Microsoft

PGConf India

pg_upgrade like a boss!

Alexander Kukushkin

PGConf India 2022, Bengaluru

2024-02-29

About me

Alexander Kukushkin

Principal Software Engineer @Microsoft

The Patroni guy

akukushkin@microsoft.com

Twitter: @cyberdemn

Agenda

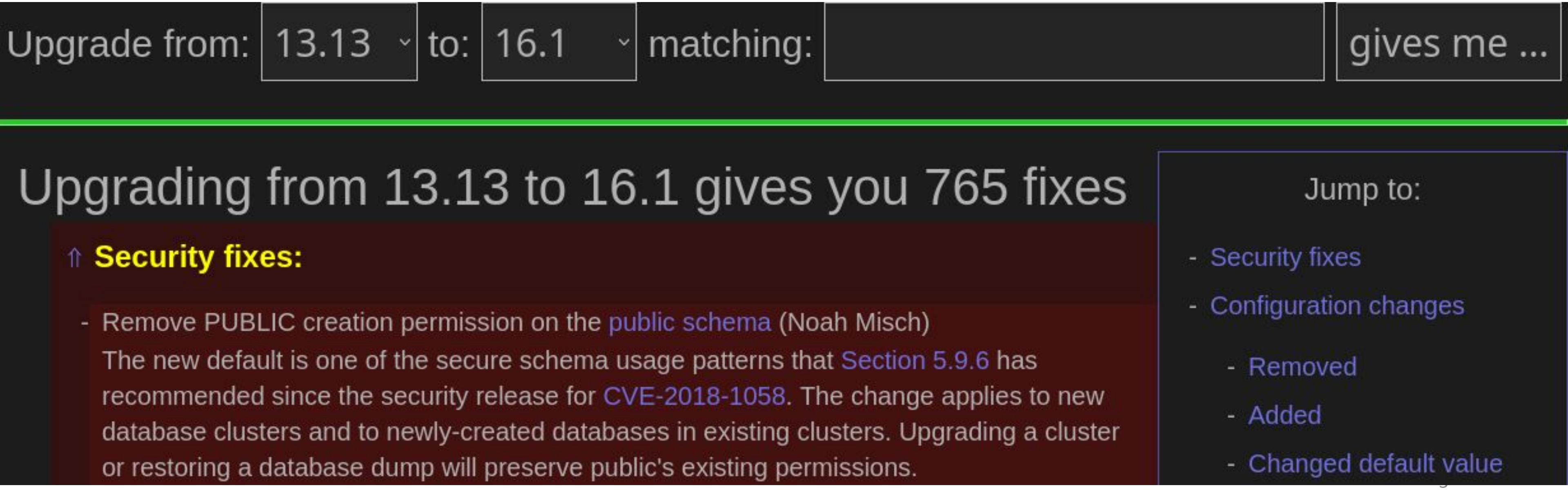
- Why upgrade?
- Types of upgrades
- pg_upgrade
- Upgrading HA setups
- Conclusion

Why upgrade?

- Security fixes
- Bugfixes
- Performance improvements
- New features

Why upgrade!

https://why-upgrade.depesz.com/show?from=13.13&to=16.1



Upgrade from: 13.13 to: 16.1 matching: gives me ...

Upgrading from 13.13 to 16.1 gives you 765 fixes

⇑ **Security fixes:**

- Remove PUBLIC creation permission on the public schema (Noah Misch)
  The new default is one of the secure schema usage patterns that Section 5.9.6 has recommended since the security release for CVE-2018-1058. The change applies to new database clusters and to newly-created databases in existing clusters. Upgrading a cluster or restoring a database dump will preserve public's existing permissions.

Jump to:

- Security fixes
- Configuration changes
  - Removed
  - Added
  - Changed default value

Versioning policy

- $major.$minor
    - 16.2, 15.6, 14.11, 13.14, 12.18
- Major releases every year
- Minor releases every quarter
- Read more about policy and release schedule

Types of upgrades

- Minor upgrade
  - 16.**0** -> 16.**2**
- Major upgrade
  - **13**.7 -> **16**.2

Minor upgrade

- 16.1 -> 16.2
- Read release notes!
  - sometimes standby needs to be upgraded first!
- Install new binaries
- Restart Postgres
- **For minor releases, *the community considers not upgrading to be riskier than upgrading!***

Major upgrades

type	downtime	resources	complexity	risk
dump/restore	high, depends on **DB size**	double (disk space)	low	low
pg_upgrade --copy	high, depends on **DB size**	double (disk space)	high	low
pg_upgrade --link	depends on the **number of objects in DB**, usually below one minute	low	high	high
pg_upgrade --clone	depends on the **number of objects in DB**, usually below one minute	low	high	low
Logical replication	sub-second	double	high	medium

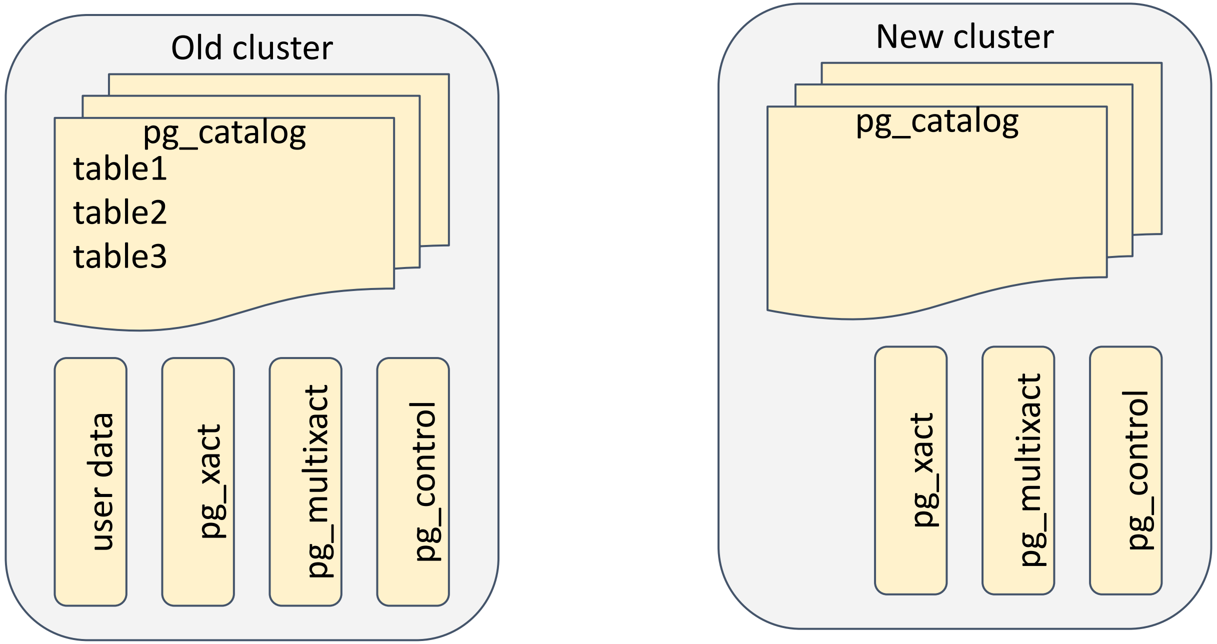
pg_upgrade --link vs --clone

- Old and new PGDATA must be located on the same filesystem
- --link
    - uses hardlinks
- --clone
    - clones files, safer than --link
    - doesn’t work with **rsync** method for upgrading standbys

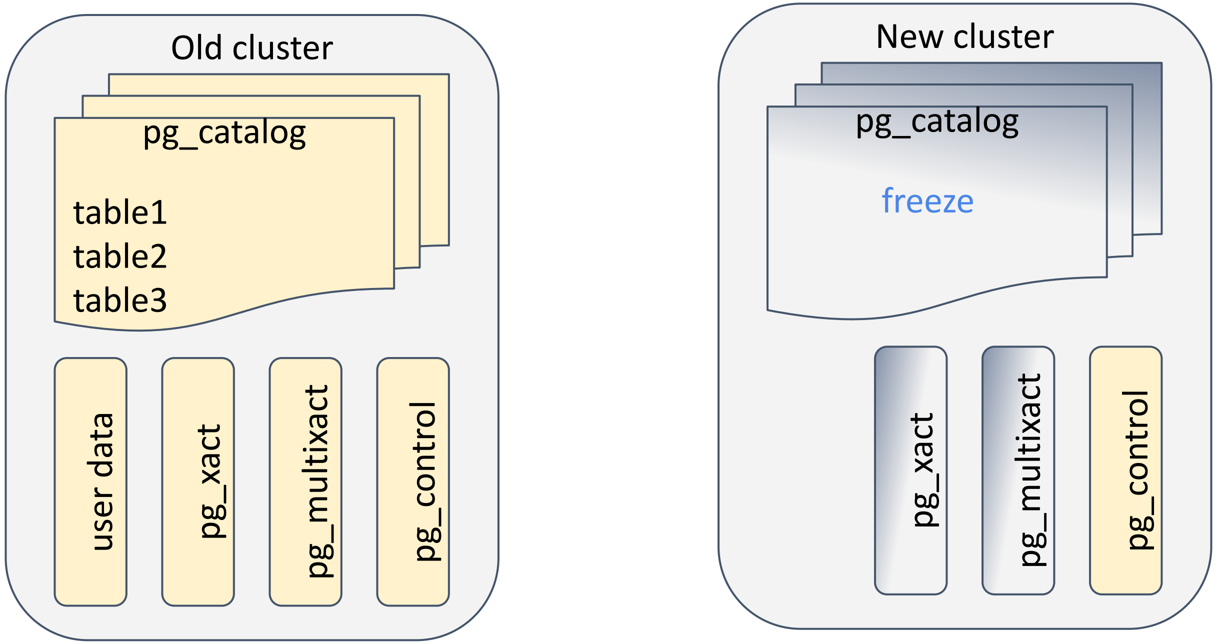
pg_upgrade workflow

1. install new major binaries
2. initdb – initialize the new cluster
3. shut down the old cluster
4. run pg_upgrade
5. start the new cluster

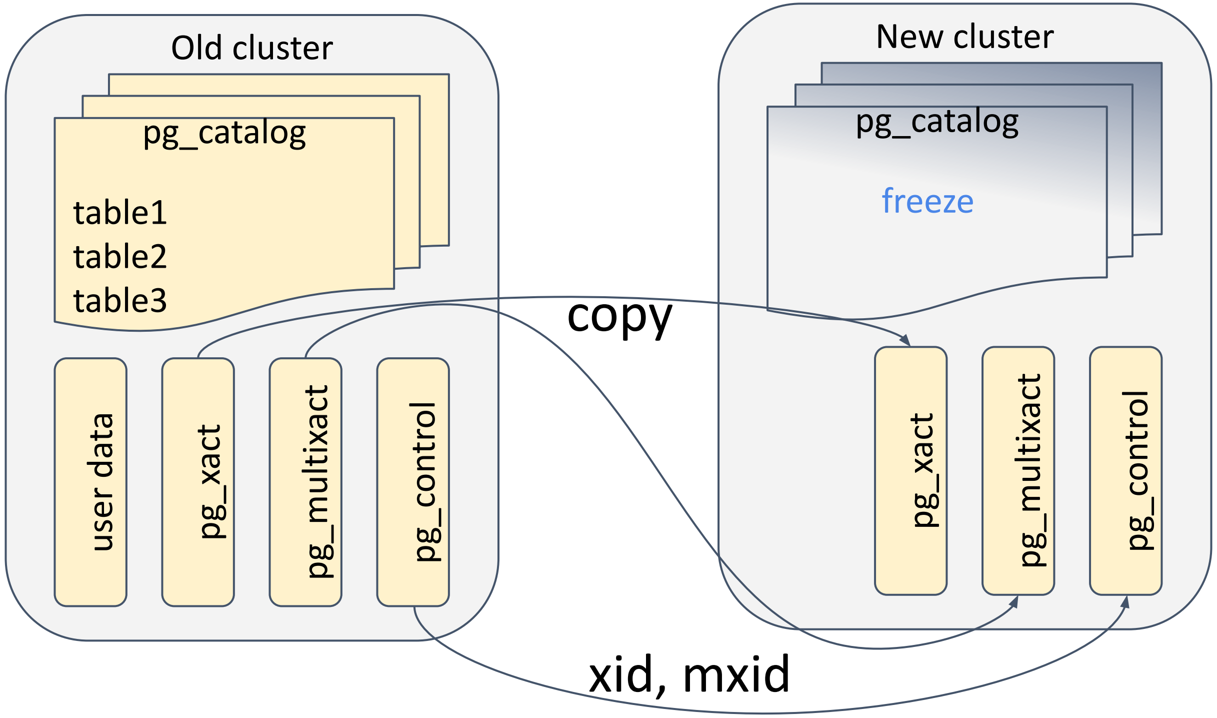
How pg_upgrade works: initial state



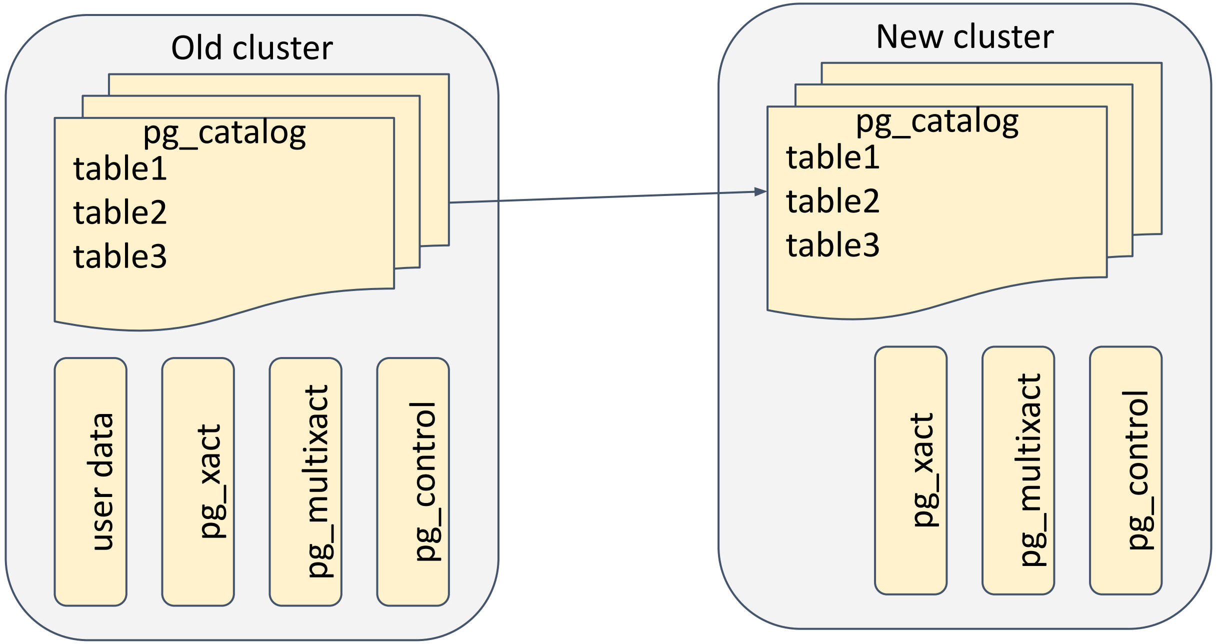
Freeze


Old cluster
pg_catalog
table1
table2
table3
user data
pg_xact
pg_multixact
pg_control
New cluster
pg_catalog
freeze
pg_xact
pg_multixact
pg_control

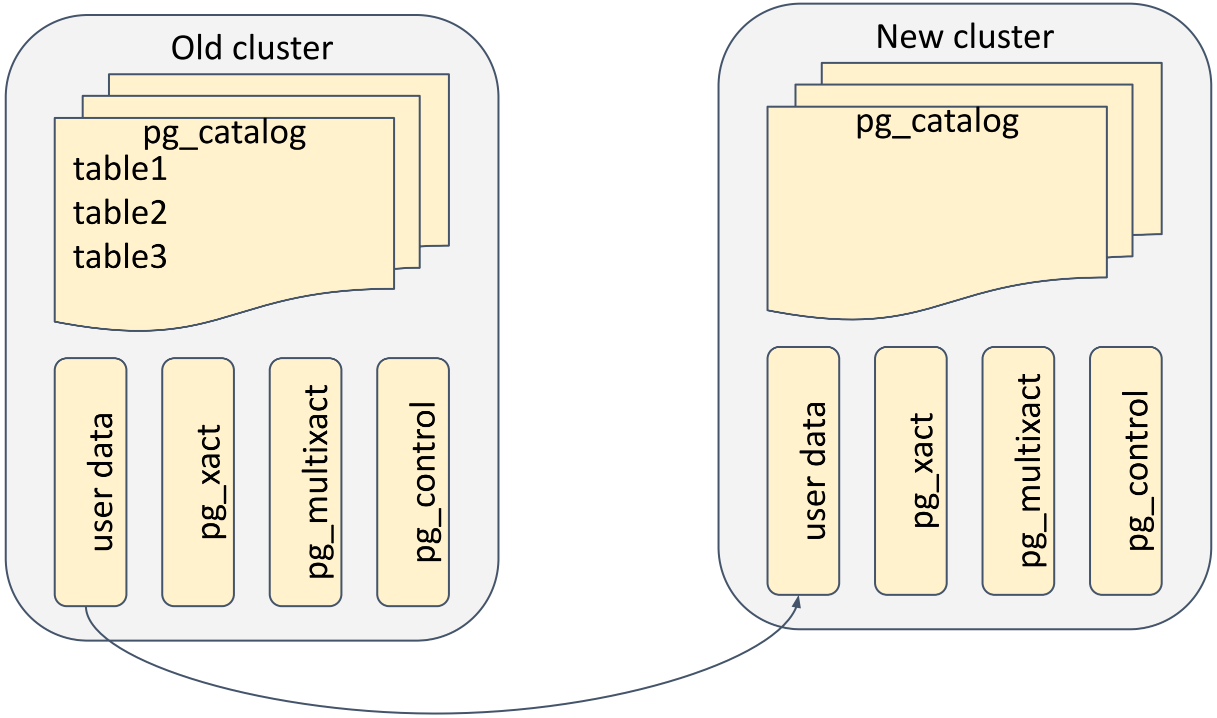
Copy clog and multixact



dump/restore schema



Copy/clone/relink relation files



Before major upgrade

- read release notes (including intermediate versions)!
  - incompatibilities must be addressed before pg_upgrade
- try pg_upgrade --check
  - if there are any problems reported - fix them
  - it can’t find everything, but improves every major release
- make a backup (pgBackRest, wal-g, barman)
- test!
  - backup/restore
  - try to upgrade restored backup

initdb

- new cluster must be initialized with the same **--locale**, **--encoding**, **--data-checksums**, and **--wal-segsize**
  - **SHOW lc_collate**;
  - **SHOW server_encoding**;
  - **SHOW data_checksums**;
  - **SHOW wal_segment_size**;

Extensions

- pg_upgrade keeps old versions of extensions
  - extension version must be available for old and new major version
  - update extensions before and/or after pg_upgrade
- some extensions need special care (pre/post upgrade)
  - Citus
  - PostGIS
- some extensions can’t be upgraded
  - pg_repack

pg_upgrade --check – false positives

```
CREATE FUNCTION test() RETURNS SETOF pg_stat_activity
LANGUAGE SQL SECURITY DEFINER
AS $$ SELECT * FROM pg_stat_activity; $$;

CREATE VIEW test AS SELECT * FROM test();
```

- pg_upgrade --check – *Clusters are compatible*
- but, pg_upgrade – *failure*
- strategy:
  - restore from the backup and run pg_upgrade
  - if fails - fix problems
  - repeat

Minimizing downtime

- Do all preparations *before* calling pg_upgrade (and stopping the primary)
  - cleanups, initdb, etc
- Manually run a few times CHECKPOINT
  - Speeds up **pg_ctl stop -m fast**
- Use pg_upgrade **--clone** or **--link**
  - New and old PGDATA must be located on the same filesystem
    - /pgdata/12 # old PGDATA
    - /pgdata/16 # new PGDATA
- Use **--jobs=N**
  - parallel schema dump/restore and relinking

After pg_upgrade

- rebuild table statistics
  - vacuumdb --all --analyze-in-stages
- restore dropped objects
- trigger creation of new basebackup!

Analyze in stages

```
SET default_statistics_target = 1;
ANALYZE;
/* at this point, usually, we are good enough to allow connections */

SET default_statistics_target = 10;
ANALYZE;
SET default_statistics_target = 100;
ANALYZE;
```

Beware non default statistics target set on columns!

```
postgres=# \d+ test
                                                   TABLE "public.test"
 COLUMN |  TYPE  | Collation | NULLABLE | DEFAULT | Storage  | Compression | Stats target | Description
--------+--------+-----------+----------+---------+----------+-------------+--------------+-------------
 id     | BIGINT |           | NOT NULL |         | plain    |             |              |
 name   | text   |           |          |         | extended |             | 1000         |
Indexes:
    "test_pkey" PRIMARY KEY, btree (id)
Access method: heap
```

- Breaks --analyze-in-stages
  - ANALYZE on **test** table will always read 300***1000** tuples instead of 300*default_statistics_target
  - Even the first stage is veeeery slow

Solution

1.
   ```
   ALTER TABLE test ALTER COLUMN name
   SET STATISTICS -1;  /* reset custom setting */
   ```

2.
   ```
   vacuumdb --all --analyze-in-stages
   ```

3.
   ```
   ALTER TABLE test ALTER COLUMN name
   SET STATISTICS 1000;  /* restore custom setting */
   ```

4.
   ```
   ANALYZE test;  /* rebuild statistics with custom setting */
   ```

Speed up vacuumdb --all --analyze-in-stages

- Use **--jobs N** parameter for vacuumdb
- But, parallelism is maybe not what you think!
  - Sequentially goes over databases in the cluster and does ANALYZE on N tables in parallel
  - What if we have 16 database with 1 huge table in each?
  - Run multiple vacuumdb -d $DB instead of a single vacuumdb --all

Upgrading HA setups

- Rebuild standby nodes using **pg_basebackup**
  - the safest option
  - but slow, speed ~1TB/h :(

- Upgrade standbys with rsync

Upgrading standbys with rsync

- Described in Postgres docs

- requires **pg_upgrade --link**
- relies on the fact that **user relation data files** in primary and standby PGDATA are fully identical
  - **We have to ensure that standby is up-to-date**!

How postgres stores relations on filesystem

```
postgres=# CREATE TABLE test(id BIGINT NOT NULL PRIMARY KEY, name text);
CREATE TABLE

postgres=# INSERT INTO test SELECT i, 'test' FROM generate_series(1, 1000000) AS i;
INSERT 0 1000000

postgres=# SELECT oid, relfilenode FROM pg_class WHERE relname = 'test';
  oid  | relfilenode
-------+-------------
 16394 |       16402
(1 ROW)

$ ls -gi pg12/base/13498/16402*
40372037 -rw------- 1 akukushkin 44285952 Jan 24 09:50 pg12/base/13498/16402
40372067 -rw------- 1 akukushkin    32768 Jan 24 09:49 pg12/base/13498/16402_fsm
```

Using pg_upgrade --link

```
$ /usr/lib/postgresql/16/bin/pg_upgrade --link \
  -b /usr/lib/postgresql/12/bin \
  -B /usr/lib/postgresql/16/bin \
  -d pg12 -D pg16

...
Adding ".old" suffix to old global/pg_control                ok

If you want to start the old cluster, you will need to remove
the ".old" suffix from pg12/global/pg_control.old.
Because "link" mode was used, the old cluster cannot be safely
started once the new cluster has been started.
Linking user relation files
                                                             ok
Setting next OID for new cluster                             ok
Sync data directory to disk                                  ok
Creating script to delete old cluster                        ok
Checking for extension updates                               ok

Upgrade Complete
```

Checking linked files

```
$ ls -gi pg12/base/13498/16402*
40372037 -rw------- 2 akukushkin 44285952 Jan 24 09:50 pg12/base/13498/16402
40372067 -rw------- 2 akukushkin    32768 Jan 24 09:49 pg12/base/13498/16402_fsm


$ ls -gir pg16/base/13498/16402*
40372067 -rw------- 2 akukushkin    32768 Jan 24 09:49 pg16/base/13498/16402_fsm
40372037 -rw------- 2 akukushkin 44285952 Jan 24 09:50 pg16/base/13498/16402
```

* Inodes in the new PGDATA remain the same.

Upgrade standby with rsync

```
$ rsync \
  --archive \           # same as -r -l -p -t -g -o -D, see below
  --delete \            # delete extraneous files from dest dirs
  --hard-links \        # look for hard-linked files in the source and link to corresponding files on the destination!
  --size-only \         # copy files only if size doesn’t match (ignore mtime and content!)
  --no-inc-recursive \ # scan the full file list before transfering files
  /var/lib/postgres/pgdata/pg12 \
  /var/lib/postgres/pgdata/pg16 \
  standby.example.com:/var/lib/postgres/pgdata
# -r – recursive
# -l – copy symlinks as symlinks
# -p – preserve permissions
# -t – preserve mtime
# -g – preserve group
# -o – preserve owner
# -D – preserve devices and special files
```

Standby after rsync

Standby before rsync:

```
/var/lib/postgres/pgdata/pg12/
/var/lib/postgres/pgdata/pg12/base/
/var/lib/postgres/pgdata/pg12/base/1/
/var/lib/postgres/pgdata/pg12/base/1/112
…
/var/lib/postgres/pgdata/pg12/13498/16402
…
/var/lib/postgres/pgdata/pg16/ # doesn’t exist
```

Standby after rsync:

```
/var/lib/postgres/pgdata/pg12/
/var/lib/postgres/pgdata/pg12/base/
/var/lib/postgres/pgdata/pg12/base/1/
/var/lib/postgres/pgdata/pg12/base/1/112
…
/var/lib/postgres/pgdata/pg12/13498/16402
…
/var/lib/postgres/pgdata/pg16/
/var/lib/postgres/pgdata/pg16/base/
/var/lib/postgres/pgdata/pg16/base/1/
/var/lib/postgres/pgdata/pg16/base/1/112
…
/var/lib/postgres/pgdata/pg16/13498/16402
…
```

copied from the primary

hardlink, no copy!

HA major upgrade - full procedure

- preparations mainly as for normal pg_upgrade
  - truncate unlogged/temp tables (to avoid copying them to standby nodes by rsync)
- make sure that standby nodes are not lagging!
- stop the primary (manual CHECKPOINT + **pg_ctl stop -m fast**)
- get **Latest checkpoint location** from pg_controldata output
  - make sure that standby applied WAL up to checkpoint LSN!
- run pg_upgrade --link …

HA major upgrade - full procedure (continue)

- Don’t start postgres on primary after pg_upgrade until rsync finished!
- Stop standby nodes (could be done in parallel with pg_upgrade)
- run rsync for all standby nodes
- start postgres on the primary
- trigger statistics rebuild on the primary:
    - vacuumdb --all --analyze-in-stages
- restore dropped objects (if needed), update extensions, etc
- trigger creation of new basebackup

HA major upgrade - full procedure (continue)

- update config files on standby nodes (they are rsynced from the primary)
  - pg_hba.conf
  - postgresql*.conf: (**primary_conninfo** & co)
  - **standby.signal)**
- start postgres on standby nodes
- verify that replication works
- remove old PGDATA on all nodes (if everything is fine)

Tricks with rsync

- usually rsync works via remote shell (ssh)
- in the cloud (containers) configuring ssh and distributing keys just for major upgrade is too much
- we can use rsync daemon instead
  - run daemon on the primary, with read-only access
  - clients on standby nodes
- rsync-ssl – wrapper to add ssl support
  - we may use the same certificates as for postgres

What if something goes wrong?

- **pg_upgrade failed** - just start the old cluster
    - sometimes requires removing **.old** suffix from **global/pg_control.old**
- **rsync failed** - rebuild standby nodes using pg_basebackup
- as a precaution keep one standby intact

Downtime

- downtime of pg_upgrade --link + rsync **depends only on the number of objects** in the cluster and **doesn’t depend on the total size of data**
- for **small and medium size** clusters it’s possible to do major upgrade with only **10s-20s** of downtime (excluding statistics rebuild)
- **waste majority** of clusters could be upgraded with downtime **less than 1 minute**.

Unsolved (yet) problems

- replication slots are lost (solved in v17)
- subscriptions are preserved, but not reactivated
- table statistics rebuild may take significantly longer than major upgrade
    - WIP: Statistics Import and Export

Conclusion

- pg_upgrade --link + rsync is a fast method of major upgrades with a small downtime
    - no additional resources required
- There are some problems, but community works on solving them

- always do backups and test recovery procedures!

Questions?