Making Sense of PostgreSQL Logical Replication

Aiven brings the best Open Source data technologies to all public clouds


Kafka


PostgreSQL


Elasticsearch


Cassandra


Grafana


Redis


InfluxDB


MySQL

Coming soon...

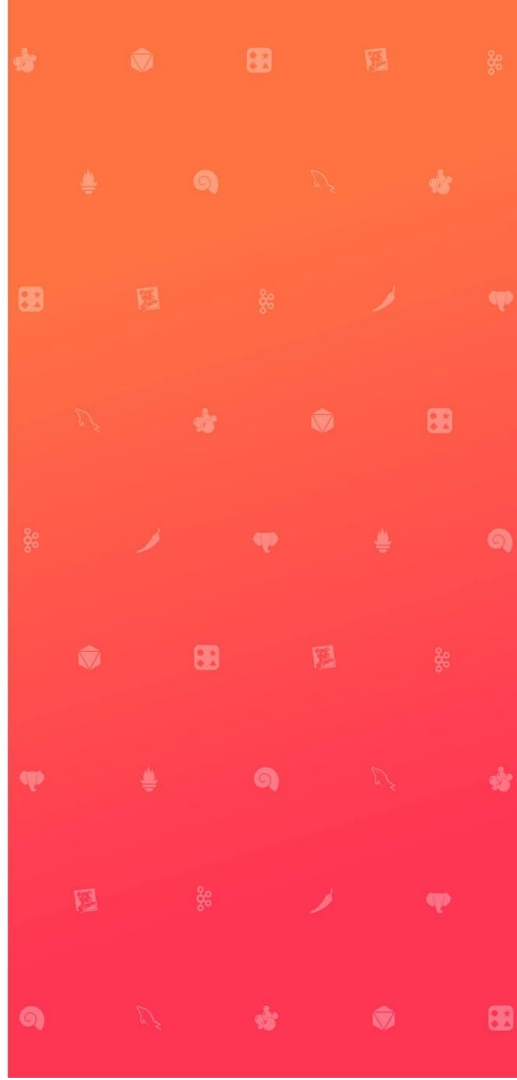

Clickhouse


M3


Flink

Agenda

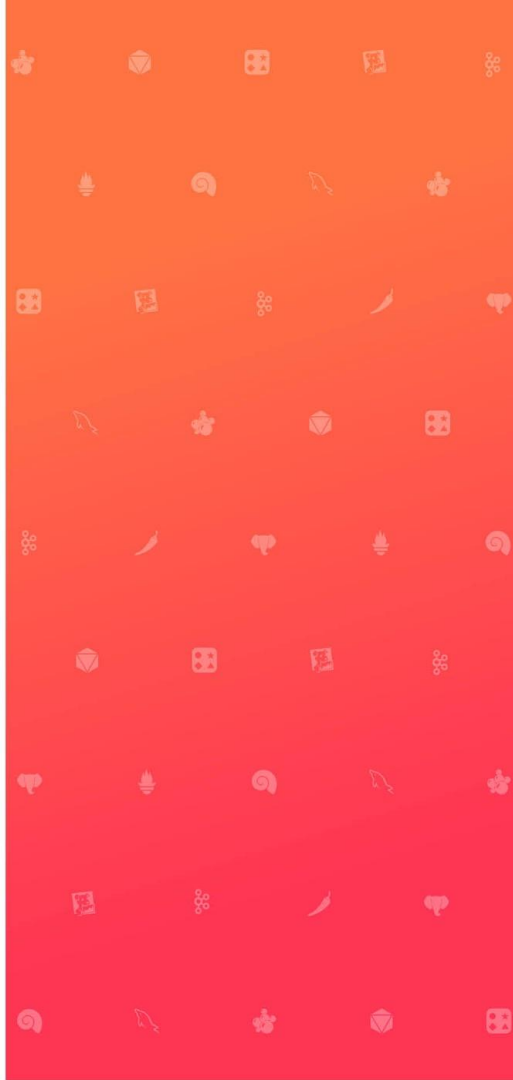
- Background about replication in PostgreSQL
- What is PostgreSQL Logical Replication
- Use cases for logical replication
- CLI example
- Pitfalls and gotchas of Logical Replication
- A quick glance at pg-migrate



Background about replication in PostgreSQL

Postgres Replication can be ...

- Physical
  - Physical Streaming Replication
  - WAL Log Shipping
- Logical
  - Logical Streaming Replication
- Trigger based
  - Slony
- Physical and logical replication are based on WAL, as are most online backup solutions, PITR



Write Ahead Logs

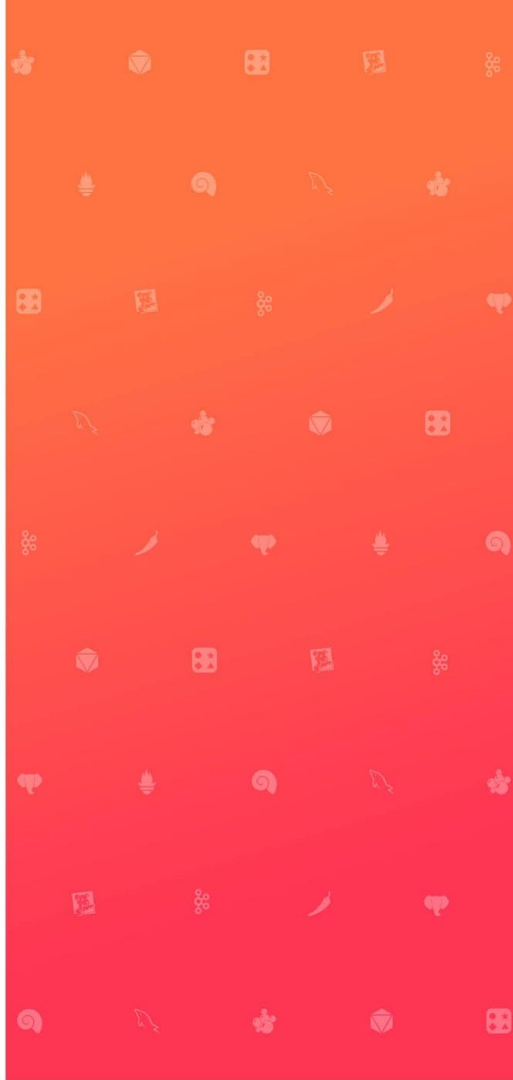
- WAL logs are binary log files, or REDO logs, used in most relational database systems
- A WAL log file consists of several WAL records
- Any transaction performed on a database is first written as a WAL record, only then written to the tables' data files
- Safeguards against data corruption with partial writes (eg power failure)
- Can be replayed from a checkpoint to restore to a current or previous state (PITR)

WAL levels

- The amount of information written to WAL files is controlled by the **wal_level** setting -
- **minimal** - faster, skips some bulk operations, not enough information to reconstruct the data from a base backup - no replication/archiving
- **replica** (default) - allows for physical replication, archiving
- **logical** - the same data as replica, plus the information needed for logical replication

Physical replication

- Before PG 10, built-in replication always happened using physical replication
- Based on WAL (Write Ahead Logs), block level changes sent to the replicas
- The replica is a bit-by-bit copy of the master, the whole database and all changes are always replicated
- Used for High-Availability and read scaling



WAL log shipping - warm standby

- A form of physical replication
- First method of replication in PG (ver 8.2 in 2006)
- WAL files are transferred to an archive with **archive_command** (eg. cp, s3cmd, gsutil cp)
- The log files are sent and processed once the WAL file is filled with enough records
- The replica processes these WAL files
- No synchronous replication
- Supported by eg. PgBarman, WAL-E, pghoard

Streaming Replication

- Introduced in PostgreSQL 9.0 (September 2010)
- Allows the replica to stay more up to date vs the log shipping
- This is because WAL records are sent directly to the replica (vs waiting for the WAL file to fill with enough records)
- Allows for synchronous replication (since 9.1)
- The records are sent directly over the network by the master's wal_sender to the replica's wal_receiver
- Can be used alongside log shipping

Physical replication recap

- Used for HA and read scaling
- The replica is always a bit-by-bit copy of the master
- Everything is replicated
- The replica is always read-only
- The replica has to have the same Postgres version as the master
- The replica has to be on the same HW architecture as the master

What is PostgreSQL's Logical Replication?

And what added value it brings vs Physical Replication

Logical replication

- Logical Replication was first introduced in PostgreSQL 10 (Oct 2017)
- A method of replicating data objects and their changes based on a replication identity (usually a primary key)
- Uses a publish / subscribe model
  - Subscribers subscribe to one or more publications on a publisher node
  - Subscribers can re-publish data to further subscribers

What this means in practise

- Processes the WAL records and decodes them to logical changes
- Instead of having a record of what bytes change in a datafile - we now know what rows change in what tables
- The changes are applied on the replica as logical INSERT, UPDATE, DELETE (and since PG 11, TRUNCATE) changes in transactional order
- We can filter changes based on their type, or the tables they affect

How it works - Pub / Sub

- A publication is defined on the master
  - Applied to one or more tables
  - Can filter based on change types
- A subscription is defined on the replica server
  - The subscriber can also publish to further subscribers
- When a subscription is created it copies a snapshot of the published tables, then changes on the publisher are sent to the subscriber in real-time

Replication identity

- Replication is done based on a “Replication Identity”
  - In practice this is usually the primary key
  - Can be any unique index
  - Finally it can be set to “FULL” which makes the whole row the key

Replica writability

- Unlike with streaming replication, the replica is writable
- Does not imply multi-master
  - Bi-directional replication is not supported
- The destination table schema can differ from the source, but needs to exist
  - Can have more columns
  - Column order doesn’t matter
  - The changes need to be able to go through

Logical replication use cases

Replication of a subset of a database

- If you wish, you can choose to replicate only certain tables from a database
  - Not possible in streaming replication
- This is more economic and performant
- The replica table is writable
  - You can add indexes, views
  - Can add columns
  - The replica's security definition can be different

Database consolidation

- Replicating select tables from several databases to a single database
- Great for analytics
- The replica can have different security definitions - you can allow wider access on the consolidated DB

Online migration

- Online database migration
- Essentially -
  - Start replication
  - Wait for replication to finish
  - Point your clients at the destination
- No master promotion involved
- Possibility to upgrade PG major versions

Replicating between different types of clusters

- Logical replication is possible between different versions* of Postgres
- Allows for replication between different platforms:
  - Windows to Linux
  - x86 to ARM

CLI example

Simple replication of one table

Set wal_level to logical

- Enable wal_level = logical on both the *source* and *destination* databases
- Restart the clusters for changes to take effect

```
$ ALTER SYSTEM SET wal_level = logical;

$ show wal_level;
 wal_level
-----------
 logical
(1 row)
```

aiven

Create some example data

- Create a source database
- In it create a table called test_table and add some test values

```
postgres=# CREATE DATABASE source;
postgres=# \c source

source=# CREATE TABLE test_table (id INT);
CREATE TABLE
source=# INSERT INTO test_table VALUES (1), (2);
INSERT 0 2
```

Create a publication

- On the source cluster, create a publication for test_table

```
source=# CREATE PUBLICATION pub1 FOR TABLE test_table;
CREATE PUBLICATION
```

aiven

Now on to the destination DB

- We need to have the schema defined on the destination, here we do it by hand, but you could use eg. pg_dump --schema-only

```
postgres=# \c destination
You are now connected to database "destination" as user
"postgres".

destination=# CREATE TABLE test_table (id INT);
CREATE TABLE
```

Create a subscription

- Now we create a subscription on the destination
- Make sure that you can reach the source postgresql server port from the destination

```
destination=# CREATE SUBSCRIPTION sub1
  CONNECTION 'host=192.168.122.33 port=5432 dbname=source'
  PUBLICATION pub1;
NOTICE:  created replication slot"sub1" on publisher
CREATE SUBSCRIPTION
```

Replication is now running!

- You can check the contents of the destination table and you should now find the test data there

```
destination=# SELECT * FROM test_table;
 id
----
  1
  2
(2 rows)
```

aiven

Replication status

- Check the status of a replication from pg_stat_subscription and pg_replication_slots

```
source=# SELECT confirmed_flush_lsn, pg_current_wal_lsn(),
  (pg_current_wal_lsn() - confirmed_flush_lsn)AS lsn_distance
  FROM pg_catalog.pg_replication_slots
       WHERE slot_name = 'sub1';
 confirmed_flush_lsn | pg_current_wal_lsn | lsn_distance
---------------------+--------------------+--------------
 0/167B708           | 0/167B708          |            0
(1 row)
```

aiven

Cleaning up

- If you are doing a migration you will want to stop the replication
- Remember to clean up unused replication slots

```
destination=# DROP SUBSCRIPTION sub1;
NOTICE:  dropped replication slot"sub1" on publisher
DROP SUBSCRIPTION
```

aiven

Logical replication towards Aiven PostgreSQL

- There is a help article about logical replication towards Aiven PostgreSQL with more examples
- The article also explains how to use the aiven-extras on the Aiven side for access to the superuser

Logical replication Pitfalls and gotchas

Things to keep in mind when using logical replication

Does not support DDL changes

- Logical replication does not support schema changes
- In DB migrations you will typically copy the schema first, then start replication
- You are also able to change the schema by hand
  - Eg. when adding a column, you can add it to the replica, then the master

Schema change example workaround

- Pause replication
- Migrate destination
- Migrate source
- New tables need to be added separately to the publication with ALTER PUBLICATION
- Refresh the subscription
- Resume replication

Sequences can be tricky

- Start values of sequences aren't transparently replicated
- A write to the destination can cause sequence collisions
- Workarounds include
  - Using non-overlapping ranges for auto-increments
  - Using external sources for the values (etcd, ZooKeeper)

No primary key?

- Logical replication needs a replication identity
- Usually a primary key - if it doesn't exist:
  - Use another unique index of the table
  - Use the full row as the replication identity
  - Or add a primary key to the table

Material about the previous cases

- A great article about the DDL changes, sequences, replication identity issues with example workarounds at PGDash

Large objects

- Postgres objects of type “large object” (BLOB) can’t be replicated
- Instead of using large objects you can reference objects on object store or network storage (GCS, S3, NFS, EFS)
- For smaller data, you can store the data in the tables themselves

Partitioned tables

- Partitioned tables are not currently (<= PG 12) fully supported
- You can however replicate the child tables by replicating them explicitly
- Partition hierarchies need to be the same
- Support for logical replication in partitioned tables coming in Postgres 13

Extensions

- Be careful with extensions when using Logical Replication - version mismatches or missing extensions can cause problems
- Require whitelisting (eg. pgextwlist) or superuser-like access to enable
- Some extensions don’t work with logical replication
    - eg. TimescaleDB is not supported

Superuser access

- Some of the operations, eg publishing ALL TABLES require superuser privileges
  - Not a problem when you have superuser access
  - More of an issue on managed platforms
- Aiven addresses this through the open source “aiven-extras” extension
- AWS RDS has a custom rds_replication role which allows for replication
- GCP CloudSQL and Heroku PG don’t support logical replication at all

aiven-db-migrate

An open source migration toolkit in the works

aiven-db-migrate

- An Aiven built open source tool in Python 3.5+ for migrating databases
- Currently in development, first release will target PostgreSQL
- Plans to expand to cover also other DBs eg. MySQL
- Not restricted to Aiven products, can also be used for migrations between non-Aiven DBs

aiven-db-migrate - PostgreSQL

- Supports logical replication
- Does sanity checking on the source and target
  - Versions compatibility
  - Extensions
  - Permissions
- Also supports pg_dump as a fallback
- Supports aiven-extras for admin level out of the box (easy migrations to Aiven)

aiven-db-migrate - PostgreSQL

- To be integrated to the Aiven platform for migrations
- Under active development - more switches and functionality to be added
- To be open sourced as a standalone tool
  - Once released - contributions are always appreciated
- Blog post will follow

aiven-db-migrate - example

```
$ pg_migrate

--source
"postgres://postgres:xxxx@noisedata-public.aaaaa.eu-west-1.rds.amazonaws.com:5432"

--target
"postgres://avnadmin:xxx@noisedata-sandbox.aivencloud.com:12691/defaultdb?sslmode=require"

--wait-until-replication-in-sync
```

aiven

Final words

Final words

- Logical Replication allows for lots of use cases and a variety of new setups
  - Partial replication
  - Multiple sources to one target
  - Replication between versions
  - Online upgrades and migrations
- It also involves a lot of moving parts so always plan and test your migrations as well as possible

Thank you for your time!

Q & A

Questions?

Links

- “Postgres 12 Documentation : Logical Replication“
  https://www.postgresql.org/docs/12/logical-replication.html
- https://pgdash.io/blog/postgres-replication-gotchas.html
- “Partitioned tables can now be replicated …”
  https://amitlan.com/writing/pg/partition-logical-replication/
- “Aiven-extras extension” https://github.com/aiven/aiven-extras
- “Using logical replication to replicate tables, or a whole database to Aiven PostgreSQL”
  https://help.aiven.io/en/articles/2464717-postgresql-logical-replication