Keeping up with the lifecycle

Thomas Boussekey - MIRAKL

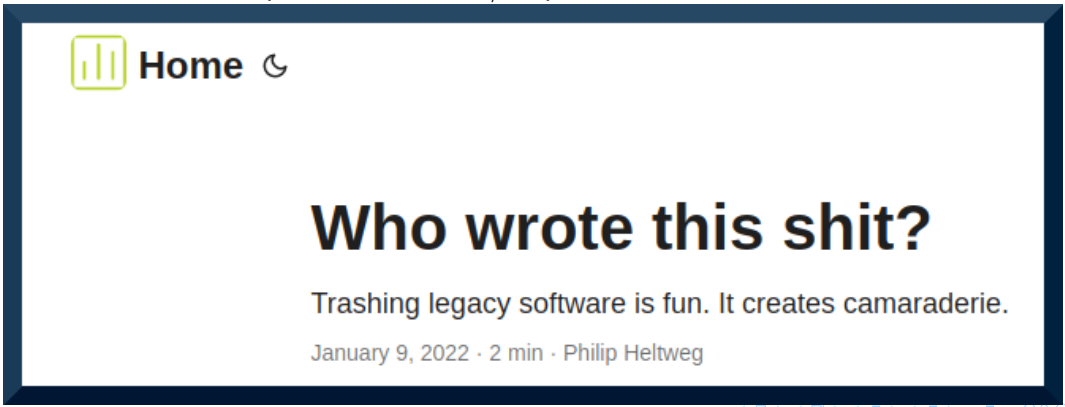
Feedback from a 4 major versions pg_upgrade

2022-03-24

Set the context

Disclaimer https://www.heltweg.org/posts/who-wrote-this-shit/

No intent to criticize previous decisions / implementations



Context

Who am I ?

Context

Who am I ?

Mirakl?

- 500+ employees (onSite + remote)
- Offices in:
  - FR: Paris/Bordeaux
  - US: Boston
  - UK: London
  - BR: Sao Paulo
  - .../...

We're hiring https://labs.mirakl.jobs/

Marketplaces

Convert an e-commerce website into a virtual mall:

- B2B / B2C
- Services
- Dropship model

- Standardize product's catalogs
- Integrate offers
- Animate operators:
  - Orders' delivery
  - Multiple relations (CRM)
- .../...

If you shop helicopter spare parts at **AIRBUS helicopter** or home theatre at **BEST BUY**, you are on a **MIRAKL**'s marketplace

Figure 1: How does it work?

Discover my new playground (April 2019)

SRE mindset

- Infra as a code +++
- no manual operation on a Pg instance
- multi customers / clouds
  –> *Automate everything*

100% cloud

PostgreSQL (2 hosting modes)

- VM + hot standby
- 1 POD on Kubernetes

Discover my new playground (April 2019)

SRE mindset
- Infra as a code +++
- no manual operation on a Pg instance
- multi customers / clouds –> *Automate everything*

100% cloud

PostgreSQL (2 hosting modes)
- VM + hot standby
- 1 POD on Kubernetes

Landscape overview in 2019
- 600 (dev + test)
- 300 demo
- 70 preprod
- 200 prod

14 TB (production live data)
- From 50 GB to 1.4 TB

My “technical ladder”

From PUPPET to ANSIBLE:

- 2 good tools
- 2 different implementations
- 2 way of thinking

Fighting against BLOAT

- Script it / Delegate it / Automate it
- Rework batches (less is better)

My “technical ladder”

From PUPPET to ANSIBLE:

- 2 good tools
- 2 different implementations
- 2 way of thinking

Fighting against BLOAT

- Script it / Delegate it / Automate it
- Rework batches (less is better)

Using LargeObjects:

- Store large documents into the database
- Need to FULL VACUUM the `pg_largeobject` table
  - No Possible **repacks**

Prefer ***BYTEA***

PostgreSQL 9.5, close to end-of-life

- 2021-02-11

Pave the way to Pg12

Dismantle **LargeObjects** customizations

Very useful at the early phase of the company

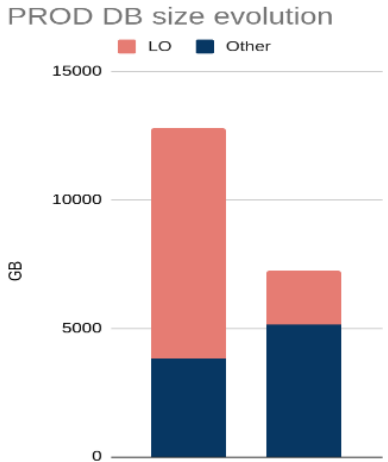
- consistent environment with:
  - relational data
  - product images
  - documents (orders, accounting...)

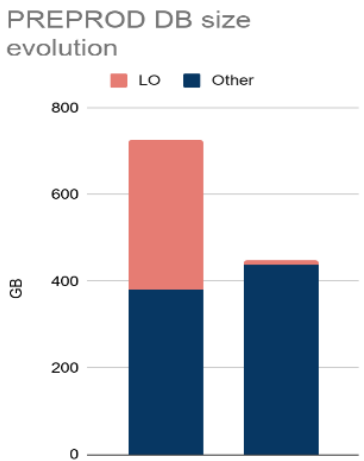
Located on a **specific tablespace**

- different disk
- lower bandwidth
- improved pricing

More and more functionnal objects have been moved to other storage types, so there was **BLOAT**

Full vacuum **LargeObjects**





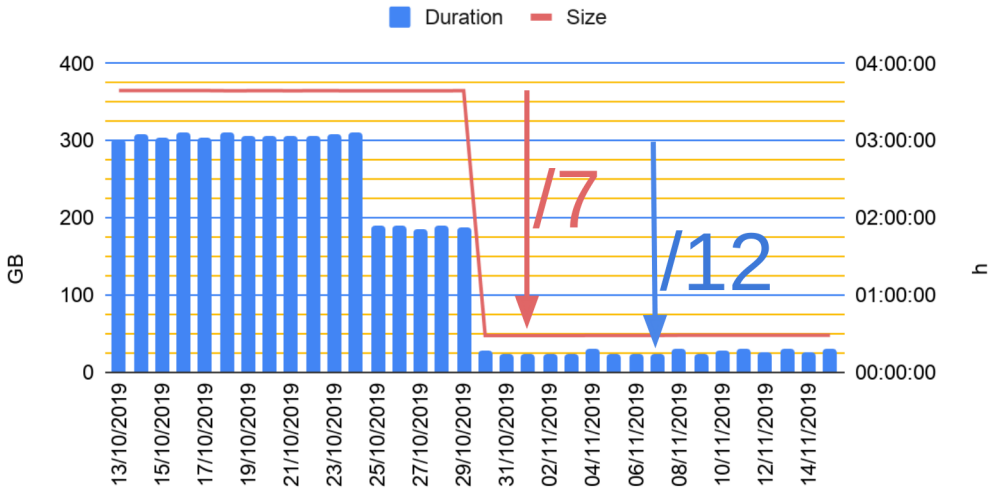


Figure 3: LargeObject cleanup benefits

Prepare the upgrade?

Mandatory tasks

1. Read the release notes
2. Try a blank install of the new version
3. Migrate an empty instance
4. Automate the installation
5. Script the migration

Improve it!

- Make it work
- **(if possible)** find intermediate steps
- Test it on live data
- Automate the migration
- Improve logging
  *View your scripts/processes **FAIL**!*

Read the release notes

4 major versions

- Pg9.5 → Pg9.6 → Pg10 → Pg11 → Pg12

Read the release notes

4 major versions

- Pg9.5 → Pg9.6 → Pg10 → Pg11 → Pg12
  Change in the version numbering

- Allow **common table expressions** (CTEs) to be inlined into the outer query (Andreas Karlsson, Andrew Gierth, David Fetter, Tom Lane)

  Specifically, CTEs are automatically inlined if they have no side-effects, are not recursive, and are referenced only once in the query. Inlining can be prevented by specifying `MATERIALIZED`, or forced for multiply-referenced CTEs by specifying `NOT MATERIALIZED`. Previously, CTEs were never inlined and were always evaluated before the rest of the query.

Figure 4: Change on CTE

- Enable **Just-in-Time** (JIT) compilation by default, if the server has been built with support for it (Andres Freund)

  Note that this support is not built by default, but has to be selected explicitly while configuring the build.

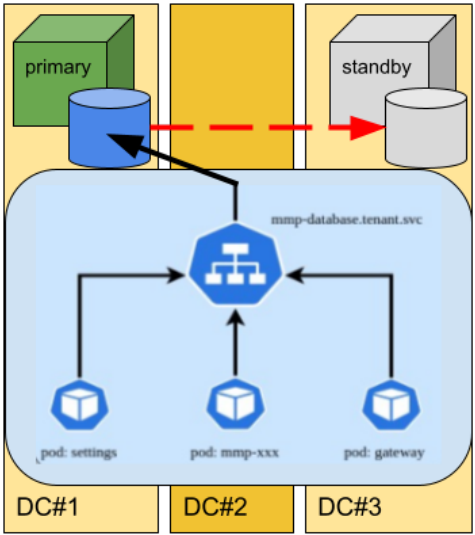
Figure 5: Change on JIT – not identified

15. **Statistics**

Because optimizer statistics are not transferred by `pg_upgrade`, you will be instructed to run a command to regenerate that information at the end of the upgrade. You might need to set connection parameters to match your new cluster.

Figure 6: Post upgrade operation

Design the technical solution



Key-drivers used to design the solution

Minimal downtime

A reliable rollback plan

Impossible to restart **WITHOUT**:

- A live standby
- A full basebackup

Upgrade process chosen

Replication:

- Resync standby with `RSYNC`
- Set a fresh Pg-12 standby instance on the former Pg9.5

Test:

- Same duration
- No basebackup if **rsync standby**

Decision:

- Perform basebackup
- Reinstate standby

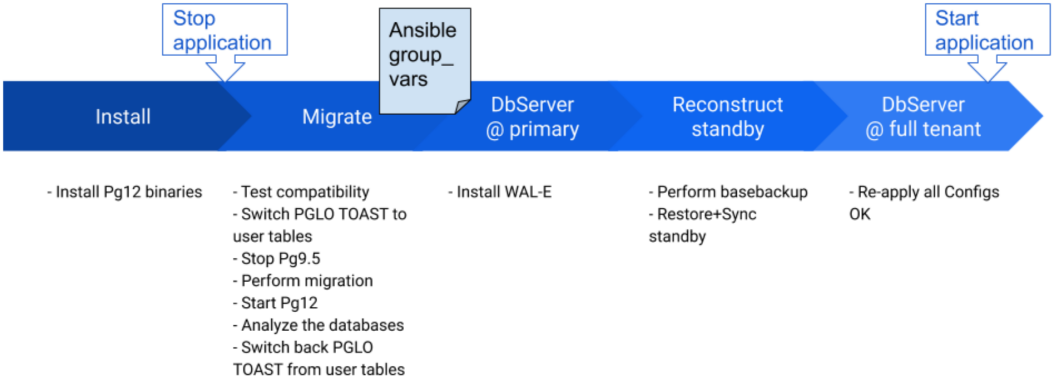


Figure 8: Process chosen

V1, automate it!

After manual operation ...

```
function perform_upgrade( ) {
  verbose=${1:-""}
  /usr/pgsql-12/bin/pg_upgrade --username=postgres \
    --old-bindir "${OLDBIN_DIR}" --new-bindir "${NEWBIN_DIR}" \
    --old-datadir "${OLDDATA_DIR}" --new-datadir "${NEWDATA_DIR}" \
    --old-options "-c config_file=${OLDCONFIG_DIR}/postgresql.conf -c
hba_file=${OLDCONFIG_DIR}/pg_hba.conf" \
    --new-options "-c config_file=${NEWCONFIG_DIR}/postgresql.conf -c
hba_file=${NEWCONFIG_DIR}/pg_hba.conf" \
    --link ${verbose} | tee -a "${LOG_UPGRADE}"
  UPGRADE_RC="${PIPESTATUS[0]}"
  echo -e "Return code from CHECK operation ::${UPGRADE_RC}::" | tee -a "${LOG_UPGRADE}"
  exit "${UPGRADE_RC}"
}
```

```
function perform_check() {
  /usr/pgsql-12/bin/pg_upgrade --username=postgres \
    --old-bindir "${OLDBIN_DIR}" --new-bindir "${NEWBIN_DIR}" \
    --old-datadir "${OLDDATA_DIR}" --new-datadir "${NEWDATA_DIR}" \
    --old-options "-c config_file=${OLDCONFIG_DIR}/postgresql.conf -c
hba_file=${OLDCONFIG_DIR}/pg_hba.conf" \
    --new-options "-c config_file=${NEWCONFIG_DIR}/postgresql.conf -c
hba_file=${NEWCONFIG_DIR}/pg_hba.conf" \
    --link --check --verbose | tee -a "${LOG_CHECK}"
  CHECK_RC="${PIPESTATUS[0]}"
  echo -e "Return code from CHECK operation ::${CHECK_RC}::" | tee -a "${LOG_CHECK}"
  exit "${CHECK_RC}"
}
```

Figure 10: Add TEST mode before live migration

Test on production-like environment

```
pg_enable_version: "9.5"
pg_version: "9.5"
pg_package_version: "95"
pg_minor_version: 9.5.19
pg_backup_enable: true
python_version: 2
psycopg2_version: 2.8.3
pg_repack_version: 1.4.4
wale_version: 0.9.2
```

```
pg_enable_version: "12"
pg_version: "12"
pg_package_version: "12"
pg_minor_version: 12.7
pg_backup_enable: true
python_version: 3
psycopg2_version: 2.8.5
pg_repack_version: 1.4.5
wale_version: 1.1.1
```

Validate that ANSIBLE playbooks can work with the 2 versions

- Introduce new variables:
  - `postgres_major_version`
  - `postgres_package_version` handling 2-version numbering
- **ALL** operational scripts must work without regression!
  - `else` new processes must be scripted, documented & shared with the team

```
pg_stat_statements.max = 1000
pg_stat_statements.track = all

{% if postgres_package_version == "12" %}
jit = off
{% endif %}
```

Figure 12: Adaptative configuration

```
TENANT_NAME="pg12inst1-test" HOSTER="${HOSTER}" \
RESTORATION_BASE="abb-prod" TAINT_PREVIOUS=true \
CREATE_MISSING_TENANT=true PERFORM_ALL_TESTS=true \
PERFORM_MIGRATION=true ./tests/tenants/pg/rebuild_tenant.sh
```

Figure 13: Test the migration

```
#!/usr/bin/env bash
#
### rebuild_tenant.sh -- Rebuild a TEST tenant according to the parameter provided
###
### Global variables:
###   PERFORM_SWITCHOVER -> Boolean to define if replication switchover needs to be tested
###            DEFAULT VALUE: false
###   PERFORM_FAILOVER   -> Boolean to define if replication failover needs to be tested
###            DEFAULT VALUE: false
###   PERFORM_RESTART    -> Boolean to define if the PG services needs to be restarted
###            DEFAULT VALUE: false
###   PERFORM_RELOAD     -> Boolean to define if the PG services needs to be reloaded
###            DEFAULT VALUE: false
###   PERFORM_ALL_TESTS  -> Boolean to define if all the tests must be performed:
###            TESTS: switchover,failover,reload,restart,pgsql_func_test,pgsql_tech_test
###            DEFAULT VALUE: false
```

Figure 14: Test configuration

Validate the rollback plan!

Keep the standby instance on Pg9.5 until primary is switched to Pg12

Disable the hot standby process

. . . before launching the UPGRADE

Streaming replication uses 2 processes in order to stay up-to-date:

- partial-WAL Streaming from primary
- full-WAL restoration recovered from archive storage

And validate:

- migration can work
- in case of **failure**, you can rollback your migration

On Pg9.5, you cannot promote a standby after using `pg_wal_replay_pause`... seems to be possible in newer release

```
--- a/var/lib/pgsql/9.5/data/recovery.conf
+++ b/var/lib/pgsql/9.5/data/recovery.conf

# Connect to the master postgres server using the replicator user we created.
-primary_conninfo = 'host=1.2.3.4 port=5432 user=xxx password=yyy'
+primary_conninfo = 'host=1.2.3.4 port=54329 user=xxx password=yyy'

# Required for archive recovery if streaming replication falls behind too far.
-restore_command = 'wal-e --s3-prefix s3://my-dump_folder/my-instance/ wal-fetch %f %p'
+restore_command = 'wal-e --s3-prefix s3://my-dump_folder/my-instance/DoNotUse/ wal-fetch
%f %p'
```

Figure 15: Block streaming replication

Ready to migrate

Migrate early stages (dev/test)

. . . but not all of them
Keep some instances for validation purposes

Switch creation process to the target version

Migrate small and least critical PREPROD & PROD instances

We secured the migration thanks to:

- strengthening the pg_upgrade script with `test`
- re-test with a full basebackup

Plan downtime operations for critical instances

Validate the configurations

Extensions (link or bridge) versions

Custom RPM for Postgis

- containing **ONLY** server tools
- lower footprint

Pg	9.5	12-managed	12
3.1			
3.0			
2.5.2	avail.	avail.	avail.
2.4.2	avail.		

OS upgrade

Should we use this PostgreSQL project to update the operating system?

OS upgrade

Should we use this PostgreSQL project to update the operating system?

No

- Work step by step
- Easier to test / validate / fix / rollback

Problems encountered

LOCALE

- Few PostgreSQL instances have been installed with `fr-FR.UTF8` instead of `en_US.UTF-8`

Others

- JIT impact on some queries
- Disable autovacuum before `pg_upgrade` to prevent locking on post-upgrade **ANALYZE**

Technical handover

Make technical migration work

Test,test and test again ... Then improve & speed up!

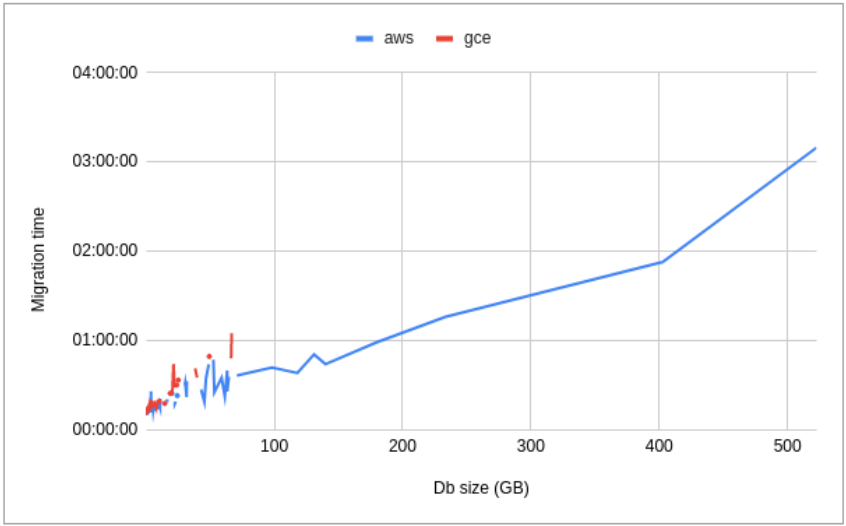
Safety measure 1

All PostgreSQL instances' migration were retested during the week before D-day (using daily basebackup on a TEST instance).

- revalidate that the instance didn't get a breaking change in its configuration
- refine the migration downtime

Safety measure 2

Check BLOAT and repack highest bloated tables, in order to speed up the migration.


aws
gce
04:00:00
03:00:00
02:00:00
01:00:00
00:00:00
Migration time
100
200
300
400
500
Db size (GB)

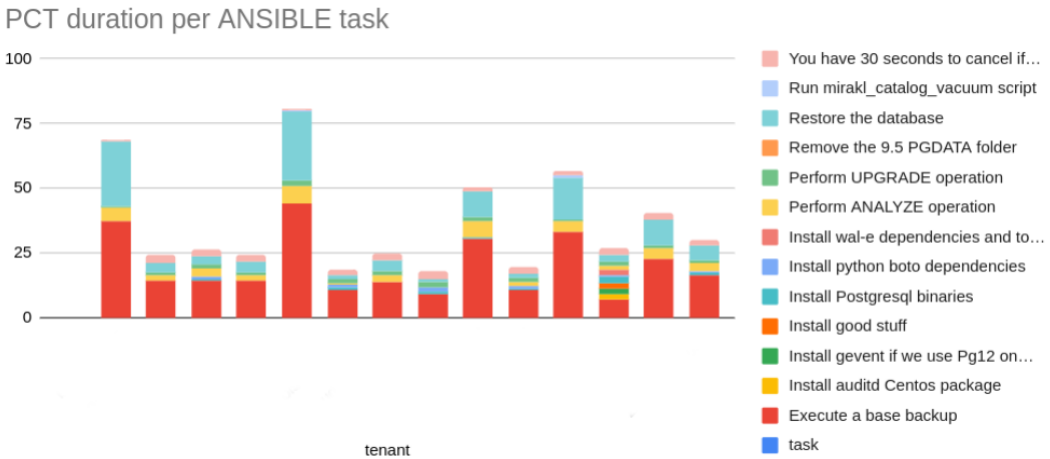


Figure 17: Focus into task step durations

From `wal-e` to `wal-g`

- often when problem occurs on `wal-e`, issues relate that it is fixed on `wal-g`
- more friendly configuration:
  - JSON configuration file (can be overidden by environment variables)
  - Ability to specific resources to basebackup operations

PostgreSQL	tool	MB/min
9.5	wal-e	11,879
12	wal-e	5,632
12	wal-g	14,863

Boost basebackup throughput

```
{
  "WALG_S3_PREFIX": "{{ wale_s3_prefix }}",
  "AWS_REGION": "{{ bucket_region }}",
  "PGDATA": "/var/lib/pgsql/12/data",
  "PGDATABASE": "postgres", .../...
{{ walg_upload_tuned_parameters if boost_walg_throughput else '' }}
}
```

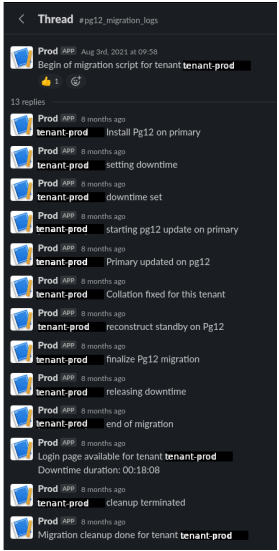
```
walg_upload_tuned_parameters: ', "WALG_UPLOAD_CONCURRENCY": {{ walg_cpu_boost_factor }},
"WALG_UPLOAD_DISK_CONCURRENCY": {{ (walg_cpu_boost_factor | int) / 16 | int }}'

walg_cpu_boost_factor: "{{ [64, (ansible_processor_vcpus|default(1) * 4) | int ] | max }}"
```

For some tenants, all these improvements were not sufficient. We had to:

- upgrade instance configuration (add CPU + RAM)
- increase disk throughput (add IOPS)

Automate it!

From manual operation launching multiple playbooks to 1 single script with **SLACK** messaging


Thread #pg12_migration_logs
Prod APP Aug 3rd, 2021 at 09:58
Begin of migration script for tenant tenant-prod
13 replies
Prod APP 8 months ago
tenant-prod Install Pg12 on primary
Prod APP 8 months ago
tenant-prod setting downtime
Prod APP 8 months ago
tenant-prod downtime set
Prod APP 8 months ago
tenant-prod starting pg12 update on primary
Prod APP 8 months ago
tenant-prod Primary updated on pg12
Prod APP 8 months ago
tenant-prod Collation fixed for this tenant
Prod APP 8 months ago
tenant-prod reconstruct standby on Pg12
Prod APP 8 months ago
tenant-prod finalize Pg12 migration
Prod APP 8 months ago
tenant-prod releasing downtime
Prod APP 8 months ago
tenant-prod end of migration
Prod APP 8 months ago
Login page available for tenant tenant-prod
Downtime duration: 00:18:08
Prod APP 8 months ago
tenant-prod cleanup terminated
Prod APP 8 months ago
Migration cleanup done for tenant tenant-prod

Use Pg12

partitioning

- Native partitioning is easier to implement than trigger-based one!
- with or without `pg_partman`

Great for log tables without foreign key, that contains only last X days

On Pg12, *not a good idea for tables with cascading foreign keys and you cannot rewrite the queries:*

- ex: job history framework with a job/step logic, and queries to get the steps per execution status

Improve configuration, again and again

Tune some parameters:

- `autovacuum_freeze_max_age`
- `max_wal_size`

Draw the future

Pg15

- Remove LargeObjects in order to use logical replication for migration

Dev DB?

For **reliability purposes**, early stage databases were moved from a single Kubernetes POD to a cluster of databases on a cloud managed database.

- Managed or not managed?
- Re-contenerized with operator + replica?

Conclusion

Scrumify your project

- Easier to manage
- More motivating, you will validate more steps
- Can be performed by more people in parallel
  - Some task may not require database knowledge

Trust your configuration management tool

- Improve it
- Strengthen it
- Cross-check it

Test, test again as close as possible to PRODUCTION

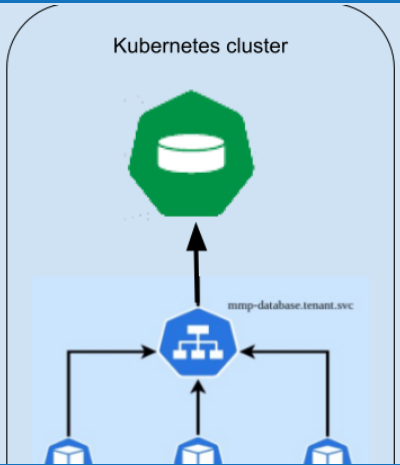
. . .

Thanks for your attention!

Any question?

Switch dev DB

New PostgreSQL major version, full new ecosystem?



Drawbacks of the existing solution

- downtime on k8s node rollout
- Solution to improve the situation
  - Managed DB
    - Already implemented on other projects
  - Implement operator + replica
    - Fear to spend time to study PostgreSQL operators

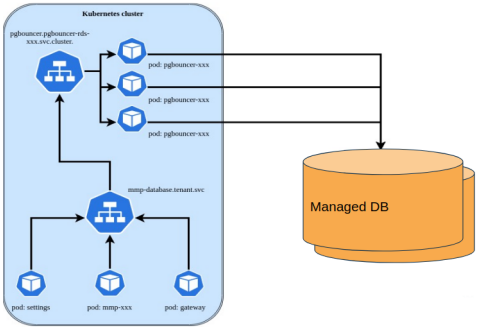


Figure 20: architecture