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

GitLab

GitLab

Alexander Sosna

Senior Database Reliability Engineer

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!

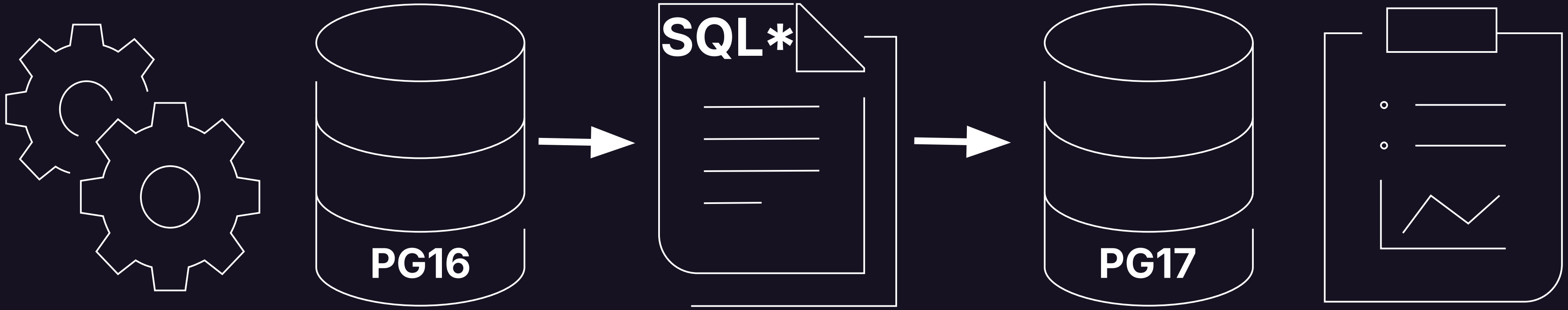
README

- Slides with the white triangle in the corner are not included in the presentation
- They are added to provide more context when reading the slides

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

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)

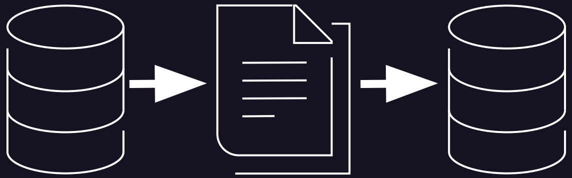
4\.
create indexes
collect statistic

Upgrade Method - pg_dumpall

- Data is extracted and brought to a logical representation
  - SQL, or optimized internal format
- Logical data is then imported in the new cluster
- Both operations are resource and time consuming
  - Can be performed in parallel to disk
    OR
  - Piped from old to new cluster
- All data gets validated
- All indexes are freshly created
- No bloat in the new cluster

x86

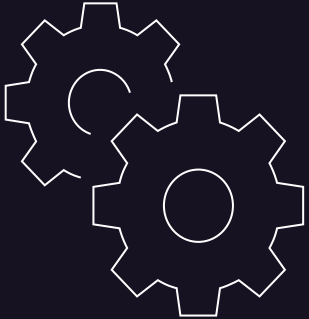
Upgrade Method - pg_dumpall



- **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!

Upgrade Method - pg_upgrade



1\.

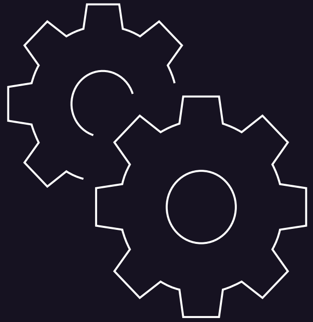
Maintenance mode
(offline / RO with standby)

User impact starts

2\.

In-place upgrading
binary data

Upgrade Method - pg_upgrade



1.

Maintenance mode

(offline / RO with standby)

User impact starts

2.

In-place upgrading

binary data

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!

How did we perform Upgrades in the Past?

pg_upgrade, with significant downtime

1. Create second cluster from backup, called *Target*
2. Sync *Target* with *Source* cluster via streaming replication
3. **Put GitLab.com into maintenance**
4. Used *pg_upgrade* to upgrade *Target* cluster primary
5. Re-create all standbys in *Target* cluster
6. Run full QA tests and benchmark on *Target* cluster (multiple hours)
7. Switch application to use new cluster
8. Bring **GitLab.com** back online

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

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

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 :)

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!

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

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

How do we achieve Zero Downtime?

How do we achieve Zero Downtime?

Logical Replication

How are we achieving Zero Downtime?

Logical Replication
(and a lot of automation)

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, ...

Logical Replication - What is the catch?

1. Database schema and DDL commands are not replicated!
2. Sequences are not replicated, but are needed for auto increment values
3. Each table needs a *REPLICA IDENTITY*, to distribute changes
    - Primary key
    - Other unique key
    - *FULL*, last resort, all changes need to be recorded
4. More complex
    - Prone to human errors
    - Automation and testing is highly advised

Logical Replication - DDL is not replicated

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

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

Logical Replication - Sequences are not replicated

- Sequences are vital to PostgreSQL
  - Generates unique sequential numbers wherever they are needed
  - Used for SERIAL (AUTO INCREMENT)

Logical Replication - Sequences are not replicated

- Sequences are vital to PostgreSQL
  - Generates unique sequential numbers wherever they are needed
  - Used for SERIAL (AUTO INCREMENT)

Our solution

- Measure the daily growth of all sequences
- Defined a large “sequences buffer value”, e.g. *1 million*
- Increase the sequences on the Target cluster by this value
- Before switchover check that the sequences on OLD, have not grown more than expected (optional)
- Simple solution, only uses up a fraction of the key space of 64-bit integer

Logical Replication - REPLICA IDENTITY

- Each table needs a *REPLICA IDENTITY* to clearly identify rows, like:
  - Primary key
  - Other unique key
  - *FULL Record*, last resort, all changes need to be recorded

Logical Replication - REPLICA IDENTITY

- Each table needs a *REPLICA IDENTITY* to clearly identify rows, like:
    - Primary key
    - Other unique key
    - *FULL Record*, last resort, all changes need to be recorded

Our solution

- Nothing to do, GitLab already used primary keys
- You need to check **YOUR** application's schema!

Logical Replication - Complexity

- More complex
  - Prone to human errors

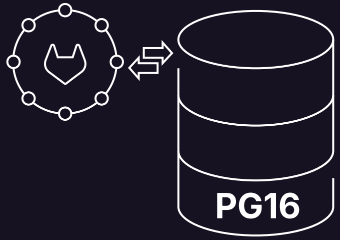
Logical Replication - Complexity

- More complex
  - Prone to human errors

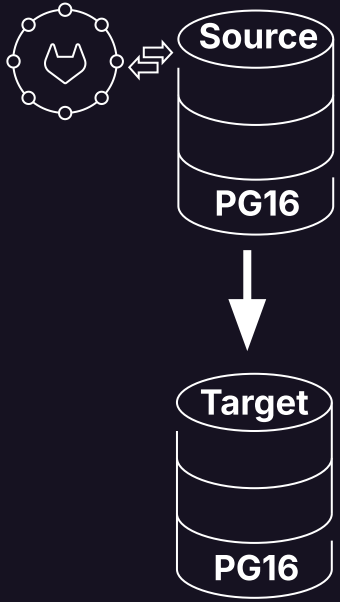
Our solution

- Automation
  - Orchestration via Ansible
  - Process as CR issue which could be executed repetitively
- Excessive testing - *"When it hurts, do it more often"*
  - Intense QA tests before switchover, rollback if not perfect
  - Dry runs in production

Logical Replication + pg_upgrade

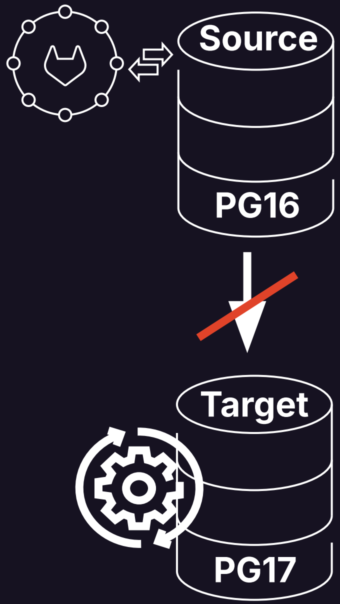


Logical Replication + pg_upgrade



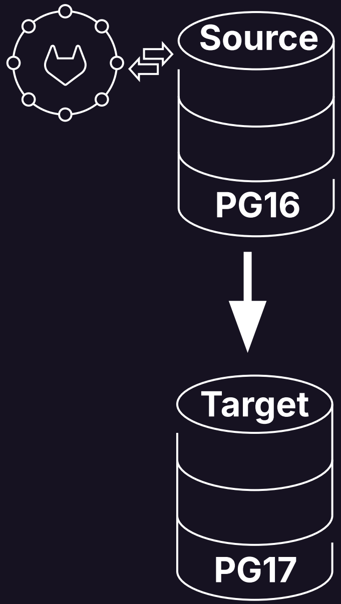
Create and sync Target

Logical Replication + pg_upgrade



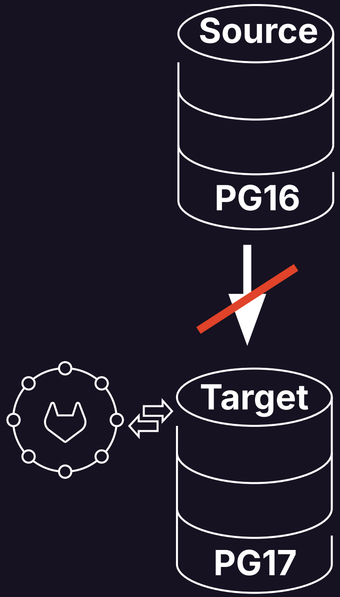
Upgrade Target
(no sync during upgrade)

Logical Replication + pg_upgrade



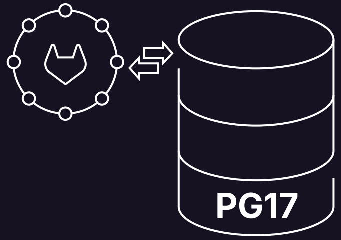
Resync via LR

Logical Replication + pg_upgrade

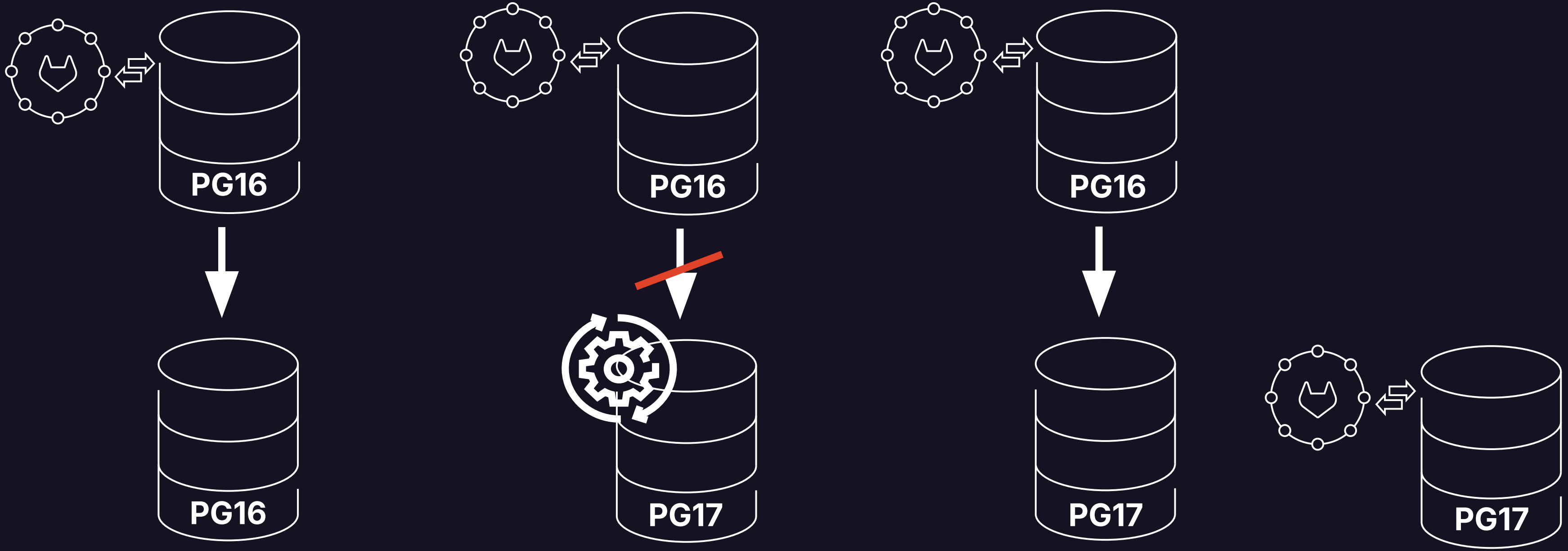


Application Switchover

Logical Replication + pg_upgrade



PostgreSQL Upgrade - State 2023



1.Sync Target

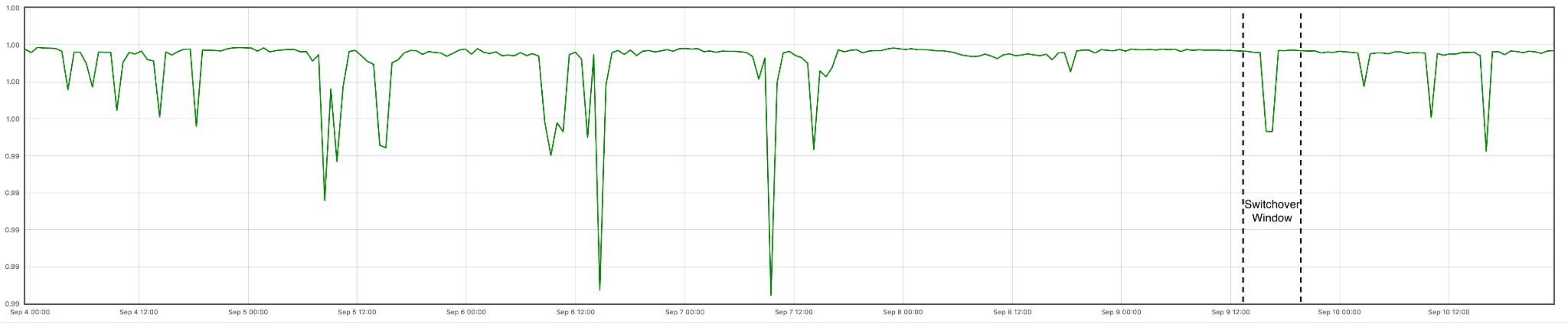
2. Upgrade Target

3. Resync

4. Switchover

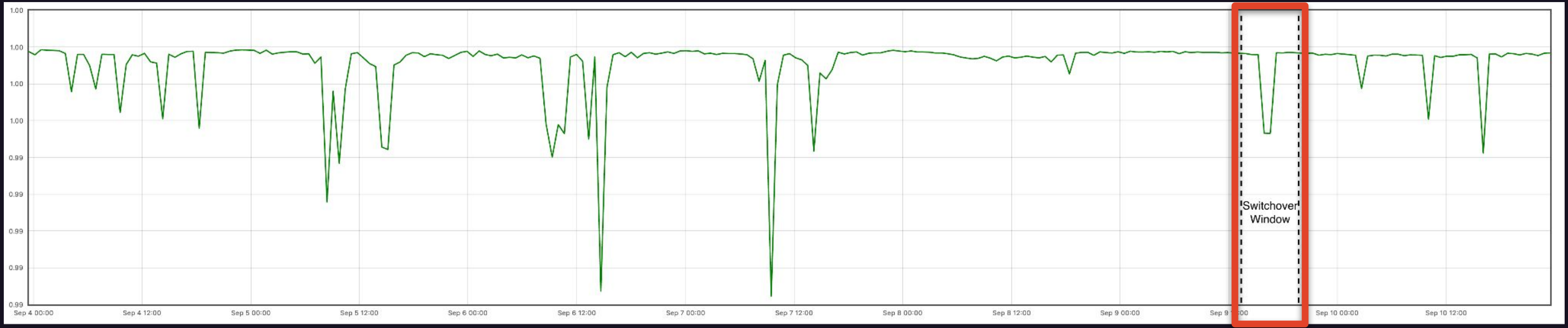
What is actual the User Impact?

How well did we do? - Web Apdex



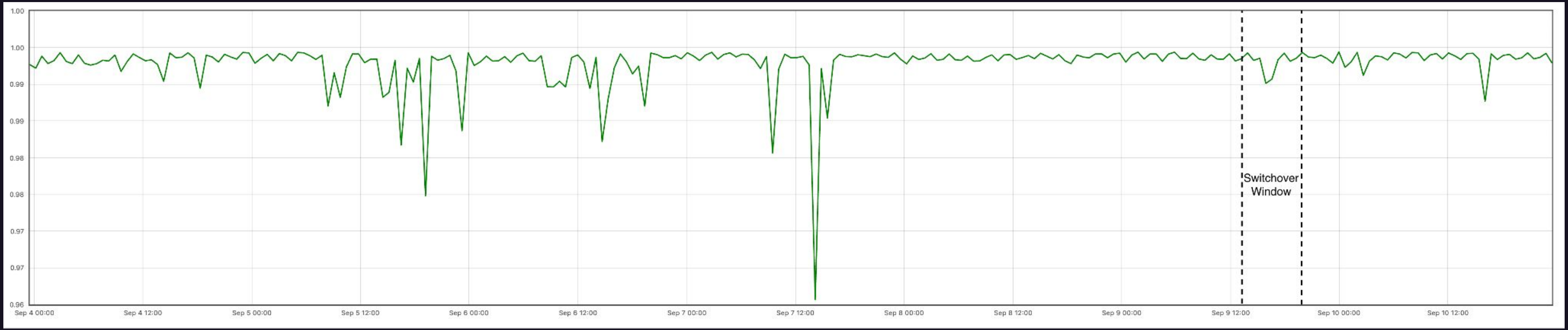
- Web Service Apdex - top 1% (0.99 - 1.00 nit-picking view)
- Degradation SLO: 98.8%, red line would be below this graph :D

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

How well did we do? - API Apdex



Apdex top 4% (0.96 - 1.00)

Can we improve further?

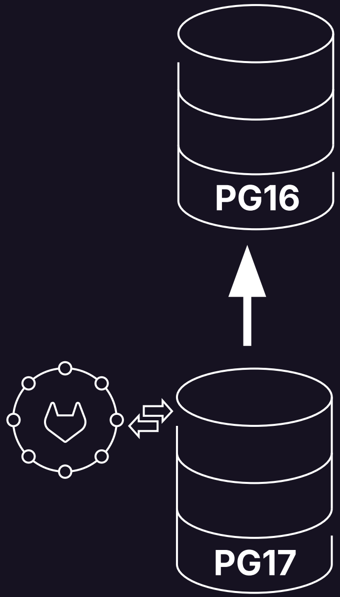
1. Switchover is a Point of no Return
   - If performance degrades or any problem occurs, we can’t go back!
   - Significant business risk!

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

(Re)move Point of no Return - Reverse Replication

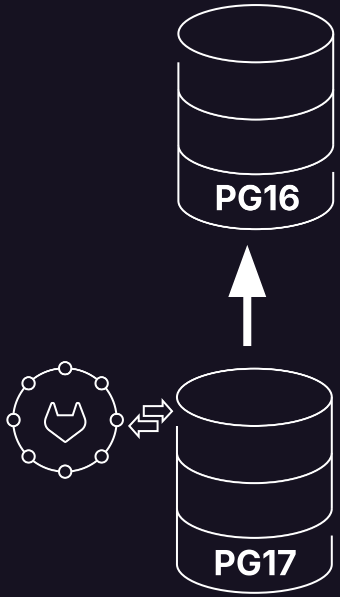
- After the Switchover we reverse the replication
- Enables swift rollback without data loss

(Re)move Point of no Return - Reverse Replication



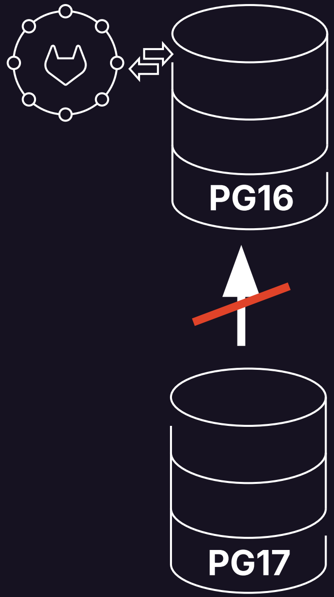
Reverse Replication

(Re)move Point of no Return - Reverse Replication



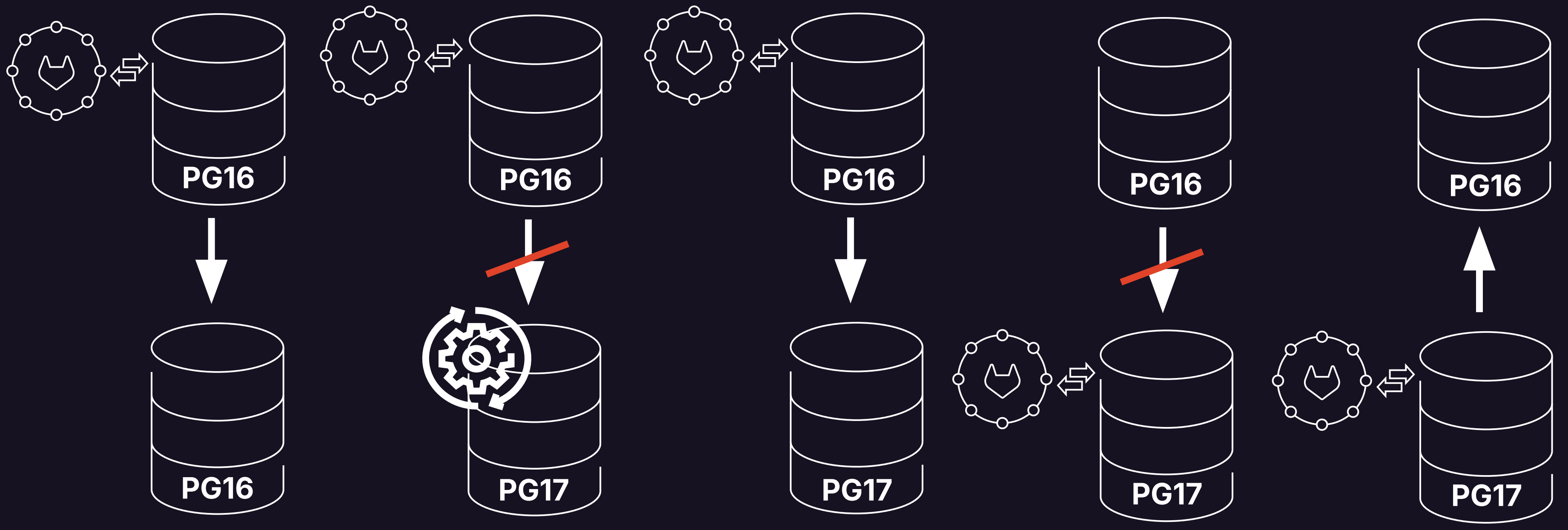
Operation and Monitoring

(Re)move Point of no Return - Reverse Replication



Late Rollback
(optional)

PostgreSQL Upgrade - With Reverse Replication



1.Sync Target

2. Upgrade Target

3. Resync

4. Switchover

5. Reverse Replication

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!

OS Upgrade - Simple Solution

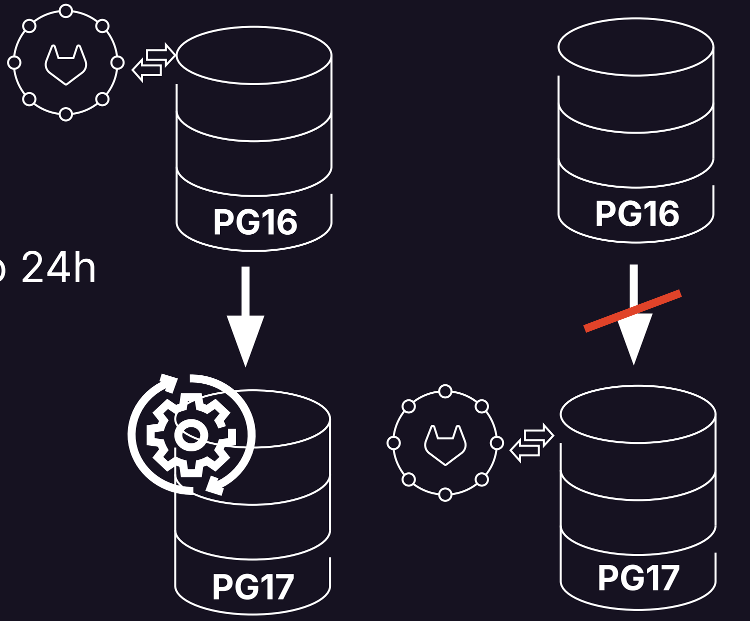
- Some data types don’t use collations and are unproblematic, e.g. INTEGER
- Rebuild all indexes (on strings) with the current collation
  - If this works for your use-case, great!
  - If you use the pg_dumpall upgrade method you get it automatically
  - For GitLab.com this would take multiple days,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

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

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

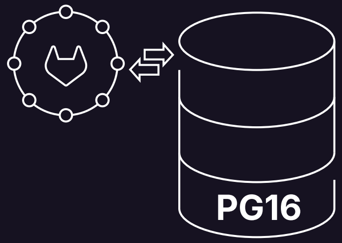


Recreate Indexes
Run amcheck

Switchover

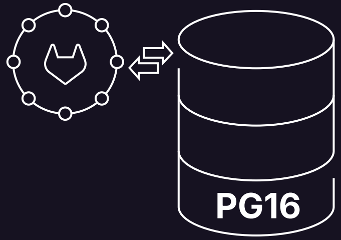
PostgreSQL and OS Upgrade

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

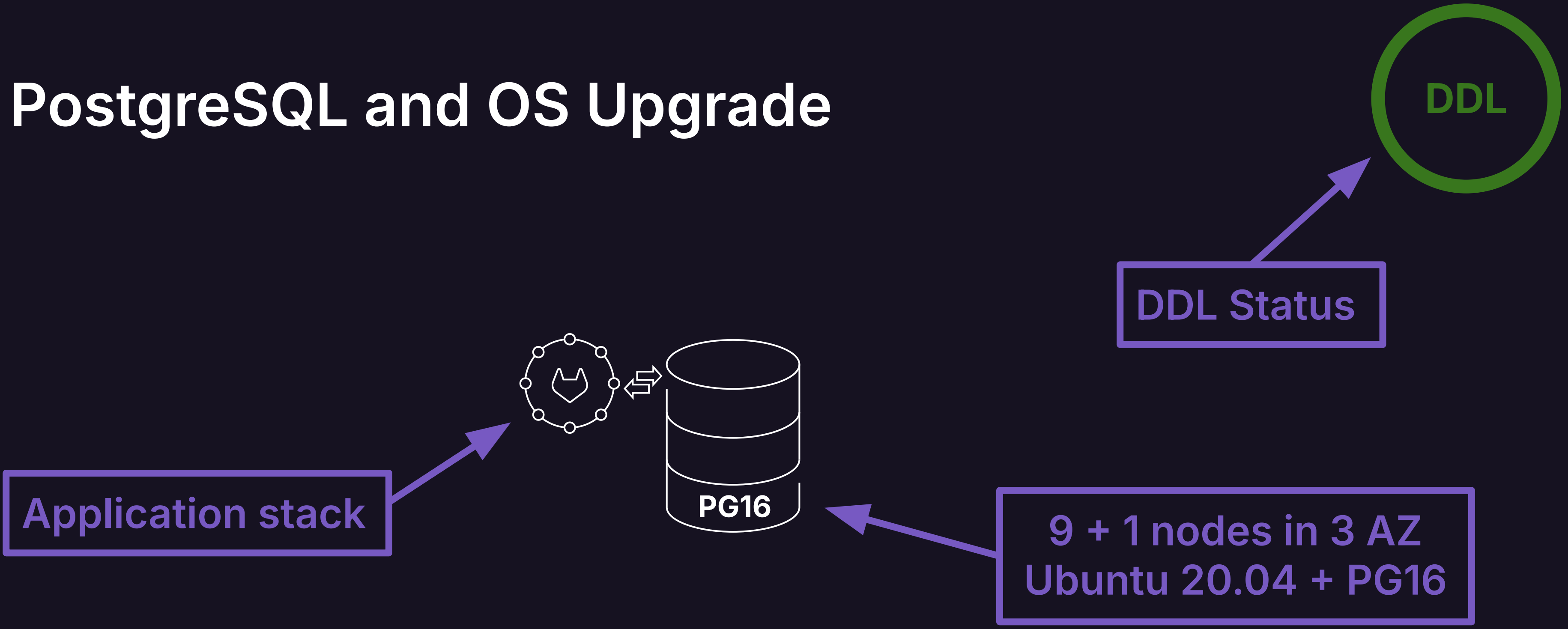


PostgreSQL and OS Upgrade

DDL

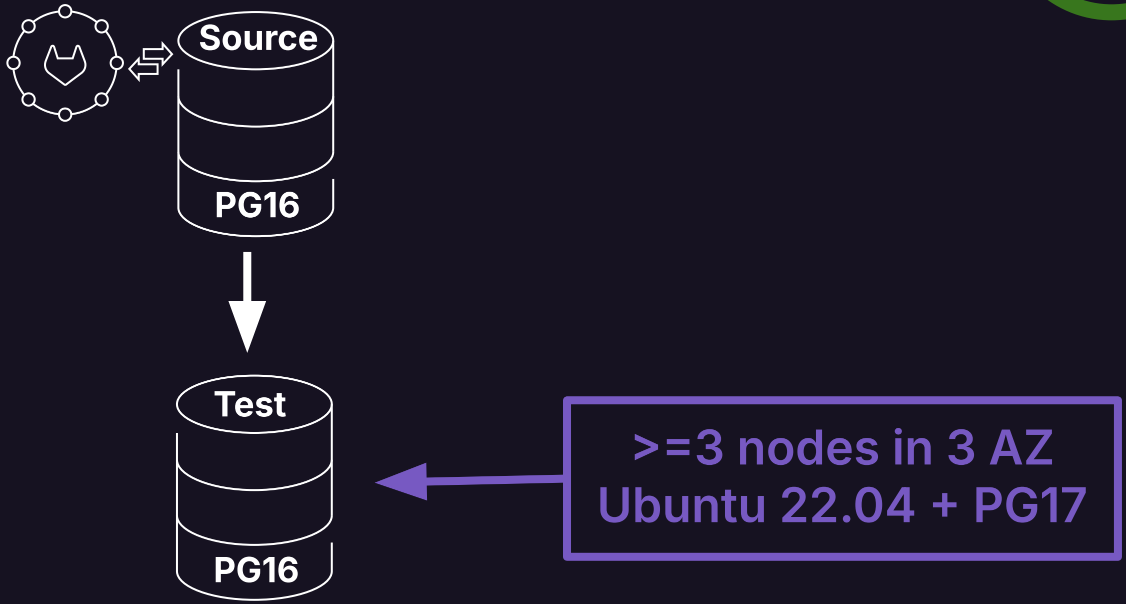


PostgreSQL and OS Upgrade



PostgreSQL and OS Upgrade - Preparation

DDL

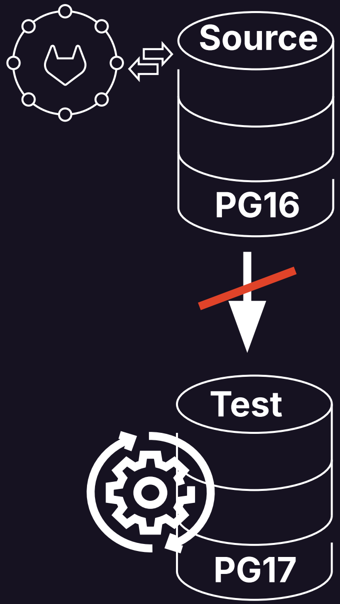


>=3 nodes in 3 AZ
Ubuntu 22.04 + PG17

Create Test clone

PostgreSQL and OS Upgrade - Preparation

DDL



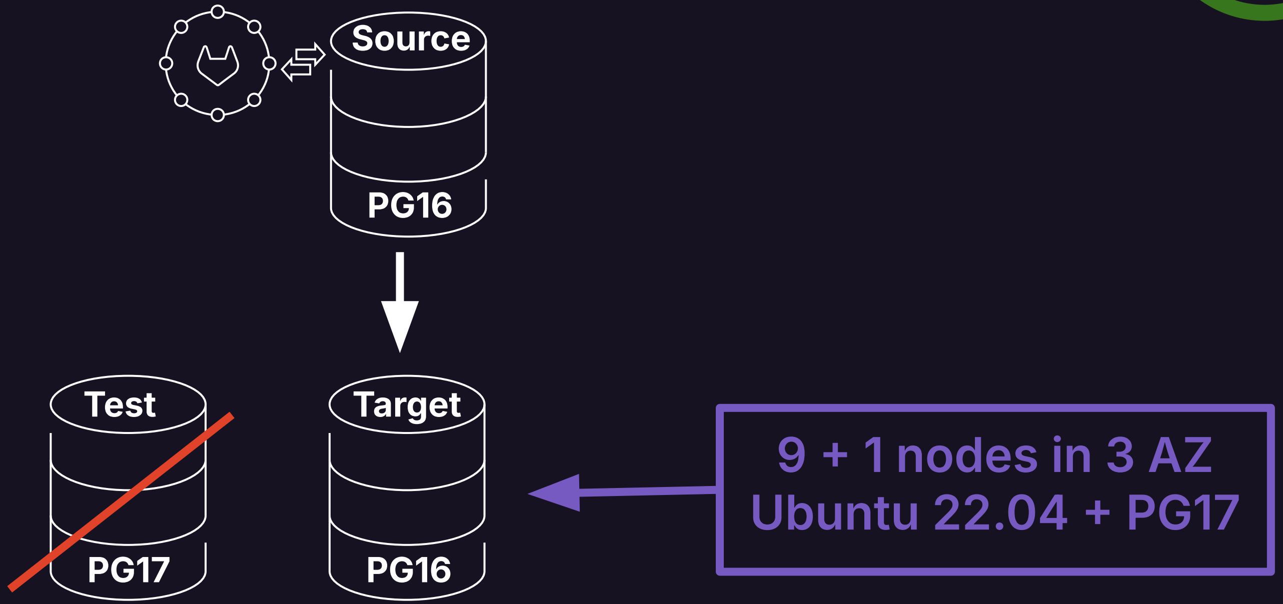
Test upgrade

Get execution times

Get list of corrupted indexes

PostgreSQL and OS Upgrade - Preparation

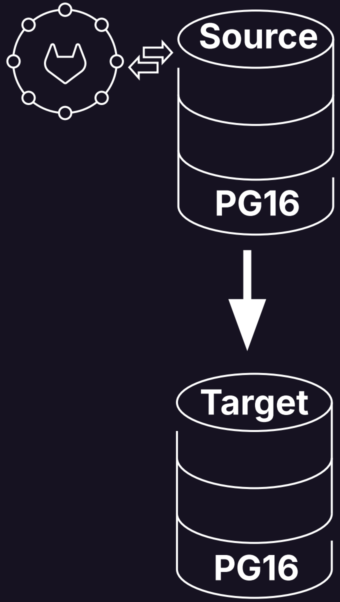
DDL



Remove Test Cluster
Create Target Cluster

PostgreSQL and OS Upgrade - Saturday

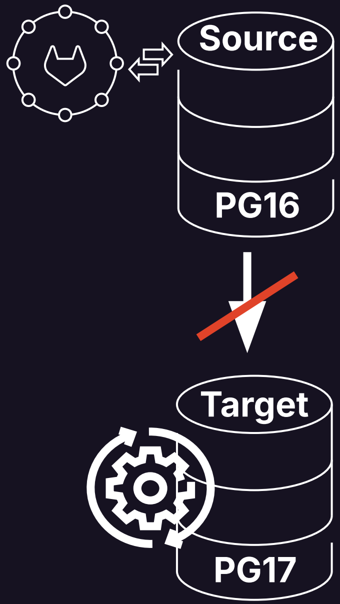
No DDL



Switch to logical replication
(DDL would break it)

PostgreSQL and OS Upgrade - Saturday

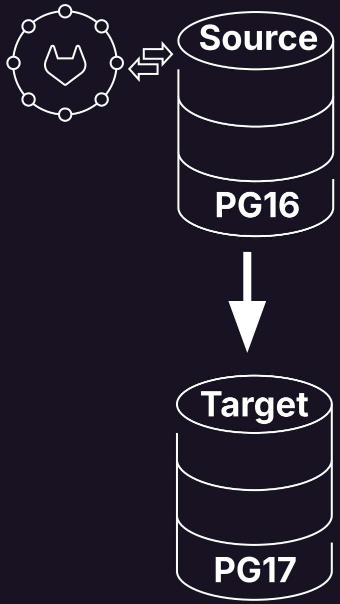
No DDL



Upgrade Target
(no sync during upgrade)

PostgreSQL and OS Upgrade - Saturday

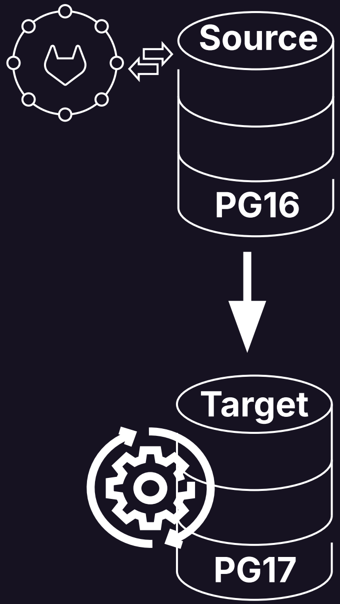
No DDL



Resync

PostgreSQL and OS Upgrade - Saturday

No DDL



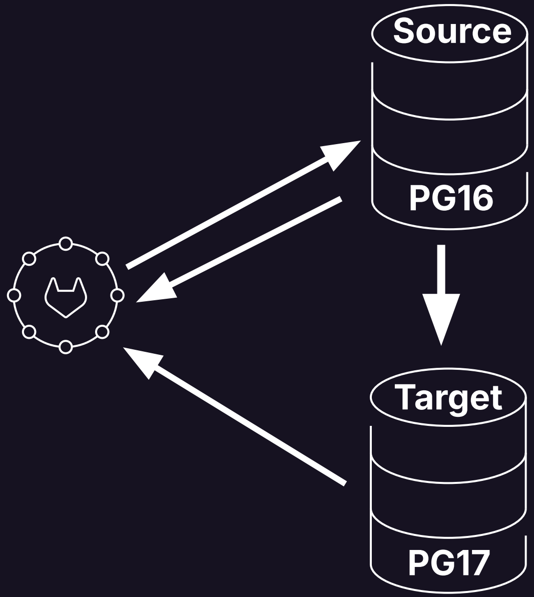
Reindex

Analyze (collect statistics)

Corruption Check

PostgreSQL and OS Upgrade - Sunday

No DDL

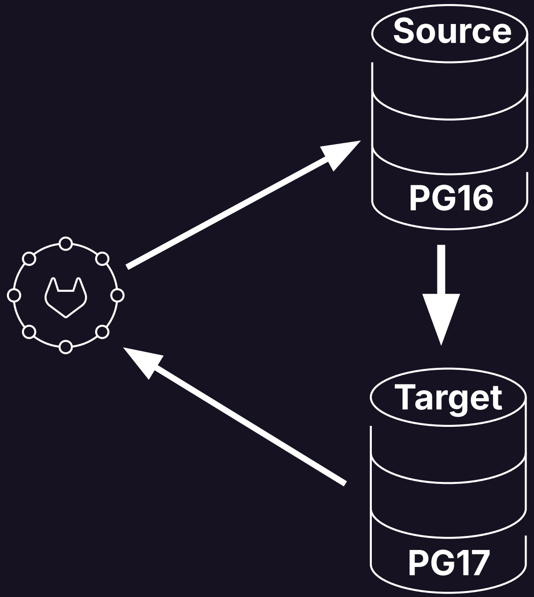


Switchover read-only queries partially

Monitor performance

PostgreSQL and OS Upgrade - Sunday

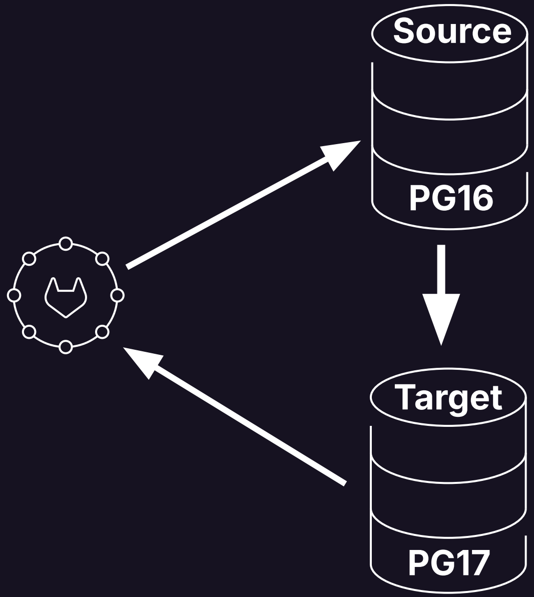
No DDL



Switchover all read-only queries
Monitor performance

PostgreSQL and OS Upgrade - Sunday

No DDL



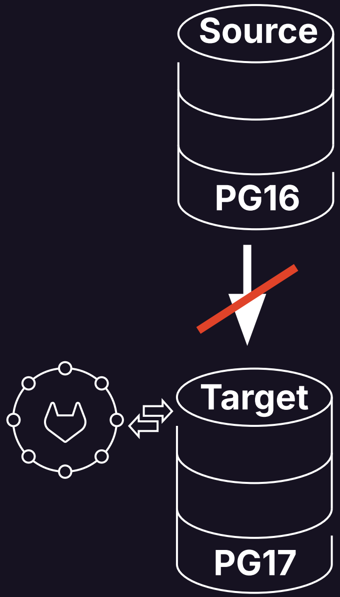
Run full QA test suite

QA + live traffic

Monitor performance

PostgreSQL and OS Upgrade - Sunday

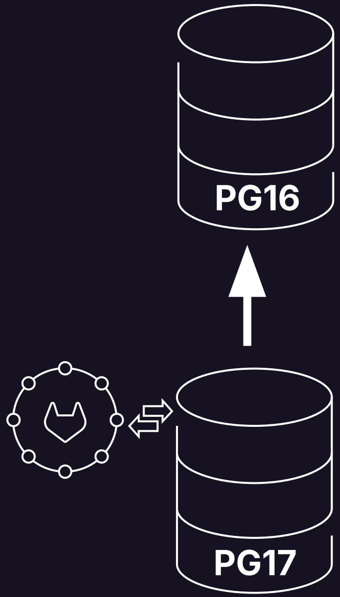
No DDL



Switchover all load

PostgreSQL and OS Upgrade - Sunday

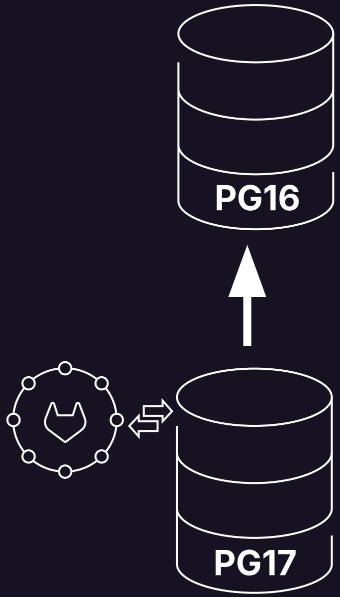
No DDL



Reverse Replication

PostgreSQL and OS Upgrade - Monday

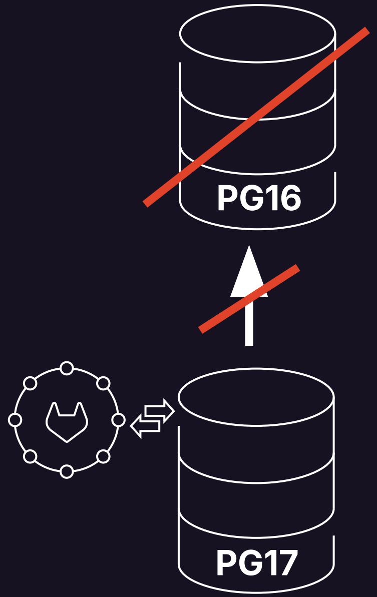
No DDL



Monitoring during peak hours
(Fast Rollback possible)

PostgreSQL and OS Upgrade - Tuesday

No DDL

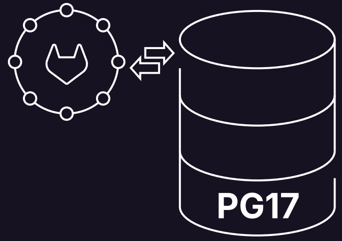


Point of no return

Remove PG16 cluster

PostgreSQL and OS Upgrade - Tuesday

DDL



Normal operation

Start planning next upgrade ;)

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

Questions?

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

sosna.de