Achieving minimal downtime in PostgreSQL maintenance tasks

data egret

Your remote PostgreSQL DBA team

Stefan FERCOT

stefan.fercot@dataegret.com

SECURING YOUR DATABASE AVAILABILITY, SO THAT YOUR TEAM CAN FOCUS ON NEW FEATURE DEVELOPMENT.

- Migrations
- DB audit
- Performance optimisation
- Backup & restore
- Architectural review
- Advising Data Science teams
- Developer training

on premise & cloud

EXPERTISE

Senior DBA with **10+ years** of PostgreSQL administration **experience**

DEVELOPMENT

Involved in **new feature and extension development**

TAILORED APPROACH

Felxible approach and **dedicated team** focused on success of your project

COMMUNITY

Contributing Sponsor. Deeply involved in the PostgreSQL community

Data Egret GmbH | 66583 Spiesen-Elversberg | Germany

contact@DataEgret.com

Stefan Fercot

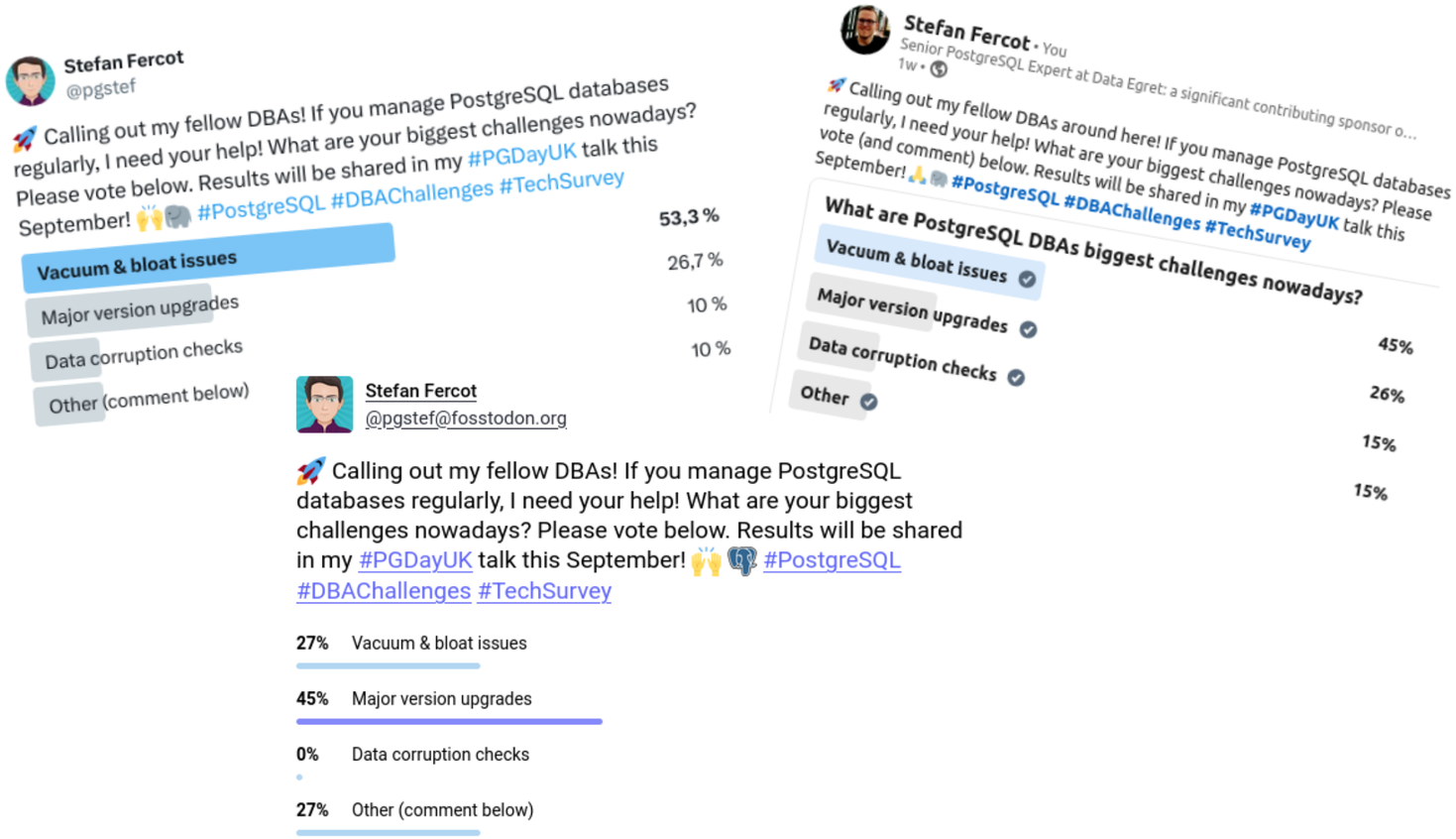
- Senior PostgreSQL Expert **@Data Egret**
- pgBackRest fan & contributor
- aka. pgstef
- https://pgstef.github.io

Need a Disaster and Recovery Plan? ;-)
*Contact **Data Egret** to talk to me about backups and high-availability!*

What are PostgreSQL DBAs biggest challenges nowadays?

#blamevacuum

Quick social-media poll

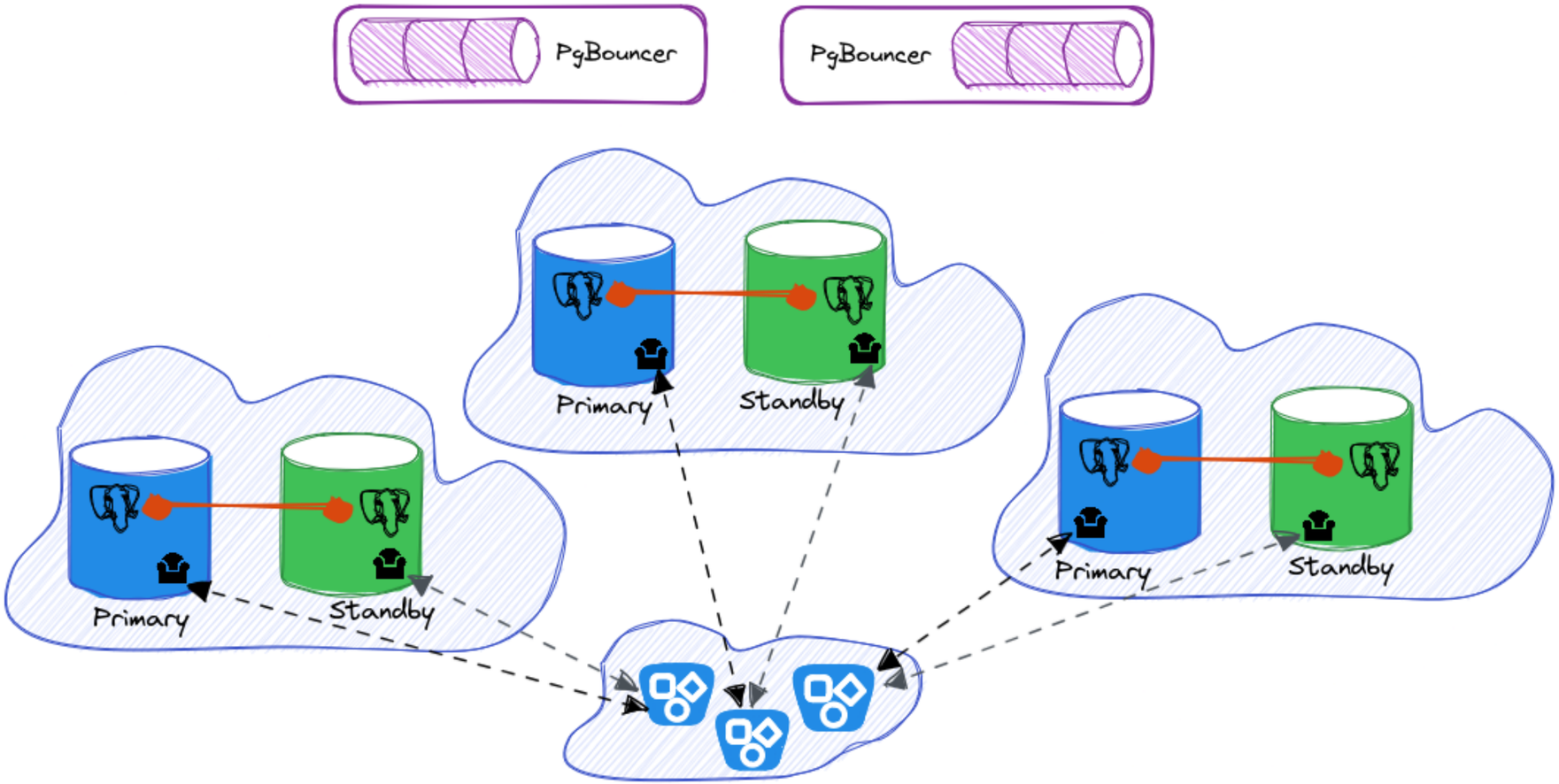


How it started…

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?

The customer setup



Today’s agenda

- page checksums
- PostgreSQL upgrade strategies
- how we achieved it with minimal downtime
  - implementation details
  - step-by-step example

Page checksums

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

- Why is it important?
  - ensures data integrity and helps in early detection of corruption
- What's the problem?
  - not enabled by default (see pgsql-hackers)
  - cluster must be offline during the enabling process, which can be slow

Verifying the checksums

- `pg_checksums` can enable/disable/verify checksums…
  - on an offline cluster!
- pgBackRest `--checksum-page` option
  - validate all data page checksums while backing up a cluster
  - automatically enabled when data page checksums are enabled

`pg_checksums`

https://www.postgresql.org/docs/current/app-pgchecksums.html

performing the operation on the primary and finally recreating the standbys from scratch is safe

Possible procedure

1. shutdown standby
2. enable page checksums
3. bring standby back on
4. turn off primary and promote the standby
5. reconstruct old primary as new standby

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”)
- 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

Possible procedure

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

- if everything's fine
  - reconstruct standby
- else
  - promote old standby or restore a backup

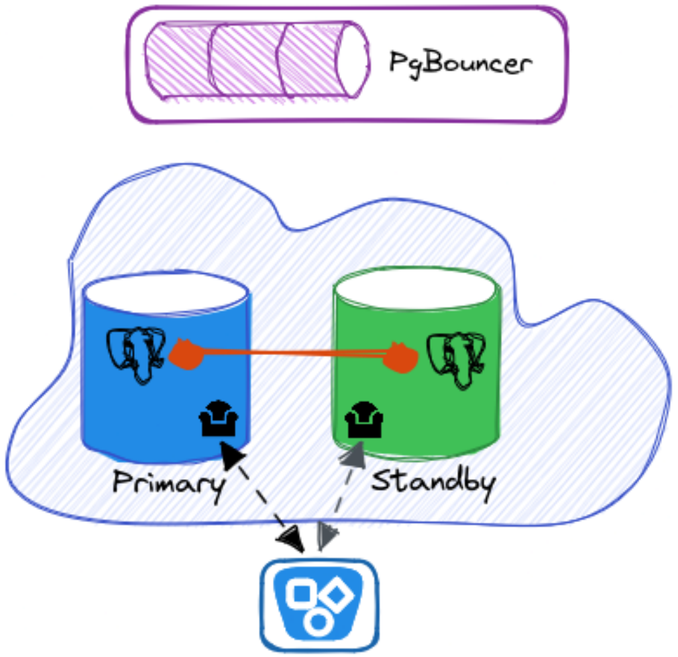
Combining both tasks

1. 🟢🔴 shutdown standby
2. 🟢🔴 enable page checksums
3. 🟢🟢 bring standby back on
4. 🟢🟢 make sure standby catches up with the primary replication state
5. 🔴🟢 shutdown primary
6. 🔴🔴 promote and then stop the standby
7. 🔴🔴 run `pg_upgrade --link` on the new primary
8. 🔴🟢 bring new primary back on
9. 🔴🟢 take a backup
10. 🟢🟢 reconstruct old primary as new standby

Minimizing downtime

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

Implementation details



https://github.com/pgstef/vagrant-playgrounds/blob/main/pgbouncer

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

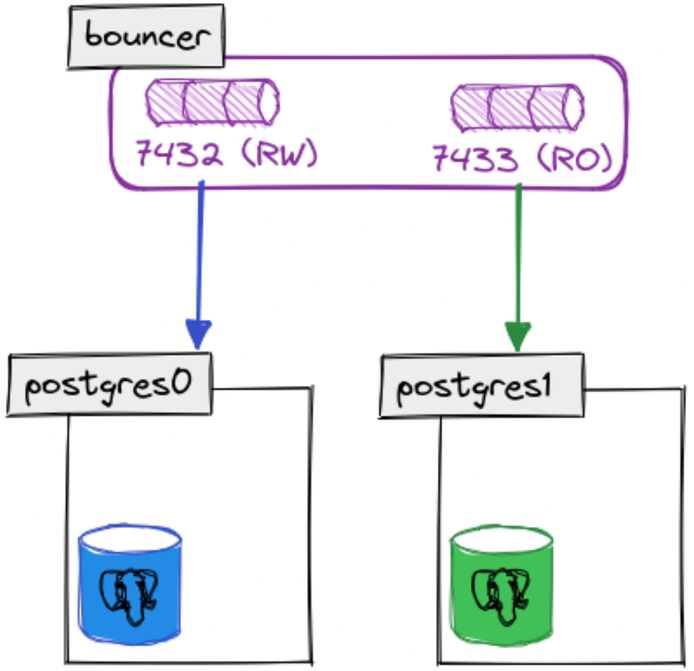
`/etc/pgbackrest.conf`

```
[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/15/main
recovery-option=primary_conninfo=host=postgres1 port=5432 user=replicator
```

Step-by-step example

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

Connections state (initial)

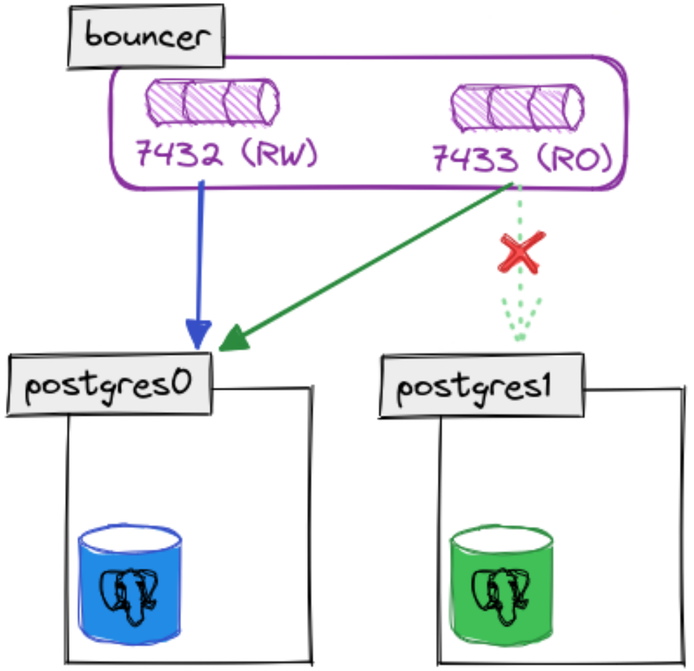


Checksums enabling on standby (1)

- redirect the traffic (SQL queries) 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;"
```

Connections state



Checksums enabling on standby (2)

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

```
$ /usr/lib/postgresql/15/bin/pg_controldata -D /var/lib/postgresql/15/main \
    |grep -E '(Database cluster state)|(REDO location)|(page checksum)'
Database cluster state:               shut down in recovery
Latest checkpoint's REDO location:    0/80000D8
Data page checksum version:           0
```

Checksums enabling on standby (3)

- enable the checksums using `pg_checksums`

```
$ /usr/lib/postgresql/15/bin/pg_checksums \
    -D /var/lib/postgresql/15/main --enable --progress
29/29 MB (100%) computed
Checksum operation completed
Files scanned:   1242
Blocks scanned:  3739
Files written:  1024
Blocks written: 3739
pg_checksums: syncing data directory
pg_checksums: updating control file
Checksums enabled in cluster
```

Checksums enabling on standby (4)

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

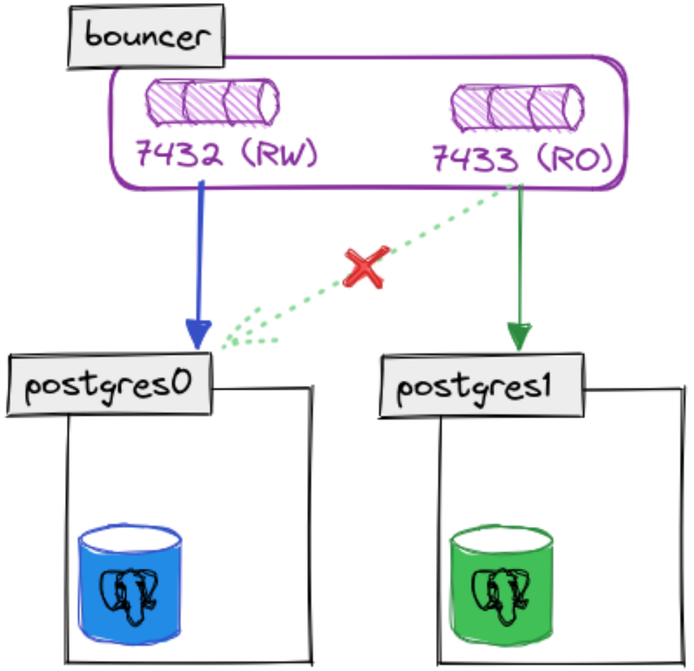
```
$ psql -xtc "SELECT status,sender_host FROM pg_stat_wal_receiver;"
status                | streaming
sender_host           | postgres0

$ psql -Axc 'show data_checksums'
data_checksums|on
```

Checksums enabling on standby (5)

- resume traffic 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;"
```

Connections state



Promote standby (1)

- pause traffic to the primary and standby

```
$ psql -U pgbouncer -p 10001 -c "PAUSE;"
$ psql -U pgbouncer -p 10002 -c "PAUSE;"
$ psql -U pgbouncer -p 10101 -c "PAUSE;"
$ psql -U pgbouncer -p 10102 -c "PAUSE;"
```

Promote standby (2)

- stop the primary
- make sure it was stopped properly

```
$ /usr/lib/postgresql/15/bin/pg_controldata -D /var/lib/postgresql/15/main \
    |grep -E '(Database cluster state)|(REDO location)|(page checksum)'
Database cluster state:               shut down
Latest checkpoint's REDO location:    0/9000028
Data page checksum version:           0
```

Promote standby (3)

- check that standby node received all activity from primary
- then promote

```
$ psql -c "CHECKPOINT;"
$ /usr/lib/postgresql/15/bin/pg_controldata -D /var/lib/postgresql/15/main \
    |grep -E '(Database cluster state)|(REDO location)|(page checksum)'
Database cluster state:               in archive recovery
Latest checkpoint's REDO location:    0/9000028
Data page checksum version:           1
```

```
$ psql -c "SELECT pg_promote();"
 pg_promote
------------
 t
(1 row)
```

PostgreSQL upgrade (1)

- check that the upgrade is possible

```
$ /usr/lib/postgresql/16/bin/pg_upgrade \
--old-bindir /usr/lib/postgresql/15/bin/ --new-bindir /usr/lib/postgresql/16/bin/ \
--old-datadir /var/lib/postgresql/15/main/ --new-datadir /var/lib/postgresql/16/main/ \
--old-options " -c config_file=/etc/postgresql/15/main/postgresql.conf" \
--new-options " -c config_file=/etc/postgresql/16/main/postgresql.conf" \
--new-options " -c archive_mode=off" \
--retain --jobs 4 --link --check
```

PostgreSQL upgrade (2)

- stop the running PostgreSQL cluster
- 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 (3)

- upgrade pgBackRest configuration

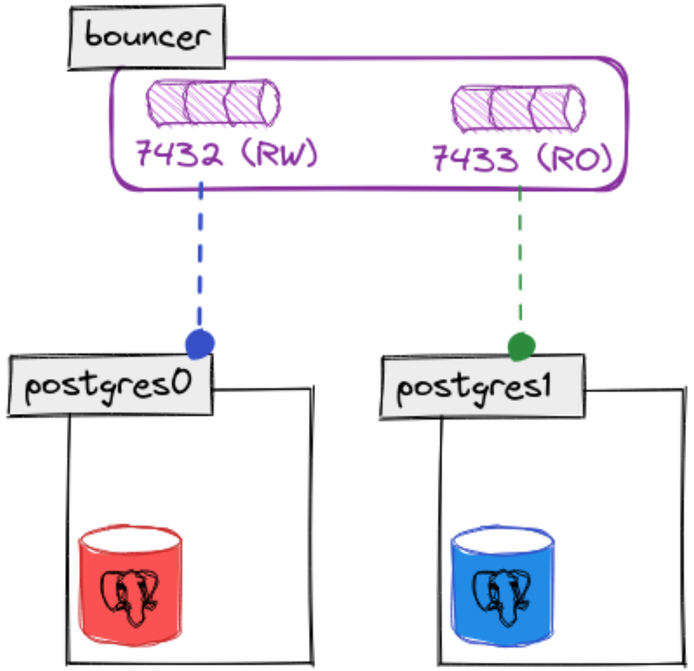
```
$ sed -i 's/\/var\/lib\/postgresql\/15\/main/\/var\/lib\/postgresql\/16\/main/g' \
    /etc/pgbackrest.conf
$ pgbackrest --stanza=mycluster --no-online stanza-upgrade
```

PostgreSQL upgrade (4)

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

```
$ /usr/lib/postgresql/16/bin/vacuumdb --all --analyze-in-stages
```

Connections state



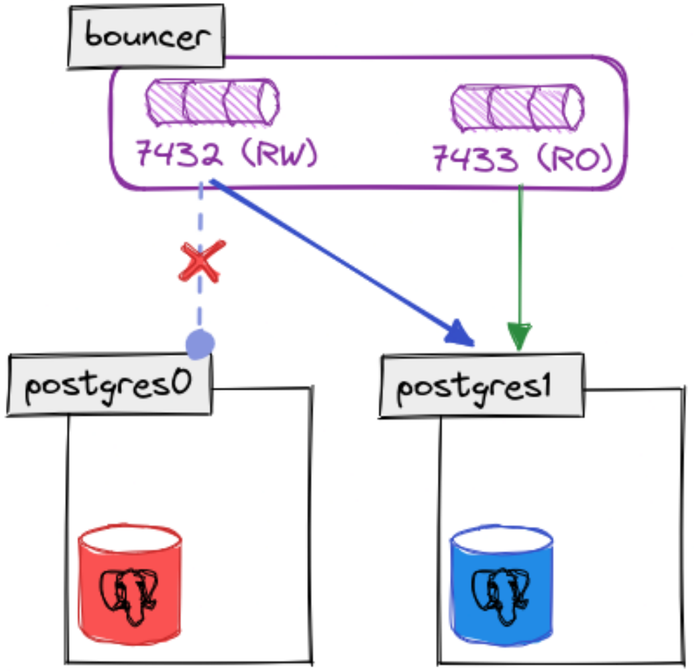
PostgreSQL upgrade (5)

- resume traffic to new primary

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

```
$ sed -i 's/postgres0/postgres1/g' /etc/pgbouncer/pgbouncer-primary-10001.ini
$ sed -i 's/postgres0/postgres1/g' /etc/pgbouncer/pgbouncer-primary-10002.ini
$ psql -U pgbouncer -p 10001 -c "RELOAD;"
$ psql -U pgbouncer -p 10002 -c "RELOAD;"
$ psql -U pgbouncer -p 10001 -c "RESUME;"
$ psql -U pgbouncer -p 10002 -c "RESUME;"
```

Connections state



PostgreSQL upgrade (6)

- make a fresh backup

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

PostgreSQL upgrade (7)

- 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/15/main'
$ sudo apt-get remove -y postgresql-15
```

Reconstruct standby (1)

- re-sync the old primary as new standby

```
$ sed -i 's/\/var\/lib\/postgresql\/15\/main/\/var\/lib\/postgresql\/16\/main/g' \
    /etc/pgbackrest.conf
$ pgbackrest --stanza=mycluster --delta --process-max=4 --type=standby restore
```

Reconstruct standby (2)

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

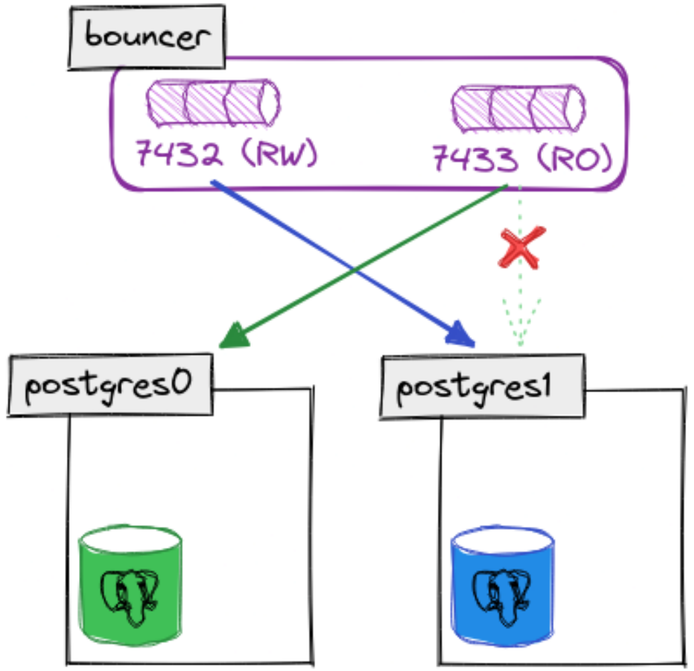
```
$ psql -xtc "SELECT status,sender_host FROM pg_stat_wal_receiver;"
status                | streaming
sender_host           | postgres1

$ psql -Axc 'show data_checksums'
data_checksums|on
```

Reconstruct standby (3)

- resume traffic back to the new standby

```
$ 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 test -d testdb -p 10101 -c "SELECT pg_is_in_recovery();"
$ psql -U test -d testdb -p 10102 -c "SELECT pg_is_in_recovery();"
```

Connections state (final)



Optional

- switch-over primary/standby roles to get back to initial state

Conclusion

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

Questions?

Thank you for your attention!