Zero-Downtime Upgrades: PostgreSQL and OS/glibc at Global Scale

GitLab

GitLab

Alexander Sosna

Senior Database Reliability Engineer

GitLab Copyright

Agenda

This talk will showcase:

- How we execute PostgreSQL and OS upgrades at GitLab, with **zero downtime**.

By answering these questions:

- PostgreSQL Upgrades - How do they work, and why are they hard?
- OS Upgrades - How do they work, and why are they hard?
- What did we do to minimize impact to our users?

To fit the time slot, some aspects are simplified, details and code in the linked resources!

GitLab Copyright

Why are PostgreSQL Major Upgrades hard?

- Major releases (can) change the layout of system tables
- Data files can not be used by newer versions
- Rewriting of system tables and metadata is necessary
- Helping structures like indexes might require a rebuild
- Depending on data size and complexity this can take significant time

GitLab Copyright

Upgrade Method - pg_dumpall (default)

1. maintenance mode (DB becomes RO)
User impact starts

2. physical to logical (binary to SQL*)

3. logical to physical (SQL* to binary)

4. create indexes collect statistic

GitLab Copyright

x86

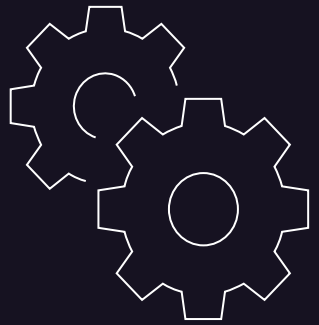
Upgrade Method - pg_dumpall

- **Safest method** available
- Also able to upgrade
  - OS/glibc
  - Hardware architecture, e.g. x86 ⇒ RISC-V
- Some data types like *jsonb* get validated
- Requires **downtime based on data and indexes**
  - Hard to provide simple estimate: our *~40 TiB DB will take > 24h*
  - *You can easily try it out and measure to get exact timing*
- No quick rollback after upgrade!

If this fulfills your needs, it’s the safest option! Don’t look any further!

GitLab Copyright

Upgrade Method - pg_upgrade

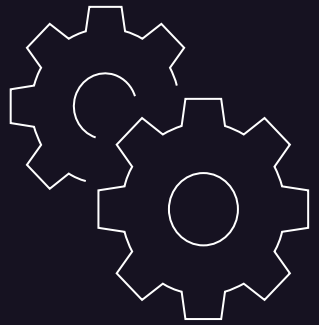


1.
Maintenance mode
(offline / RO with standby)
User impact starts

2.
In-place upgrading
binary data

GitLab Copyright

Upgrade Method - pg_upgrade



1.
Maintenance mode
(offline / RO with standby)
User impact starts

2.
In-place upgrading
binary data

GitLab Copyright

Upgrade Method - pg_upgrade

- Quite simple
- Reasonable fast
  - Additional operations like a reindex or tests can take longer!
- Reasonable safe
- No quick rollback after upgrade!
- When I joined GitLab, we used it as well
  - Due to mandatory QA tests total downtime was >4h per upgrade
  - Upgrades where avoided due to downtime

If this fulfills your needs, it’s a safe and simple option! Don’t look any further!

GitLab Copyright

Why can’t we use a boring solution for GitLab.com?

GitLab Copyright

Why can’t we use a boring solution for GitLab.com?

- GitLab.com is a globally used SaaS offering
  - > 50 million users around the world
  - > 2,500 team members, all-remote and globally distributed (>65 countries)
  - > 1 Million SQL queries per second on PostgreSQL (US working hours)
  - There is not a single minute, at witch a downtime would not impact users and team members!
  - Data Sources ir.gitlab.com, about.gitlab.com/company/team
- No budget for downtime
- We need to be able to roll back if the new DBMS does not perform

GitLab Copyright

How do you define “Zero Downtime” in SaaS?

- User requests are not handled instantaneously
- When a user presses a button it takes time before the result is shown
- We can’t go for “0 ms” downtime :)

GitLab Copyright

How do you define “Zero Downtime” in SaaS?

- **User requests are not handled instantaneously**
- **When a user presses a button it takes time before the result is shown**
- **We can’t go for “0 ms” downtime :)**

“Zero Downtime” ⇒ no user impact!

GitLab Copyright

How is GitLab measuring User Impact?

- Apdex (Application Performance Index)
  - Open standard for measuring application performance
  - Based on classifying user interactions ins
    - “satisfied”
    - “tolerating”
    - “frustrated”
  - Requires tuned thresholds to classify samples
  - Details: wikipedia.org/wiki/Apdex

$$\text{Apdex}_t = \frac{\text{SatisfiedCount} + (0.5 \cdot \text{ToleratingCount}) + (0 \cdot \text{FrustratedCount})}{\text{TotalSamples}}$$

GitLab Copyright

How do we achieve Zero Downtime?

GitLab Copyright

How do we achieve Zero Downtime?

Logical Replication

GitLab Copyright

How are we achieving Zero Downtime?

Logical Replication
(and a lot of automation)

GitLab Copyright

Logical Replication

- Unlike Streaming Replication, LR can replicate across different PG versions
- We can upgrade a clone of our production database and bring it in sync

- Does it come with restrictions?
  - Yes!
  - Watch my previous talk or read the extended slide deck
  - How we execute PG major upgrades at GitLab, with zero downtime. (PGConf.EU 2023) youtube.com/watch?v=o08kJggkovg
  - Important for this talk: Schema changes would break LR!
    - No DDL allowed: CREATE, ALTER, DROP, ...

GitLab Copyright

Logical Replication - DDL is not replicated

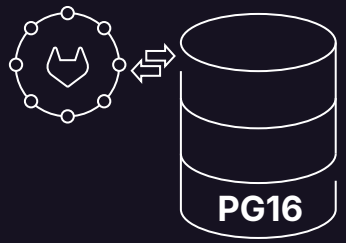
- Schema changes would break logical replication!
  - No DDL allowed: CREATE, ALTER, DROP

Our solution

- Disable all deployments, migration, and maintenance jobs creating DDL
  - GitLab features
    - Database upgrade DDL lock
    - *disallow_database_ddl_feature_flags, MR130554*
  - You need to check **YOUR** applications DDL usage!
    - Most common software will not erratically execute DDL

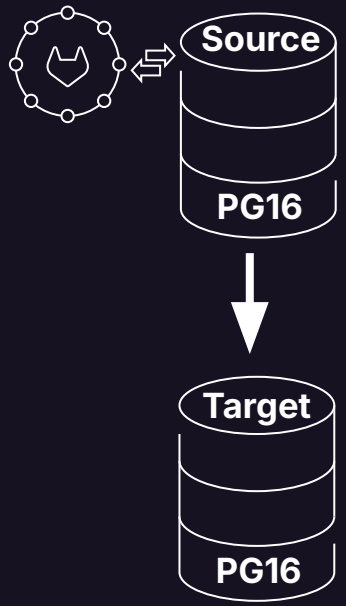
GitLab Copyright

Logical Replication + pg_upgrade



GitLab Copyright

Logical Replication + pg_upgrade



Create and sync Target

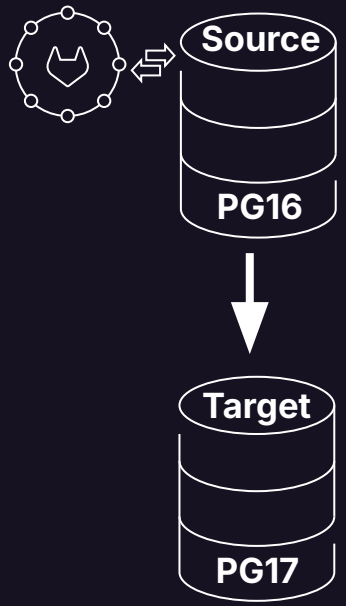
GitLab Copyright

Logical Replication + pg_upgrade

Upgrade Target
(no sync during upgrade)

GitLab Copyright

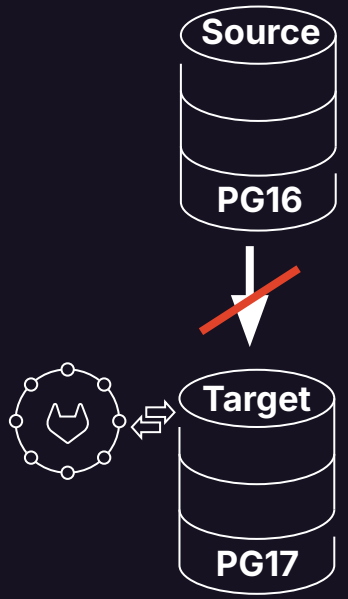
Logical Replication + pg_upgrade



Resync via LR

GitLab Copyright

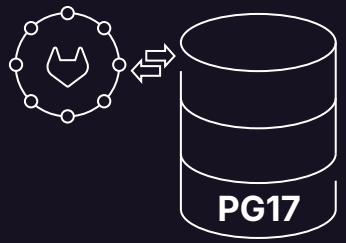
Logical Replication + pg_upgrade



Application Switchover

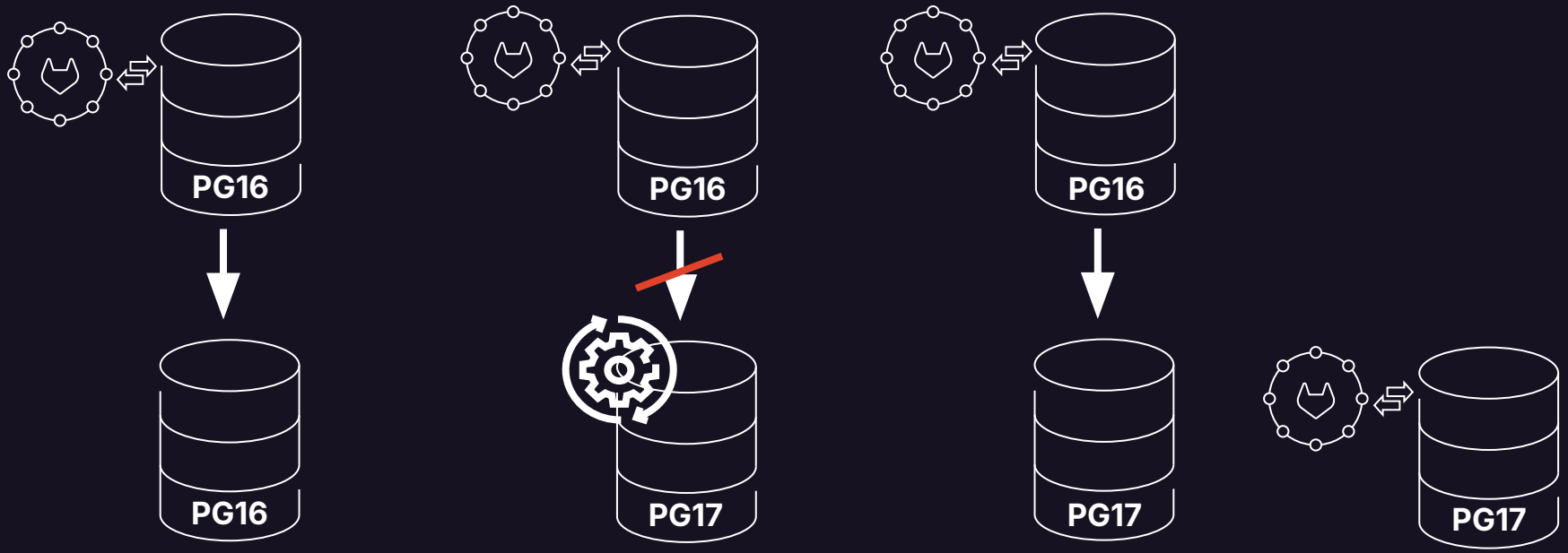
GitLab Copyright

Logical Replication + pg_upgrade



GitLab Copyright

PostgreSQL Upgrade - State 2023



1.Sync Target

2. Upgrade Target

3. Resync

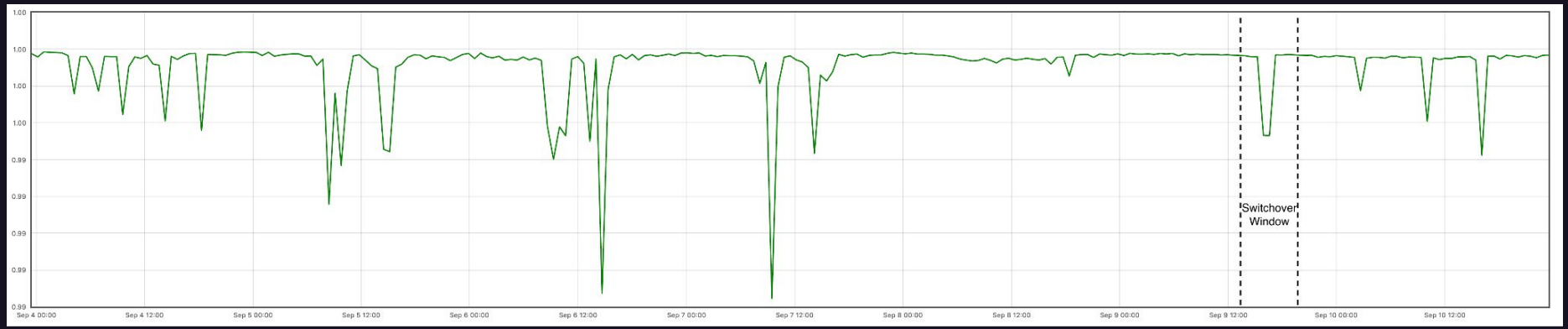
4. Switchover

GitLab Copyright

What is actual the User Impact?

GitLab Copyright

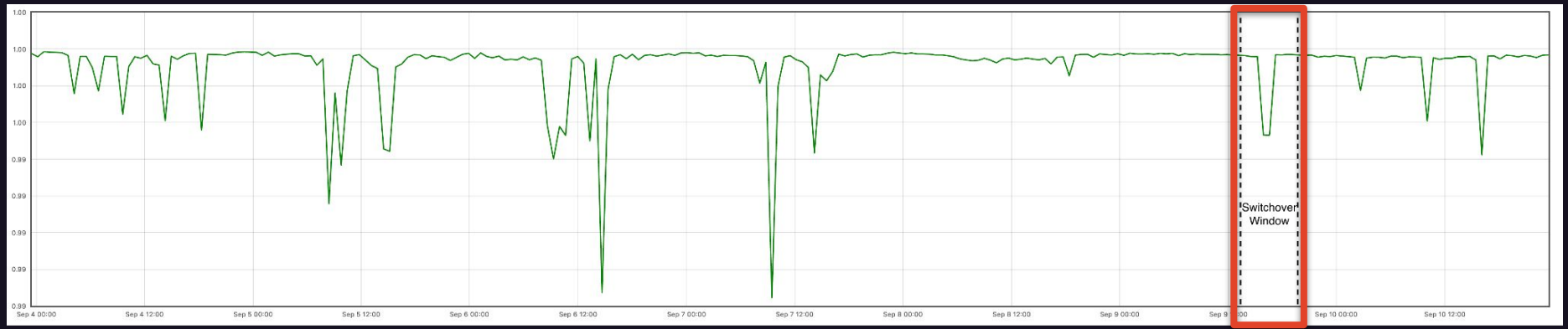
How well did we do? - Web Apdex



- Web Service Apdex - top 1% (0.99 - 1.00 nit-picking view)
- Degradation SLO: 98.8%, red line would be below this graph :D

GitLab Copyright

How well did we do? - Web Apdex



- Web Service Apdex - top 1% (0.99 - 1.00 nit-picking view)
- Degradation SLO: 98.8%, red line would be below this graph :D

GitLab Copyright

Can we improve further?

1. Switchover is a Point of no Return
    - If performance degrades or any problem occurs, we can’t go back!
    - Significant business risk!

2. This approach only upgrades PostgreSQL
    - OS or library upgrades are not handled

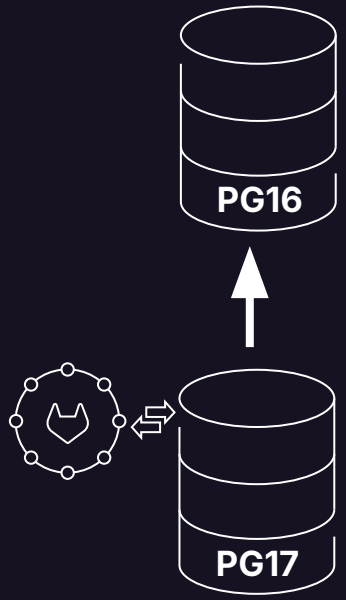
GitLab Copyright

(Re)move Point of no Return - Reverse Replication

- After the Switchover we reverse the replication
- Enables swift rollback without data loss

GitLab Copyright

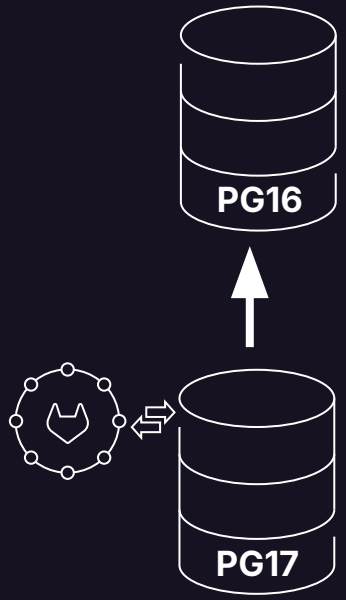
(Re)move Point of no Return - Reverse Replication



Reverse Replication

GitLab Copyright

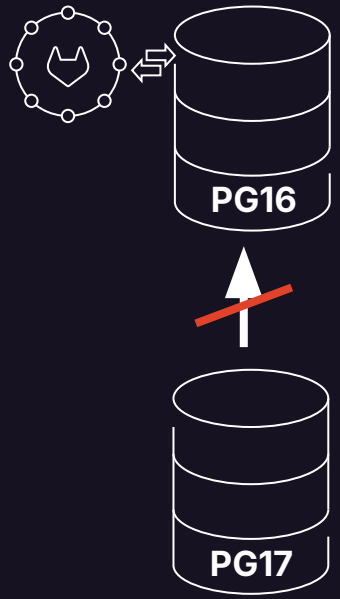
(Re)move Point of no Return - Reverse Replication



Operation and Monitoring

GitLab Copyright

(Re)move Point of no Return - Reverse Replication



Late Rollback
(optional)

GitLab Copyright

Why are OS Upgrades hard?

- When upgrading the OS, you will get a new version of glibc (GNU C Library)
  - This library defines the system-wide collation
- Collation: Set of rules that describe how strings are compared and ordered
  - `“A” < “B” < “C”`
  - `“1” < “2” < “3”`
  - `“10” < “2” OR “10” > “2”`
  - `“\” < “/” OR “/” > “\”`
- Indexes
  - Need to be used with the collation they were created with!
  - If not, data corruption can occur!

GitLab Copyright

OS Upgrade - Simple Solution

- Some data types don’t use collations and are unproblematic, e.g. INTEGER
- Rebuild all indexes (on strings) with the current collation
  - If this works for your use-case, great!
  - If you use the pg_dumpall upgrade method you get it automatically
  - For GitLab.com this would take multiple days, longer than our upgrade window

GitLab Copyright

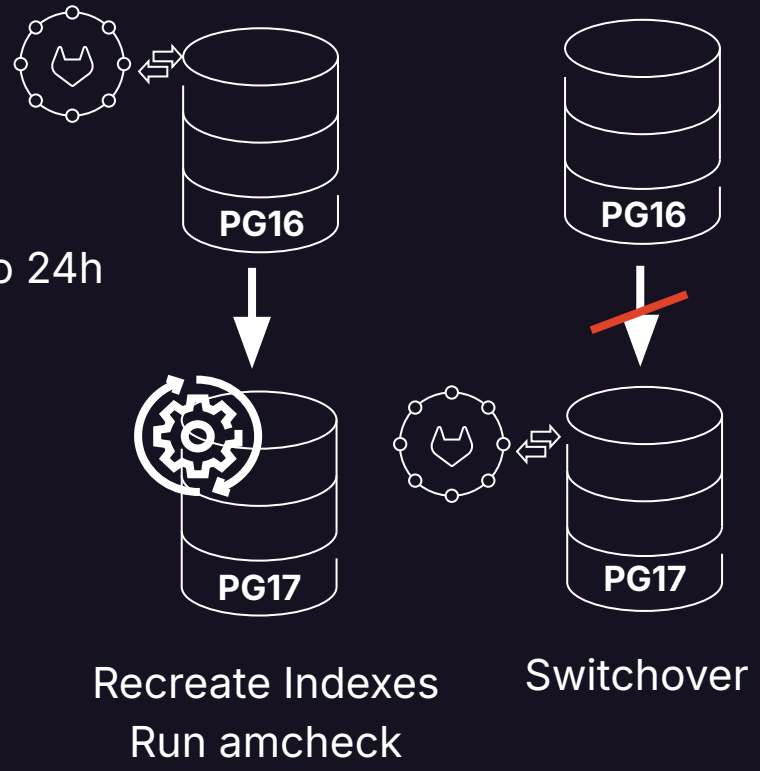
OS Upgrade - Optimized Approach

- Before we start the upgrade we automatically create a list of all indexes, where the new collation can lead to corruption. (Script based on amcheck)
  - No need to recreate non-problematic types like INTEGER
  - No need to recreate indexes only containing non-problematic data
    - Example: md5 hashes (strings)
- We recreate all listed indexes on a test system, to measure the execution time
  - If it takes longer than acceptable, we can optimize beforehand
    - Replace indexes
      - Different type
      - Multiple partial indexes
    - If non-disruptive: lazily recreate after upgrade

GitLab Copyright

OS Upgrade - Optimized Approach

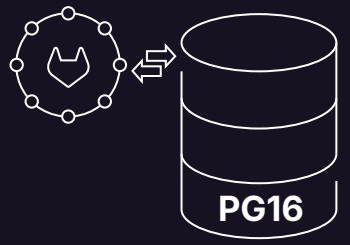
- Saturday: Upgrade
- Sunday: Switchover
- After the upgrade step we have between 12h to 24h before the Switchover to:
  - recreate all problematic indexes
  - run amcheck to verify no data corruption
  - run additional tests if necessary



GitLab Copyright

PostgreSQL and OS Upgrade

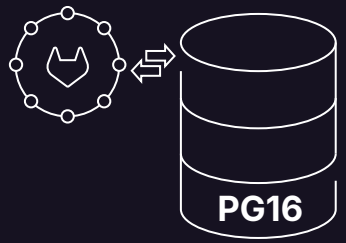
- 2025 we upgraded most of our database systems
  - PG16 ⇒ PG17
  - Ubuntu 20.04 ⇒ 22.04
- Let’s walk through one of the last upgrades



GitLab Copyright

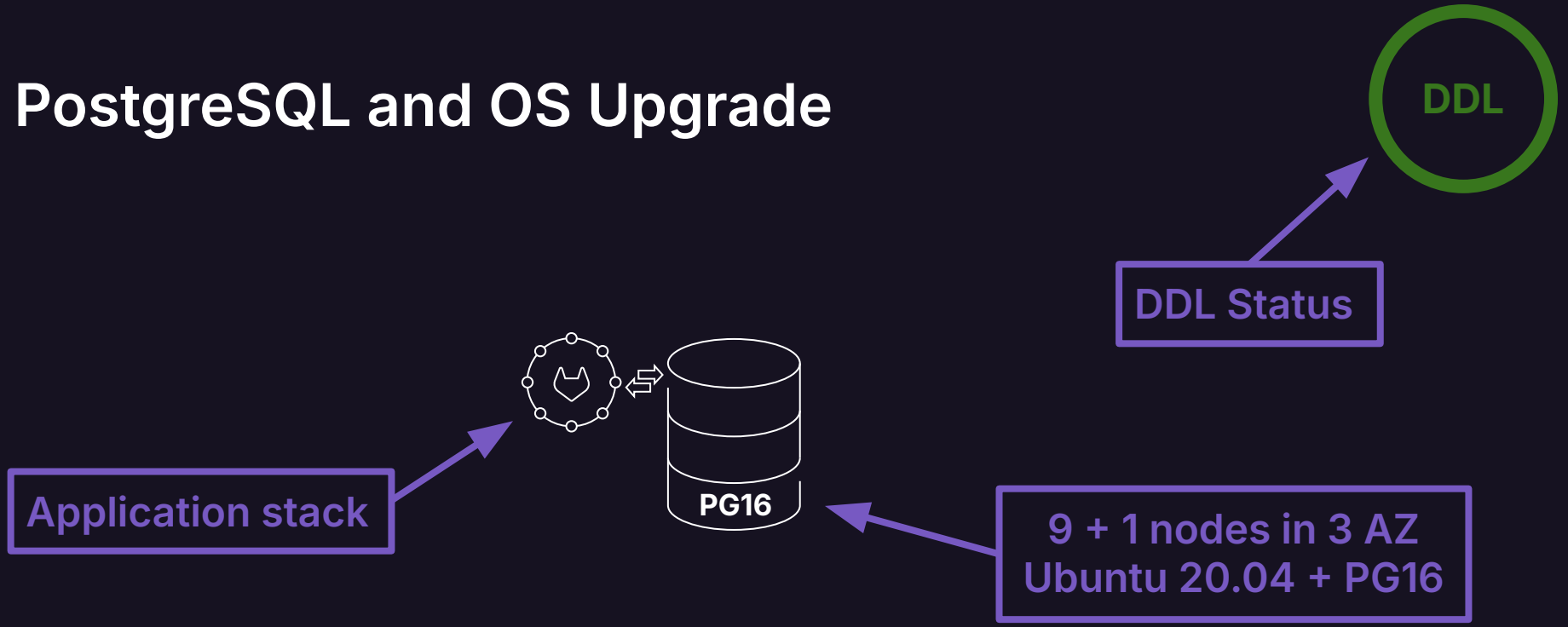
PostgreSQL and OS Upgrade

DDL



GitLab Copyright

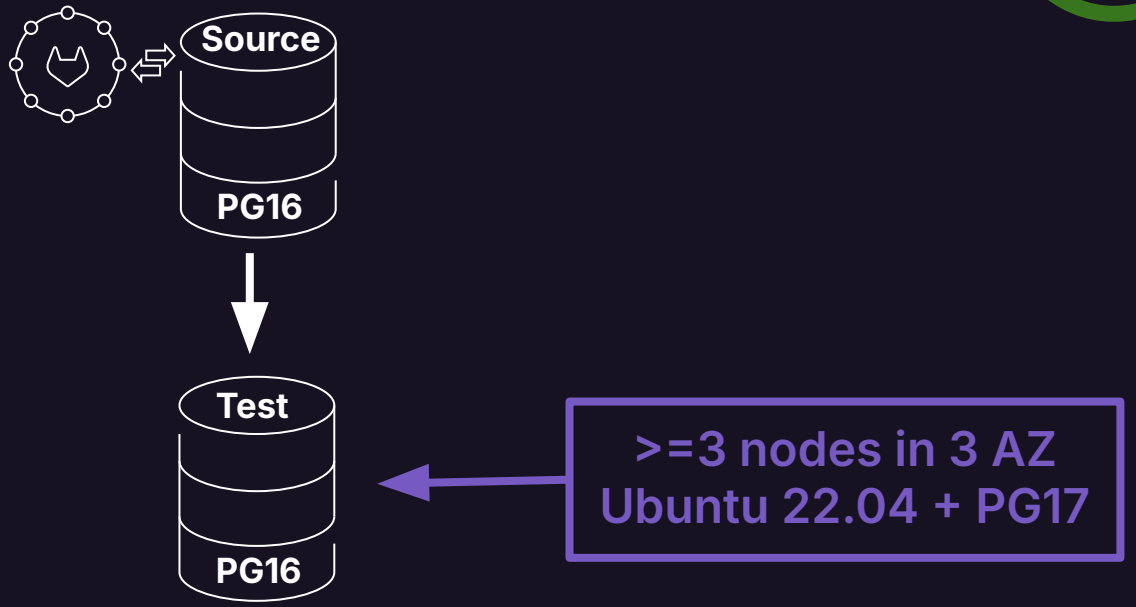
PostgreSQL and OS Upgrade



GitLab Copyright

PostgreSQL and OS Upgrade - Preparation

DDL

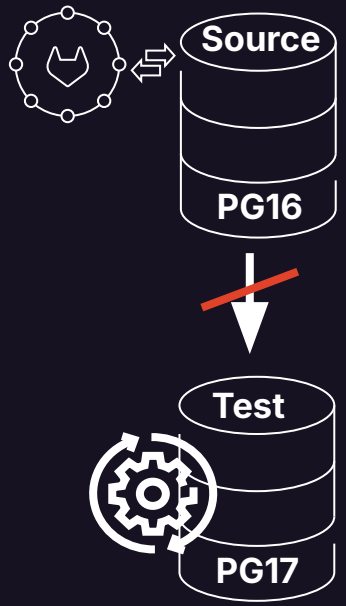


Create Test clone

GitLab Copyright

PostgreSQL and OS Upgrade - Preparation

DDL



Test upgrade

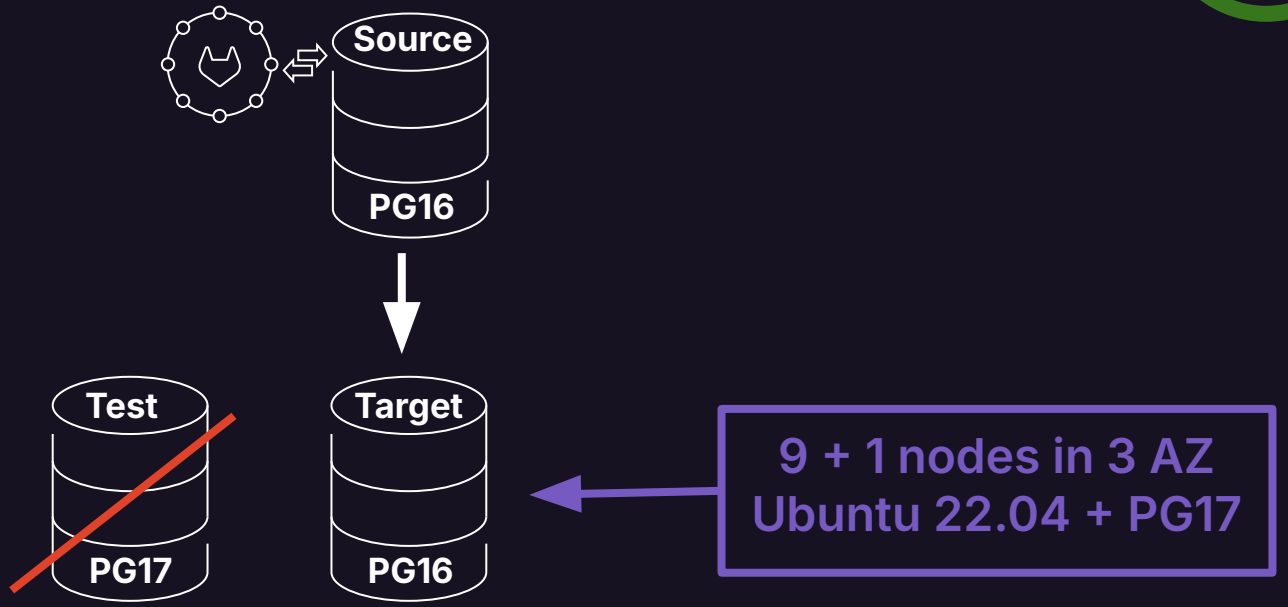
Get execution times

Get list of corrupted indexes

GitLab Copyright

PostgreSQL and OS Upgrade - Preparation

DDL



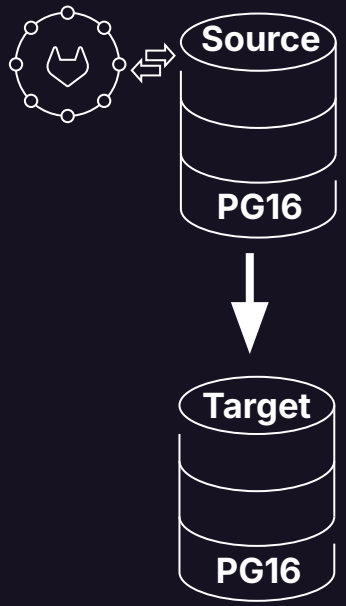
Remove Test Cluster

Create Target Cluster

GitLab Copyright

PostgreSQL and OS Upgrade - Saturday

No DDL



Switch to logical replication
(DDL would break it)

GitLab Copyright

PostgreSQL and OS Upgrade - Saturday

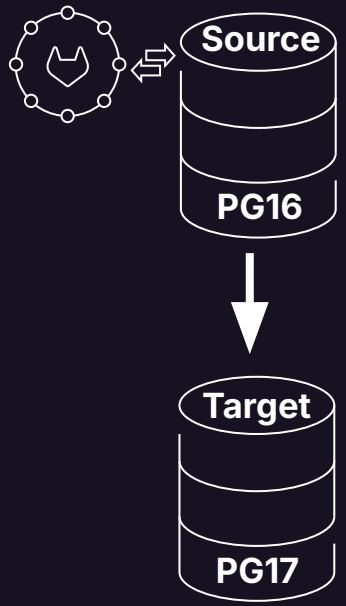
No DDL

Upgrade Target
(no sync during upgrade)

GitLab Copyright

PostgreSQL and OS Upgrade - Saturday

No DDL

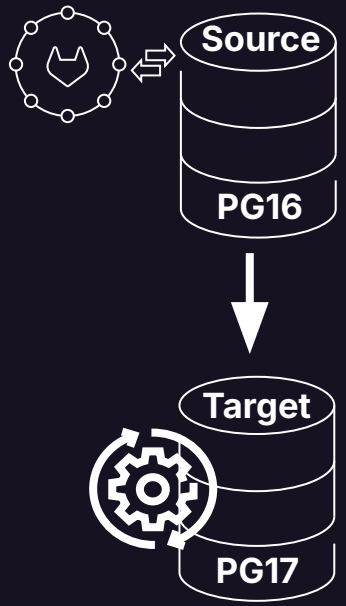


Resync

GitLab Copyright

PostgreSQL and OS Upgrade - Saturday

No DDL

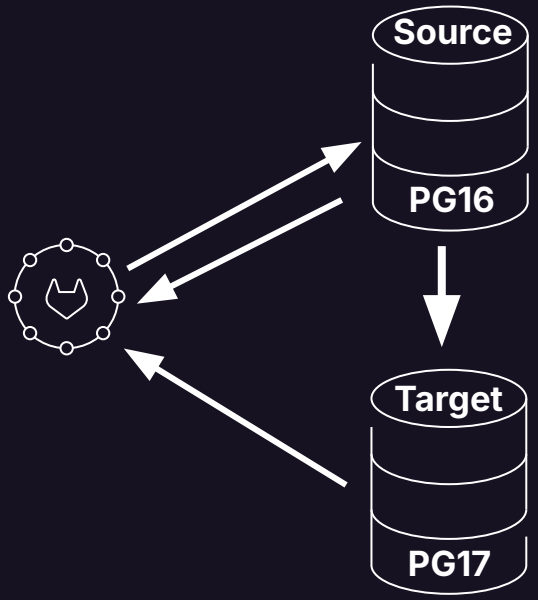


Reindex
Analyze (collect statistics)
Corruption Check

GitLab Copyright

PostgreSQL and OS Upgrade - Sunday

No DDL

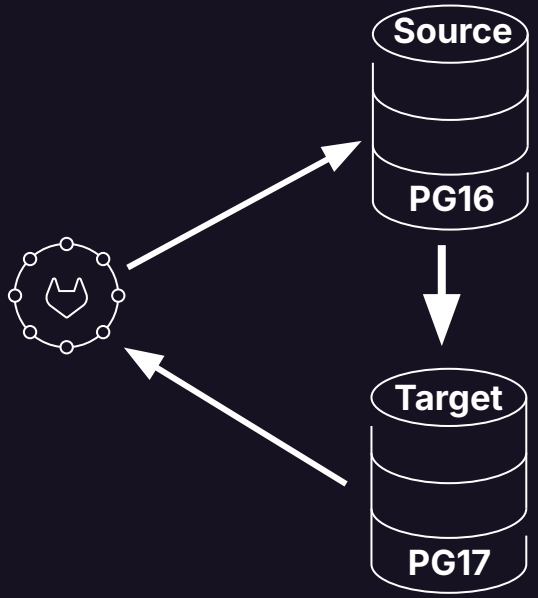


Switchover read-only queries partially
Monitor performance

GitLab Copyright

PostgreSQL and OS Upgrade - Sunday

No DDL

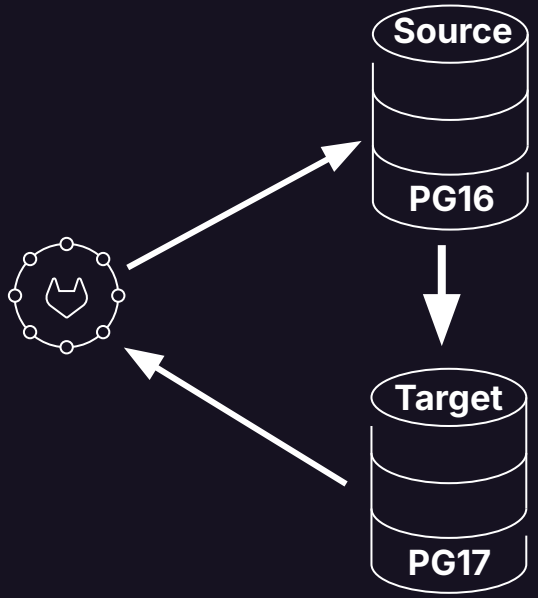


Switchover all read-only queries
Monitor performance

GitLab Copyright

PostgreSQL and OS Upgrade - Sunday

No DDL



Run full QA test suite

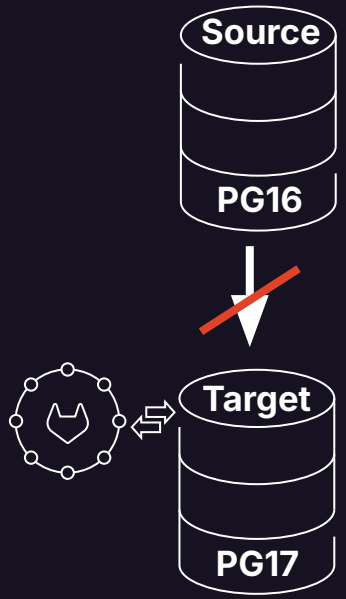
QA + live traffic

Monitor performance

GitLab Copyright

PostgreSQL and OS Upgrade - Sunday

No DDL

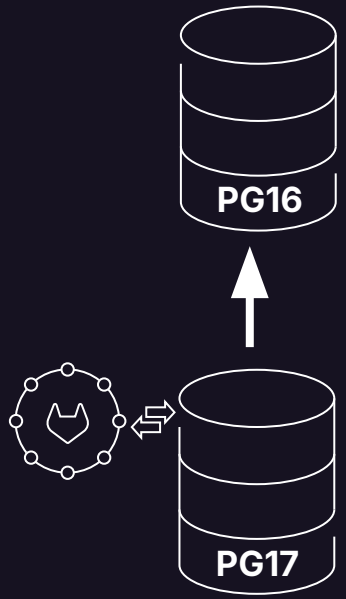


Switchover all load

GitLab Copyright

PostgreSQL and OS Upgrade - Sunday

No DDL

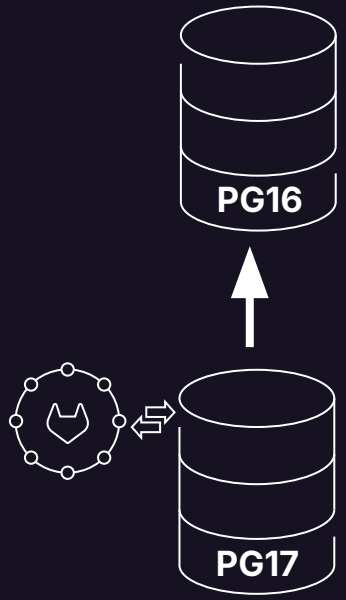


Reverse Replication

GitLab Copyright

PostgreSQL and OS Upgrade - Monday

No DDL

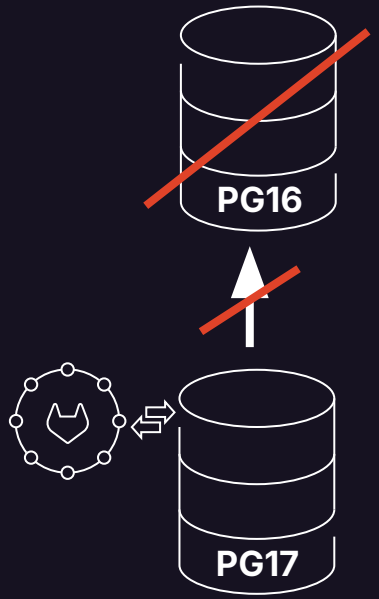


Monitoring during peak hours
(Fast Rollback possible)

GitLab Copyright

PostgreSQL and OS Upgrade - Tuesday

No DDL

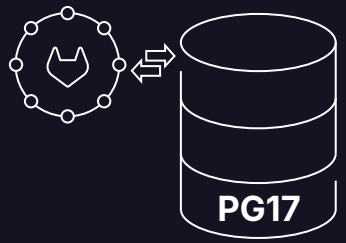


Point of no return

Remove PG16 cluster

GitLab Copyright

PostgreSQL and OS Upgrade - Tuesday



Normal operation

Start planning next upgrade ;)

GitLab Copyright

Resources

- GitLab: about.gitlab.com
- Our RDBMS: about.gitlab.com/handbook/engineering/infrastructure/database
- Ansible Playbooks: gitlab.com/gitlab-com/gl-infra/db-migration
- CR Template: ../db-migration/.gitlab/issue_templates/pg_upgrade.md
- Extended Slide Deck with addition annotations:
  - FOSDEM26 - fosdem.org/2026
  - FOSDEM PGDay 2026 - 2026.fosdempgday.org
- Previous Talk
  - How we execute PG major upgrades at GitLab, with zero downtime. (PGConf.EU 2023) youtube.com/watch?v=o08kJggkovg
- Alexander Sosna
  - sosna.de

GitLab Copyright

Questions?

- During the event
- GitLab Stand at FOSDEM
- Later
- Now!

sosna.de

GitLab Copyright