# A Refined Approach to Testing Databases

29.8.2024 / Alen Krmelj

Bitstamp

01

## AWS Section

## Our databases

**Main database:**

Type: Aurora Cluster
Size: db.r5.12xlarge
Storage: 1.98 TB of data
Largest table: 596 mil rows (market candle)

**Trading database:**

Type: Aurora Cluster
Size: db.r5.12xlarge
Storage: 3.4 TB of data
Largest table: 8.3 bil rows (order log)

**Additional notes:**

- Smallest instance we can restore at this size is db.r5.large (query problems otherwise)
- Migrations testing instance size: db.r5.4xlarge (minimal diff vs production)

# Requirements for Testing Database Migrations

As with many tech companies using RDBMS databases, it's important for us to have the ability to test the migrations that our developers create for our database.

In order to address this issue, our management came up with requirements for it:

- Run migrations for every pull request (App+Infra)
- Don't share PII data! (Infra)
- Parallel migrations, where possible (App)
- Metrics, duration and inform developer if too long, prevent merge (App)
- Don't conflict migrations (App)
- Run on real production data (Infra)
- Don't introduce significant additional costs (Infra)
- Use as many AWS managed components as possible, so we don't have to take care of the infrastructure (Infra)

## Some of the approaches

We could approach this problem from several angles. Most of them suck.

1. **Naive approach:**
- Restore DB from snapshot for each PR
- Wait 40 mins for db to come up and run the migrations
- We will probably run out of our infrastructure budget just running tests for trigger happy devs

1. **Single instance approach**:
- Restore DB from snapshot and use ONE database and run all migrations on that one
- Delete database end of the day and repeat procedure next day
- Schema problems, since it will change after first migration

1. **Single instance approach with ability to timeshift**:
- Restore DB from snapshot and use ONE database, but restore it to point in time afterwards.
- Mingling with restore times is just awful
- Only one developer at a time can create a PR with migration, otherwise they will conflict each other

## Chosen approach

**Multiple instances approach with timeshift to latest**:

- Restore DB from snapshot every day before people come to work
- Keep this instance running at all times, but it can be as small as db.t3.small!
- Do not run any query on this instance, just keep it as state db instance
- Run *aws rds restore-db-instance-to-point-in-time* to clone it for each pull request that has migration
- End of work day, destroy db

**Benefits:**

- Always be able to run on latest production db schema at all times
- We can run as many clusters as we want
- Fast. Usually around 50 seconds for whole migration report to complete (depends on migration)
- Usually other tests on PR take longer
- Relatively cheap, since we destroy instances afterwards and don’t keep them running

## Pre-requirements

1. **Use AWS Backup to create DB snapshots**

- When choosing backup window, give them as little wiggle room as possible for time to perform this backup. Minimums are: start within 1 hr and complete within 2 hours.
- Be sure to pick “Copy to destination” and enter the region you want to keep your additional regional backups
- Optionally if you are brave enough, you can choose to already copy backup into your other account where you will be doing migrations testing.

1. **Use your own KMS keys**

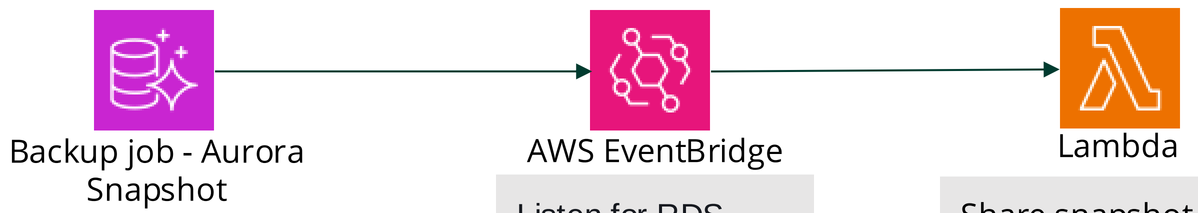
   There will be pain if you decide to go with default AWS KMS keys in the long run. You will probably want to share them with another region or account in the long run, so just generate your own in the first place.

**Side note:** We generate the required KMS keys for all services that we use upon each account provision via StackSets and name them as: our_prefix/service

## Preparation stage

### Usual way:

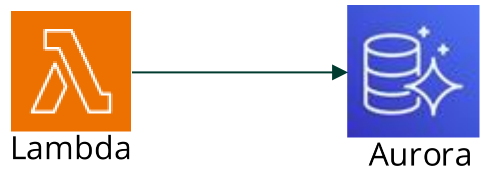
If you chose to copy the snapshot to destination account, you don't need this



Listen for RDS-EVENT-0197 and trigger Lambda

Share snapshot with account you want to do migration testing

### Testing account:

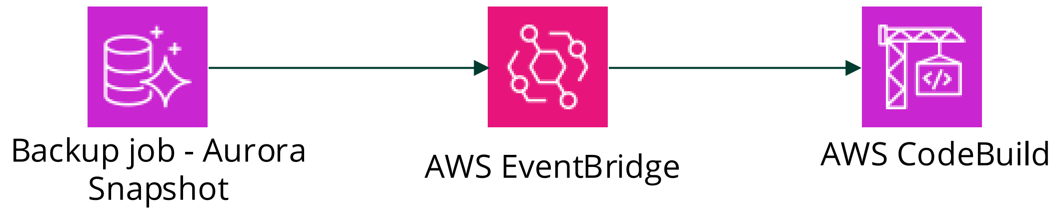


Scheduled Lambda runs 1hr before workday and at end of workday to cleanup

State instance

### Bitstamp way:

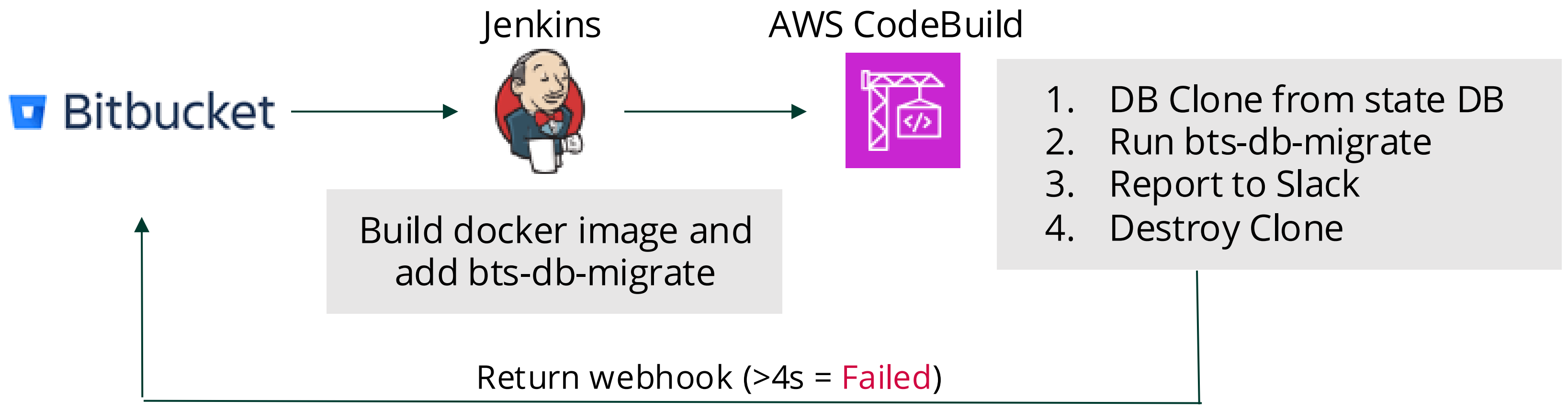
Because we need to do some additional anonymization, before we can share it to our testing account.



What we do in this stage is, run *ansible* that does this:

1. Create DB from snapshot
2. Runs our scripts that anonymize data in tables
3. Create new snapshot from this DB
4. Shares this new snapshot with destination accounts
5. Cleanup

# Migration Test


Bitbucket
Jenkins
AWS CodeBuild
Build docker image and add bts-db-migrate
1. DB Clone from state DB
2. Run bts-db-migrate
3. Report to Slack
4. Destroy Clone
Return webhook (>4s = Failed)

Excerpt of clone command:

```
aws rds --region {{ region }} restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier migrations-report \
  --db-cluster-identifier migrations-report-{{ build_id }} \
  --db-subnet-group-name {{ db_subnet_group.subnet_group.name }} \
  --restore-type copy-on-write \
  --vpc-security-group-ids {{ sg_group.group_id }} \
  --db-cluster-parameter-group-name {{ cluster_parameter_group_name | d("default.aurora-mysql5.7") }} \
  --use-latest-restorable-time
```

**Note:**
Specify **copy-on-write** in **restore-type**, or it won't be a clone!
Default: **full-copy**.

Bitstamp

## 02 Application Section

# Considerations

While doing migrations on production there are quite a few things to be aware of:

- Touching hotpath tables
- Duration of migration / duration of lock on a table
- How long do they lock table
- Support transactions
- Support rollbacks
- Skipping specific migrations (Long migrations, that will be fixed by downtime or gh-ost)
- Dependency tree of migrations
- Interactive mode

Any existing software solutions out there?

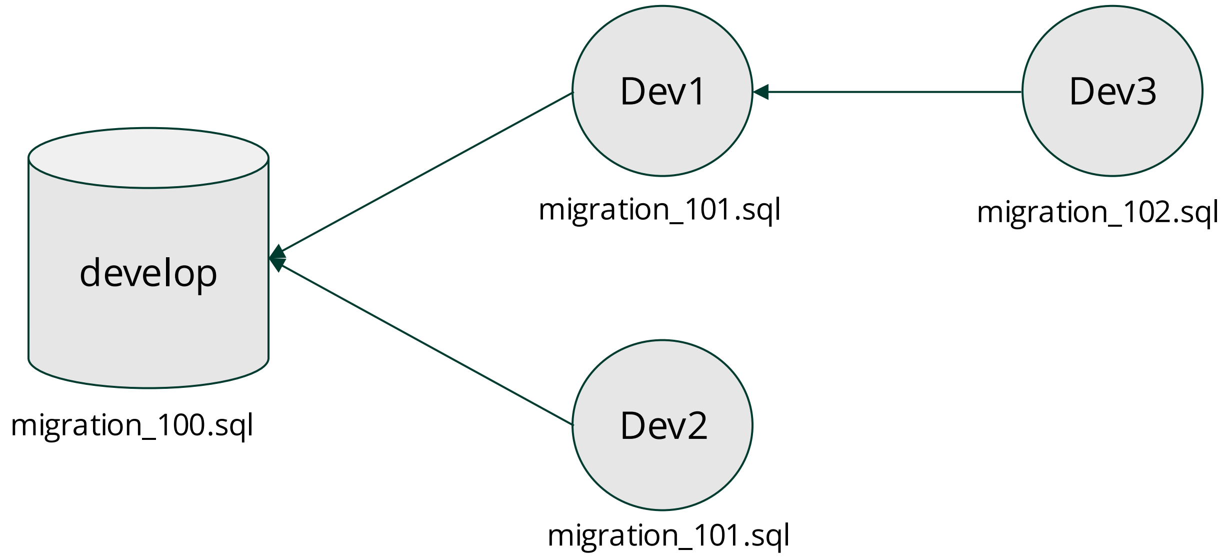
Quite a few, none that really ticks all the checkboxes. Most of them just complicate your life.

Our solution?

**bts-db-migrate** - Initially coded by us, but then taken over by application platform engineering team

## Migration dependencies

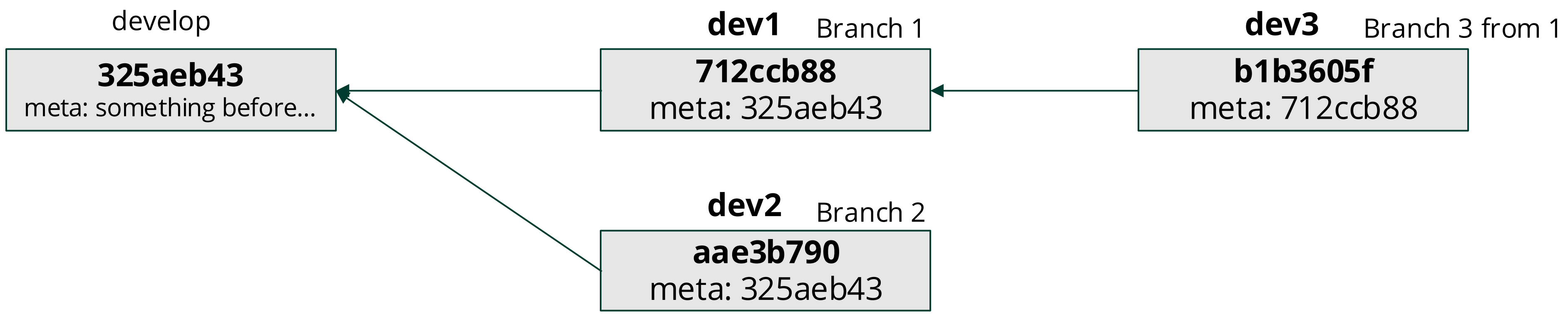
Static naming of migrations is problematic.

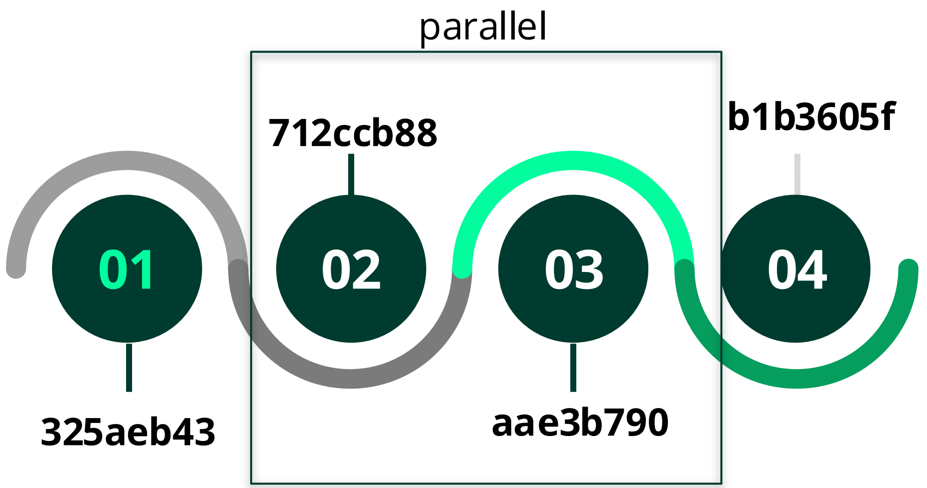


So, when do we merge in dev2’s branch and what impact will this have?

## Our solution

Create a better schema, where migration provides depends_on migration within meta file.





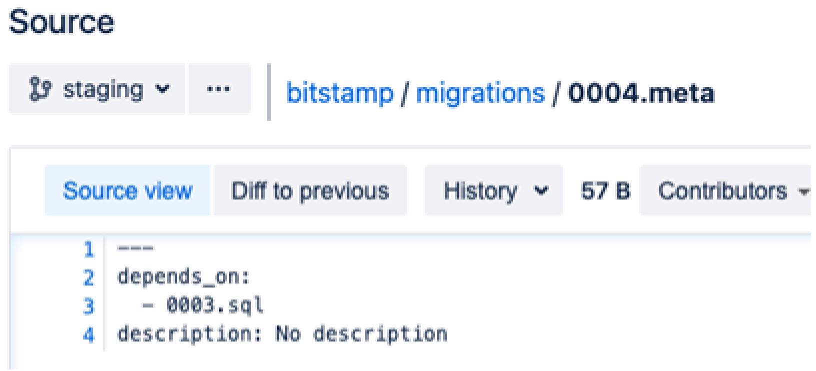
Bitstamp

# bts-db-migrate

These are core functionalities of the tool:

1. Run migration in transaction - so we can roll it back
2. Run specific migration - for debugging or whatnot reasons
3. Skip missing migrations - ability to skip a migration that wasn't applied in the past
4. Generate new migration - developer just fills in the sql
5. Ability to reset our schema to certain migration uuid (usually useful when there is hundreds of migrations piling up and start putting strain on Jenkins, we can decide to do cut and forget about old)
6. Run migrations in batches according to dependencies

**Example of metadata:**

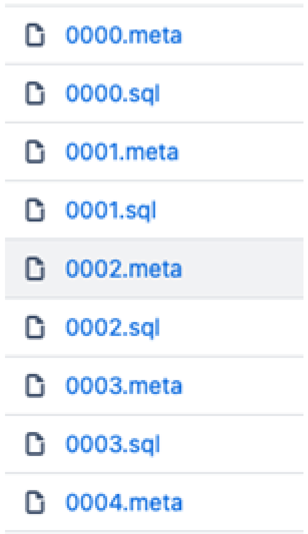


Source

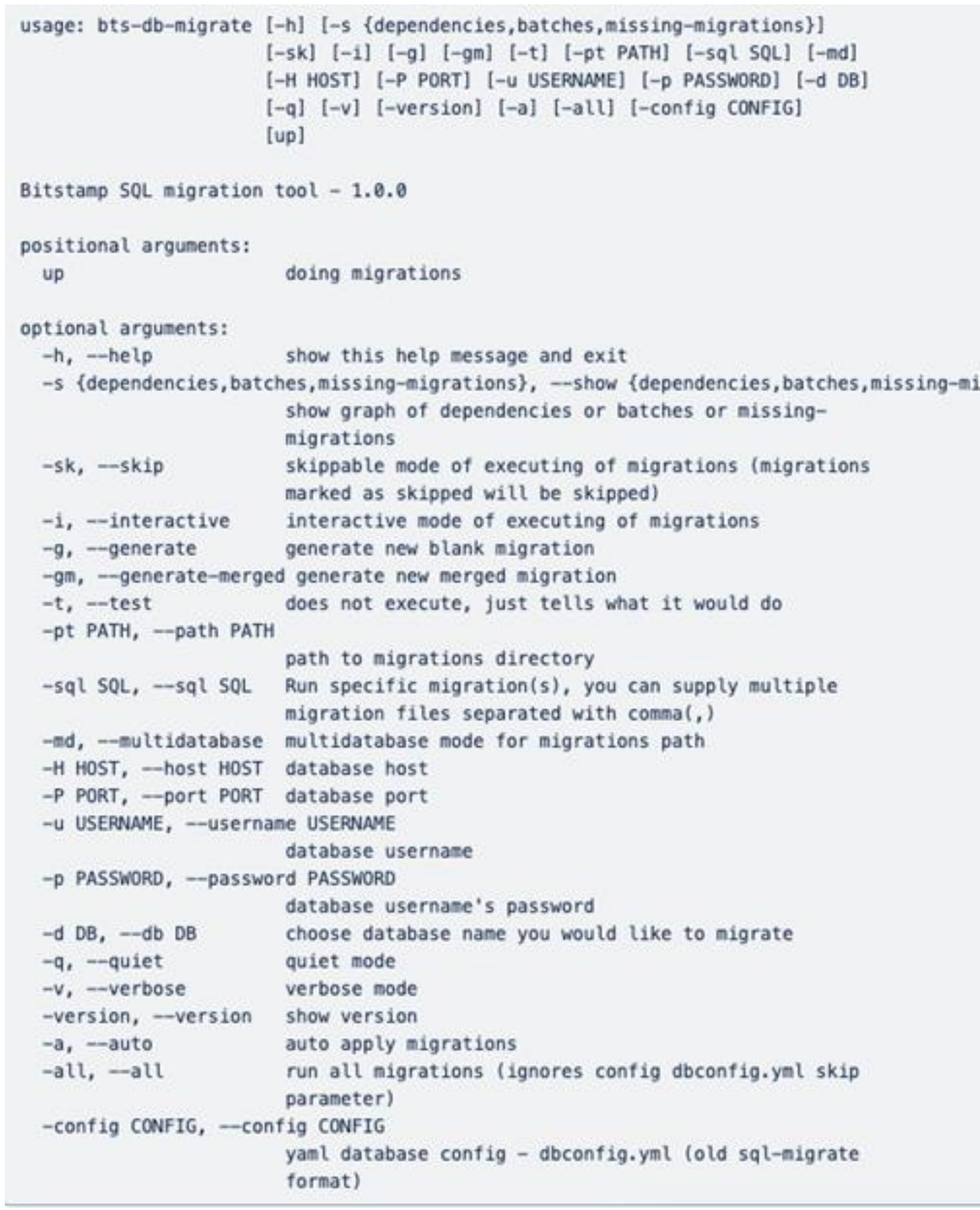
staging ⌄ ··· bitstamp / migrations / **0004.meta**

Source view | Diff to previous | History ⌄ | 57 B | Contributors

```
---
depends_on:
  - 0003.sql
description: No description
```



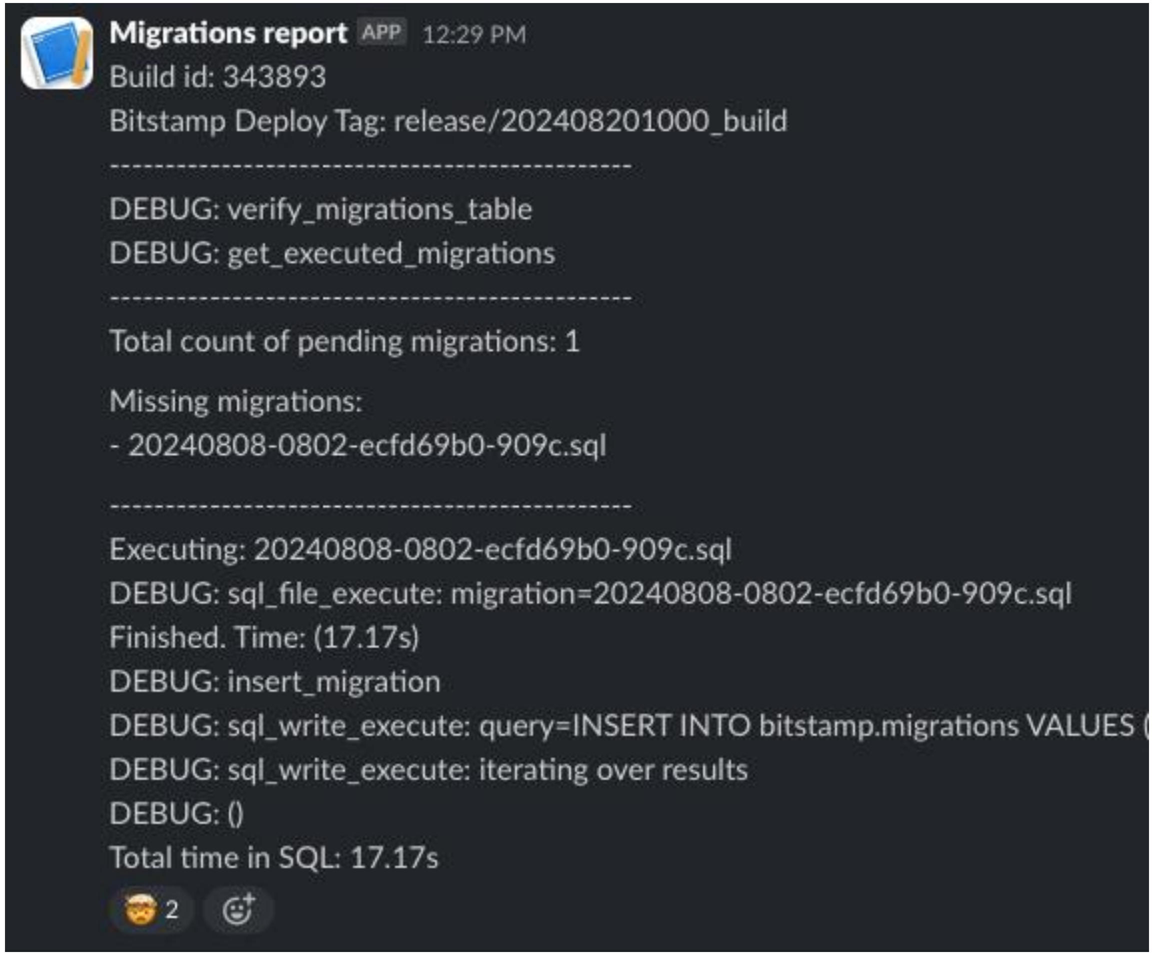
0000.meta
0000.sql
0001.meta
0001.sql
0002.meta
0002.sql
0003.meta
0003.sql
0004.meta



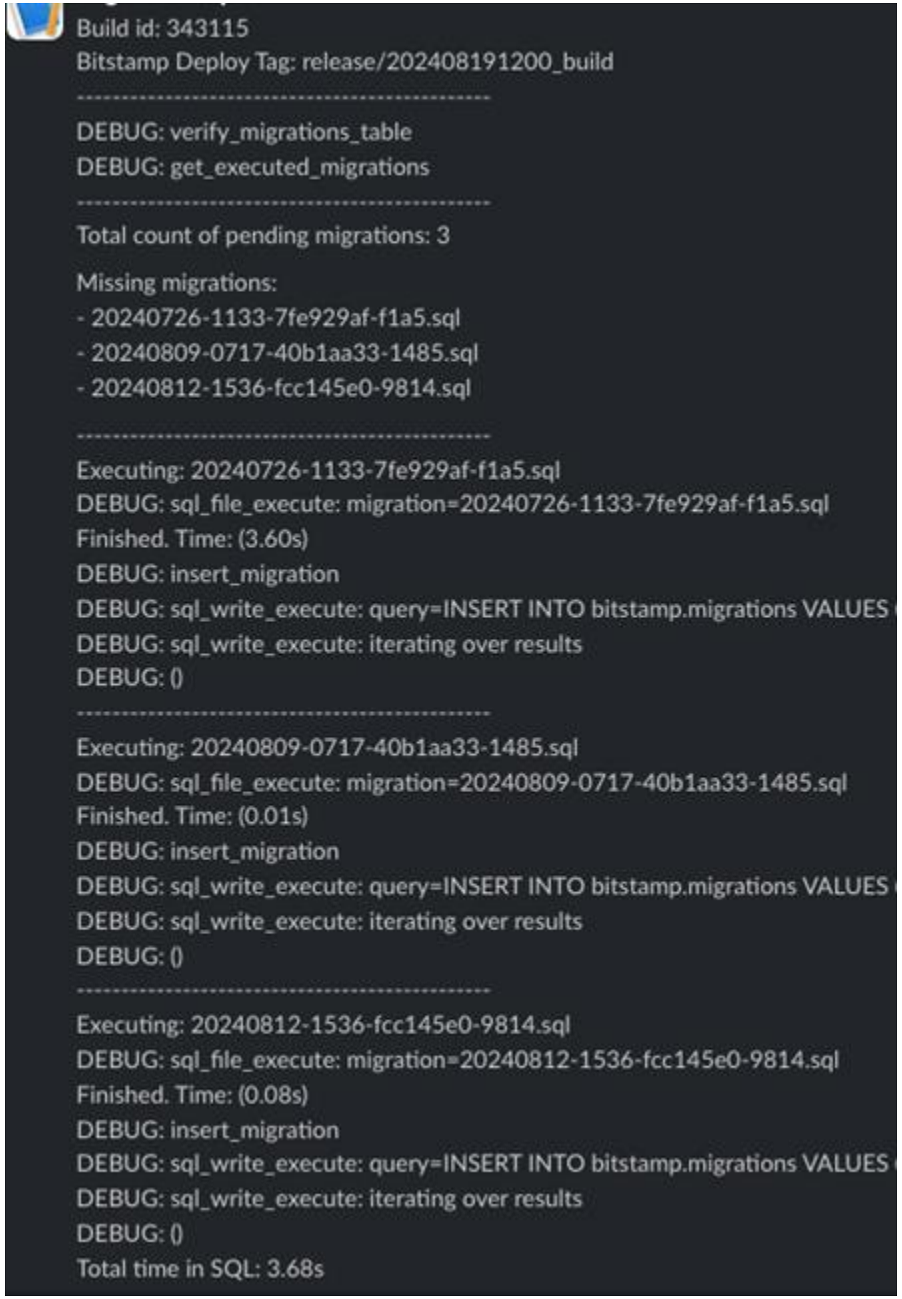
```
usage: bts-db-migrate [-h] [-s {dependencies,batches,missing-migrations}]
                      [-sk] [-i] [-g] [-gm] [-t] [-pt PATH] [-sql SQL] [-md]
                      [-H HOST] [-P PORT] [-u USERNAME] [-p PASSWORD] [-d DB]
                      [-q] [-v] [-version] [-a] [-all] [-config CONFIG]
                      [up]

Bitstamp SQL migration tool - 1.0.0

positional arguments:
  up                    doing migrations

optional arguments:
  -h, --help            show this help message and exit
  -s {dependencies,batches,missing-migrations}, --show {dependencies,batches,missing-mi
                        show graph of dependencies or batches or missing-
                        migrations
  -sk, --skip           skippable mode of executing of migrations (migrations
                        marked as skipped will be skipped)
  -i, --interactive     interactive mode of executing of migrations
  -g, --generate        generate new blank migration
  -gm, --generate-merged generate new merged migration
  -t, --test            does not execute, just tells what it would do
  -pt PATH, --path PATH
                        path to migrations directory
  -sql SQL, --sql SQL   Run specific migration(s), you can supply multiple
                        migration files separated with comma(,)
  -md, --multidatabase  multidatabase mode for migrations path
  -H HOST, --host HOST  database host
  -P PORT, --port PORT  database port
  -u USERNAME, --username USERNAME
                        database username
  -p PASSWORD, --password PASSWORD
                        database username's password
  -d DB, --db DB        choose database name you would like to migrate
  -q, --quiet           quiet mode
  -v, --verbose         verbose mode
  -version, --version   show version
  -a, --auto            auto apply migrations
  -all, --all           run all migrations (ignores config dbconfig.yml skip
                        parameter)
  -config CONFIG, --config CONFIG
                        yaml database config - dbconfig.yml (old sql-migrate
                        format)
```

Bitstamp

# Examples of reports



```
Migrations report APP 12:29 PM
Build id: 343893
Bitstamp Deploy Tag: release/202408201000_build
-----------------------------------------------
DEBUG: verify_migrations_table
DEBUG: get_executed_migrations
-----------------------------------------------
Total count of pending migrations: 1

Missing migrations:
- 20240808-0802-ecfd69b0-909c.sql

-----------------------------------------------
Executing: 20240808-0802-ecfd69b0-909c.sql
DEBUG: sql_file_execute: migration=20240808-0802-ecfd69b0-909c.sql
Finished. Time: (17.17s)
DEBUG: insert_migration
DEBUG: sql_write_execute: query=INSERT INTO bitstamp.migrations VALUES (
DEBUG: sql_write_execute: iterating over results
DEBUG: ()
Total time in SQL: 17.17s
```



```
Build id: 343115
Bitstamp Deploy Tag: release/202408191200_build
-----------------------------------------------
DEBUG: verify_migrations_table
DEBUG: get_executed_migrations
-----------------------------------------------
Total count of pending migrations: 3

Missing migrations:
- 20240726-1133-7fe929af-f1a5.sql
- 20240809-0717-40b1aa33-1485.sql
- 20240812-1536-fcc145e0-9814.sql

-----------------------------------------------
Executing: 20240726-1133-7fe929af-f1a5.sql
DEBUG: sql_file_execute: migration=20240726-1133-7fe929af-f1a5.sql
Finished. Time: (3.60s)
DEBUG: insert_migration
DEBUG: sql_write_execute: query=INSERT INTO bitstamp.migrations VALUES
DEBUG: sql_write_execute: iterating over results
DEBUG: ()
-----------------------------------------------
Executing: 20240809-0717-40b1aa33-1485.sql
DEBUG: sql_file_execute: migration=20240809-0717-40b1aa33-1485.sql
Finished. Time: (0.01s)
DEBUG: insert_migration
DEBUG: sql_write_execute: query=INSERT INTO bitstamp.migrations VALUES
DEBUG: sql_write_execute: iterating over results
DEBUG: ()
-----------------------------------------------
Executing: 20240812-1536-fcc145e0-9814.sql
DEBUG: sql_file_execute: migration=20240812-1536-fcc145e0-9814.sql
Finished. Time: (0.08s)
DEBUG: insert_migration
DEBUG: sql_write_execute: query=INSERT INTO bitstamp.migrations VALUES
DEBUG: sql_write_execute: iterating over results
DEBUG: ()
Total time in SQL: 3.68s
```

## Key takeaways

- Metadata files are awesome (flag hot path affecting migrations)
- Restoring and testing migrations helped us with SOC2 and MIFID II (backup restore testing procedures)
- Don't prewarm table data when evaluating schema migrations (use worst case scenarios)
- Locks during migrations are problematic mostly when on hot path, otherwise not really
- Too long migration? Use gh-ost or schedule a downtime.
- Rollback is not always possible DYOR or… just test it!
- Stopped reading specs that this or that operation became inline and on this mysql version now executes immediately - just test it!
- Developers want to run additional "data amending" queries
    - They are not migrations per se
    - Needs different approach and bunch of additional safety checks

## Last but not least…

**Alen Krmelj**
Infrastructure Team Lead

**LinkedIn** - https://www.linkedin.com/in/alen-krmelj/

**BitNirmata** - https://bitnirmata.com/ - Tech Blog

**Feel free to subscribe to my tech blog rambling :)**

Bitstamp

## Thank You