncer_primary
0.0.0.0:10001                        pgbouncer_primary@10001.socket
0.0.0.0:10002                        pgbouncer_primary@10002.socket
0.0.0.0:7432                         pgbouncer_primary@10001.socket
0.0.0.0:7432                         pgbouncer_primary@10002.socket
/run/postgresql/.s.PGSQL.10001       pgbouncer_primary@10001.socket
/run/postgresql/.s.PGSQL.10002       pgbouncer_primary@10002.socket
```

```
$ sudo systemctl list-sockets | grep pgbouncer_standby
0.0.0.0:10101                        pgbouncer_standby@10101.socket
0.0.0.0:10102                        pgbouncer_standby@10102.socket
0.0.0.0:7433                         pgbouncer_standby@10102.socket
0.0.0.0:7433                         pgbouncer_standby@10101.socket
/run/postgresql/.s.PGSQL.10101       pgbouncer_standby@10101.socket
/run/postgresql/.s.PGSQL.10102       pgbouncer_standby@10102.socket
```

pgBackRest

`/etc/pgbackrest.conf`

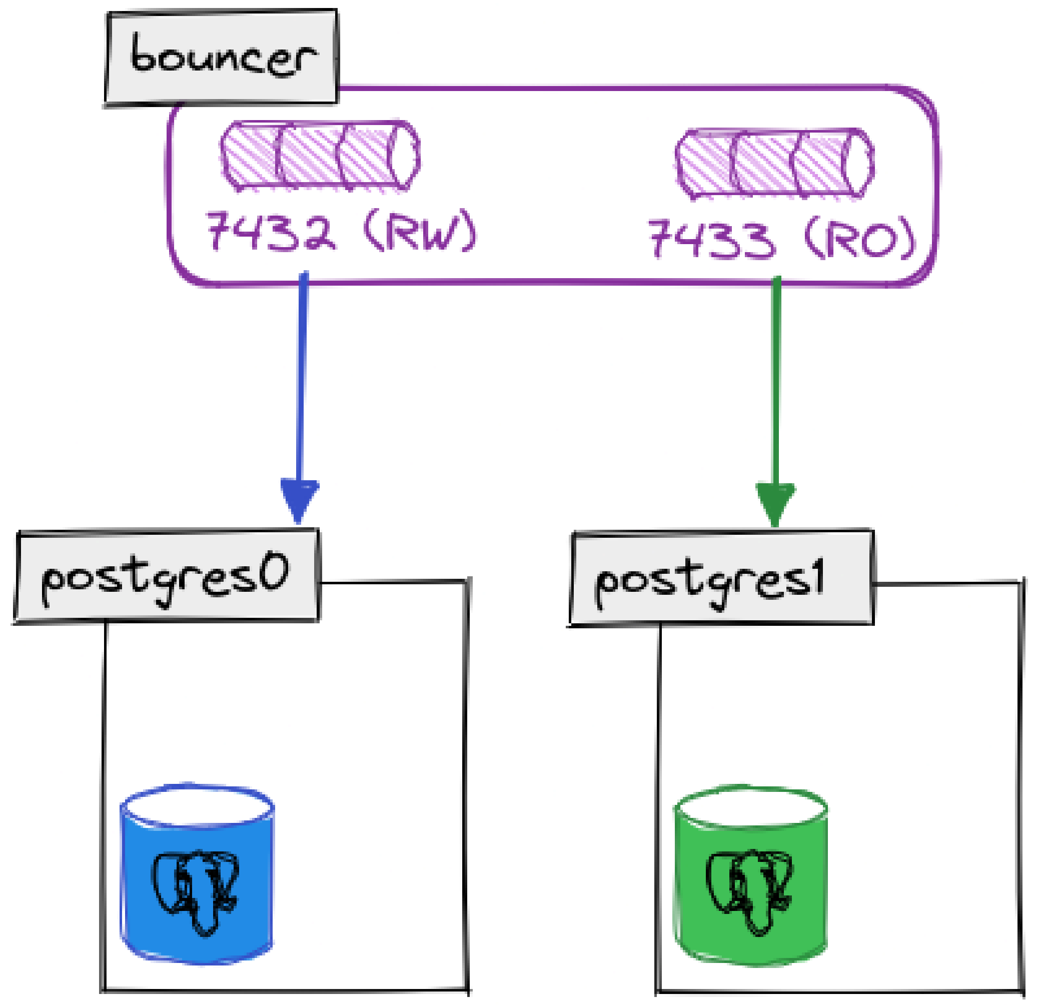
```
[global]
repo1-type=posix
repo1-path=/shared/backups
repo1-bundle=y
repo1-retention-full=4
compress-type=zst
start-fast=y
delta=y
log-level-console=info
log-level-file=detail

[mycluster]
pg1-path=/var/lib/postgresql/16/main
recovery-option=primary_conninfo=host=postgres1 port=5432 user=replicator
```

Step-by-step example

- upgrade the PostgreSQL version on the primary
  - give us a new primary with the new PostgreSQL version installed
  - old standby would still be there in case something goes wrong
- take a new backup using pgBackRest
- use that backup to reconstruct the standby

Connections state (initial)



Check that the upgrade is possible

```
$ /usr/lib/postgresql/17/bin/pg_upgrade \
--old-bindir /usr/lib/postgresql/16/bin/ --new-bindir /usr/lib/postgresql/17/bin/ \
--old-datadir /var/lib/postgresql/16/main/ --new-datadir /var/lib/postgresql/17/main/ \
--old-options " -c config_file=/etc/postgresql/16/main/postgresql.conf" \
--new-options " -c config_file=/etc/postgresql/17/main/postgresql.conf" \
--new-options " -c archive_mode=off" \
--retain --jobs 4 --link --check
```

Clusters are compatible

Stop the primary (1)

- pause traffic (SQL queries) to the primary

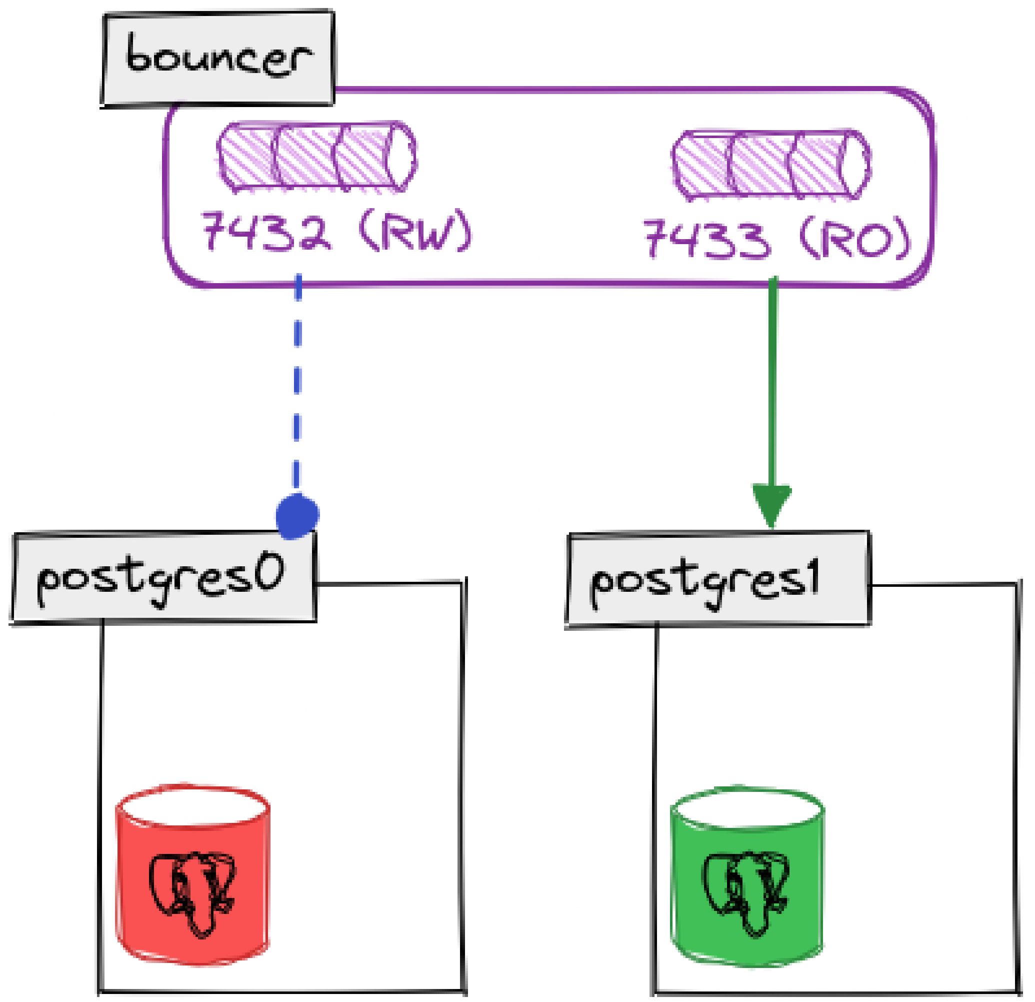
```
$ psql -U pgbouncer -p 10001 -c "PAUSE;"
$ psql -U pgbouncer -p 10002 -c "PAUSE;"
```

Stop the primary (2)

- stop the primary server
  - (manual CHECKPOINT + `pg_ctl stop -m fast`)
- make sure it was stopped properly

```
$ /usr/lib/postgresql/16/bin/pg_controldata -D /var/lib/postgresql/16/main \
    |grep -E '(Database cluster state)|(REDO location)'
Database cluster state:               shut down
Latest checkpoint's REDO location:    0/D000028
```

Connections state



Stop the standby (1)

- pause traffic to the standby

```
$ psql -U pgbouncer -p 10101 -c "PAUSE;"
$ psql -U pgbouncer -p 10102 -c "PAUSE;"
```

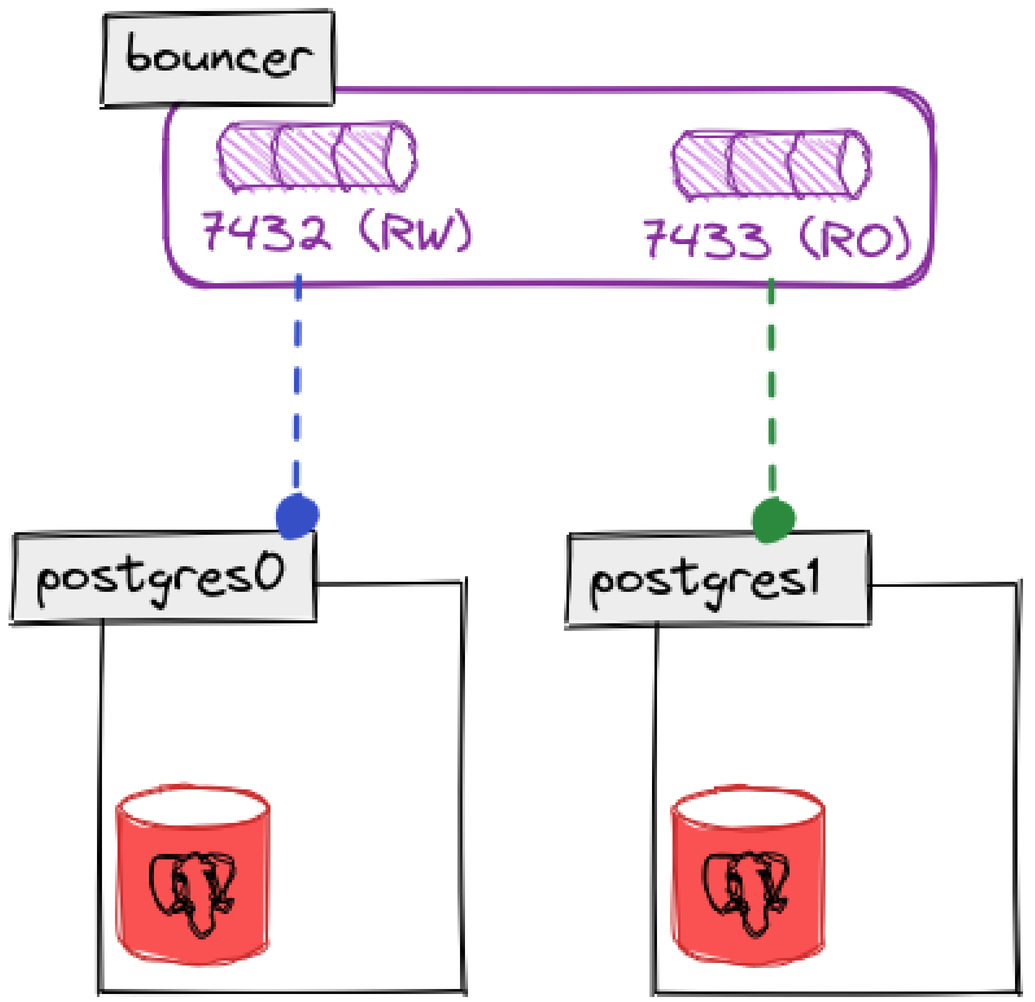
Stop the standby (2)

- stop the standby server
- make sure it was stopped properly

```
$ /usr/lib/postgresql/16/bin/pg_controldata -D /var/lib/postgresql/16/main \
    |grep -E '(Database cluster state)|(REDO location)'
Database cluster state:               shut down in recovery
Latest checkpoint's REDO location:    0/D000028
```

- could be done in parallel with `pg_upgrade`

Connections state



PostgreSQL upgrade (1)

- perform the upgrade using the previous command (without `--check`)

```
Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade.
Once you start the new server, consider running:
    /usr/lib/postgresql/16/bin/vacuumdb --all --analyze-in-stages
Running this script will delete the old cluster's data files:
    ./delete_old_cluster.sh
```

PostgreSQL upgrade (2)

- upgrade pgBackRest configuration

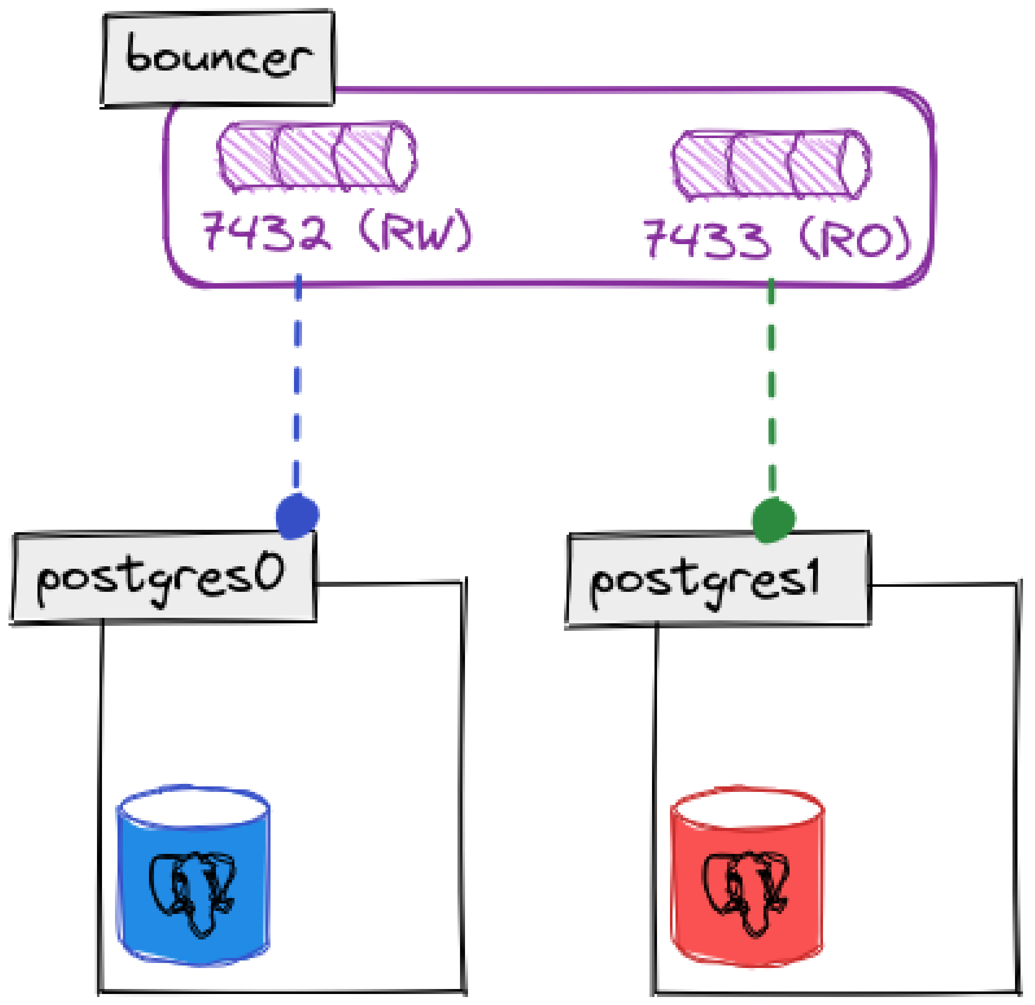
```
$ sed -i 's/\/var\/lib\/postgresql\/16\/main/\/var\/lib\/postgresql\/17\/main/g' \
    /etc/pgbackrest.conf
$ pgbackrest --stanza=mycluster --no-online stanza-upgrade
```

PostgreSQL upgrade (3)

- start the new upgraded cluster
- run `vacuumdb` as suggested in the `pg_upgrade` output

```
$ /usr/lib/postgresql/17/bin/vacuumdb --all --analyze-in-stages
```

Connections state



PostgreSQL upgrade (4)

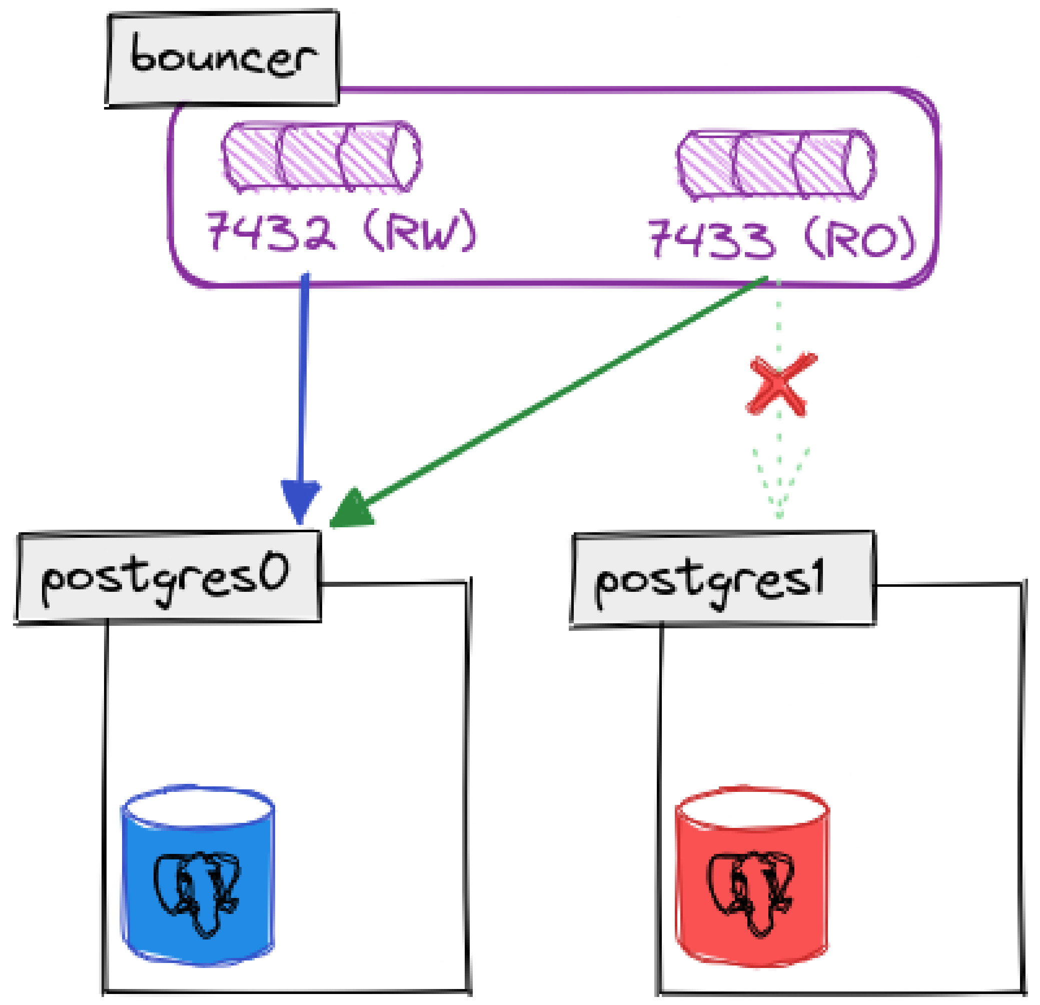
- resume traffic to the primary

```
$ psql -U pgbouncer -p 10001 -c "RESUME;"
$ psql -U pgbouncer -p 10002 -c "RESUME;"
```

- redirect and resume the traffic from the standby to the primary

```
$ sed -i 's/postgres1/postgres0/g' /etc/pgbouncer/pgbouncer-standby-10101.ini
$ sed -i 's/postgres1/postgres0/g' /etc/pgbouncer/pgbouncer-standby-10102.ini
$ psql -U pgbouncer -p 10101 -c "RELOAD;"
$ psql -U pgbouncer -p 10102 -c "RELOAD;"
$ psql -U pgbouncer -p 10101 -c "RESUME;"
$ psql -U pgbouncer -p 10102 -c "RESUME;"
```

Connections state



PostgreSQL upgrade (5)

- make a fresh backup

```
$ pgbackrest --stanza=mycluster --type=full backup
$ pgbackrest --stanza=mycluster info
```

- consider adding `--exclude=pg_upgrade_output.d`

PostgreSQL upgrade (6)

- once you’re sure that everything is fine with the new cluster…
  - clean-up the old cluster directory and packages

```
$ rm -rf '/var/lib/postgresql/16/main'
$ sudo apt-get remove -y postgresql-16
```

Reconstruct standby (1)

- re-sync the standby using pgBackRest `--delta` restore

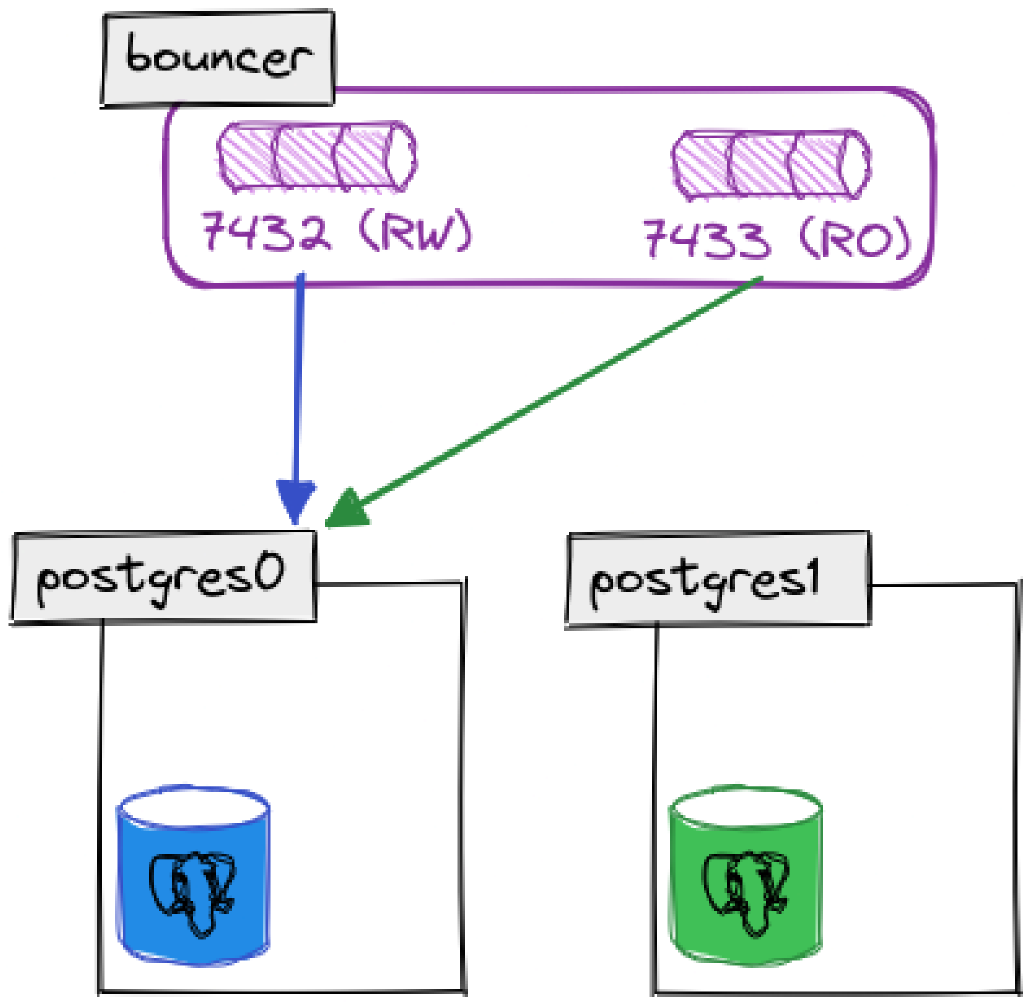
```
$ sed -i 's/\/var\/lib\/postgresql\/16\/main/\/var\/lib\/postgresql\/17\/main/g' \
    /etc/pgbackrest.conf
$ pgbackrest --stanza=mycluster --delta --process-max=4 --type=standby restore
```

Reconstruct standby (2)

- start the standby server
- make sure it catches up with the primary replication state

```
$ psql -xtc "SELECT status,sender_host FROM pg_stat_wal_receiver;"
status                    | streaming
sender_host               | postgres1
```

Connections state



Reconstruct standby (3)

- resume traffic back to the standby

```
$ sed -i 's/postgres0/postgres1/g' /etc/pgbouncer/pgbouncer-standby-10101.ini
$ sed -i 's/postgres0/postgres1/g' /etc/pgbouncer/pgbouncer-standby-10102.ini
$ psql -U pgbouncer -p 10101 -c "RELOAD;"
$ psql -U pgbouncer -p 10102 -c "RELOAD;"
$ psql -U test -d testdb -p 10101 -c "SELECT pg_is_in_recovery();"
$ psql -U test -d testdb -p 10102 -c "SELECT pg_is_in_recovery();"
```

Conclusion

- for small and medium size clusters
  - major upgrade usually takes only 10s-20s of downtime
  - **excluding statistics rebuild**!
- most clusters could be upgraded in less than 1 minute
- even less impact thanks to PgBouncer `PAUSE` / `RESUME`

Waiting for PostgreSQL 18

```
$ git show 1fd1bd871012732e3c6c482667d2f2c56f1a9395
Author: Jeff Davis <jdavis@postgresql.org>
Date:   Thu Feb 20 01:29:06 2025 -0800
    Transfer statistics during pg_upgrade.

    Add support to pg_dump for dumping stats, and use that during
    pg_upgrade so that statistics are transferred during upgrade. In most
    cases this removes the need for a costly re-analyze after upgrade.

    Some statistics are not transferred, such as extended statistics or
    statistics with a custom stakind.
```

`vacuumdb --missing-stats-only`

Final Thoughts

Don’t be scared, be prepared

- using proper tools helps manage downtime
  - both when things go right and when they don’t
- upgrading on time makes it easier
- extensions can surprise you
  - some of them (e.g., PostGIS) require special care

PostgreSQL 💙 Belgium

PgBE PostgreSQL Users Group Belgium meetup group

PG Day France - Mons, June 3-4

Thank You!

contact@dataegret.com