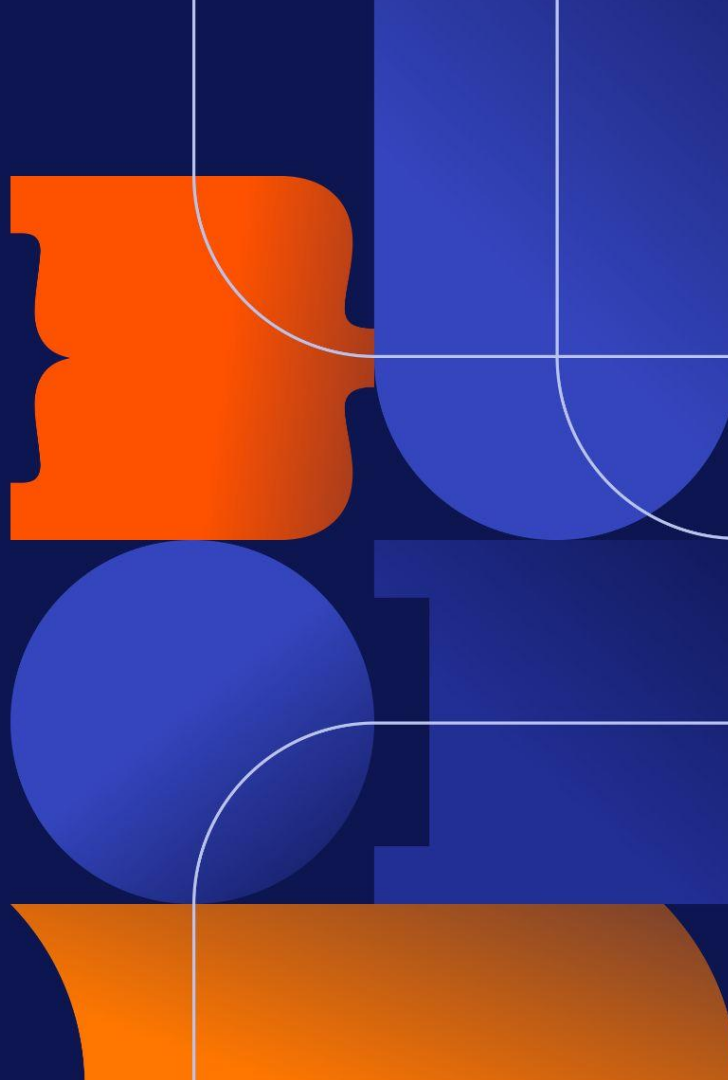
Logical replication - for fun and profit

Patrick Stählin, Aiven



Agenda

01. **History of replication**

How did we end up with replication in PostgreSQL?

02. **What is logical replication**

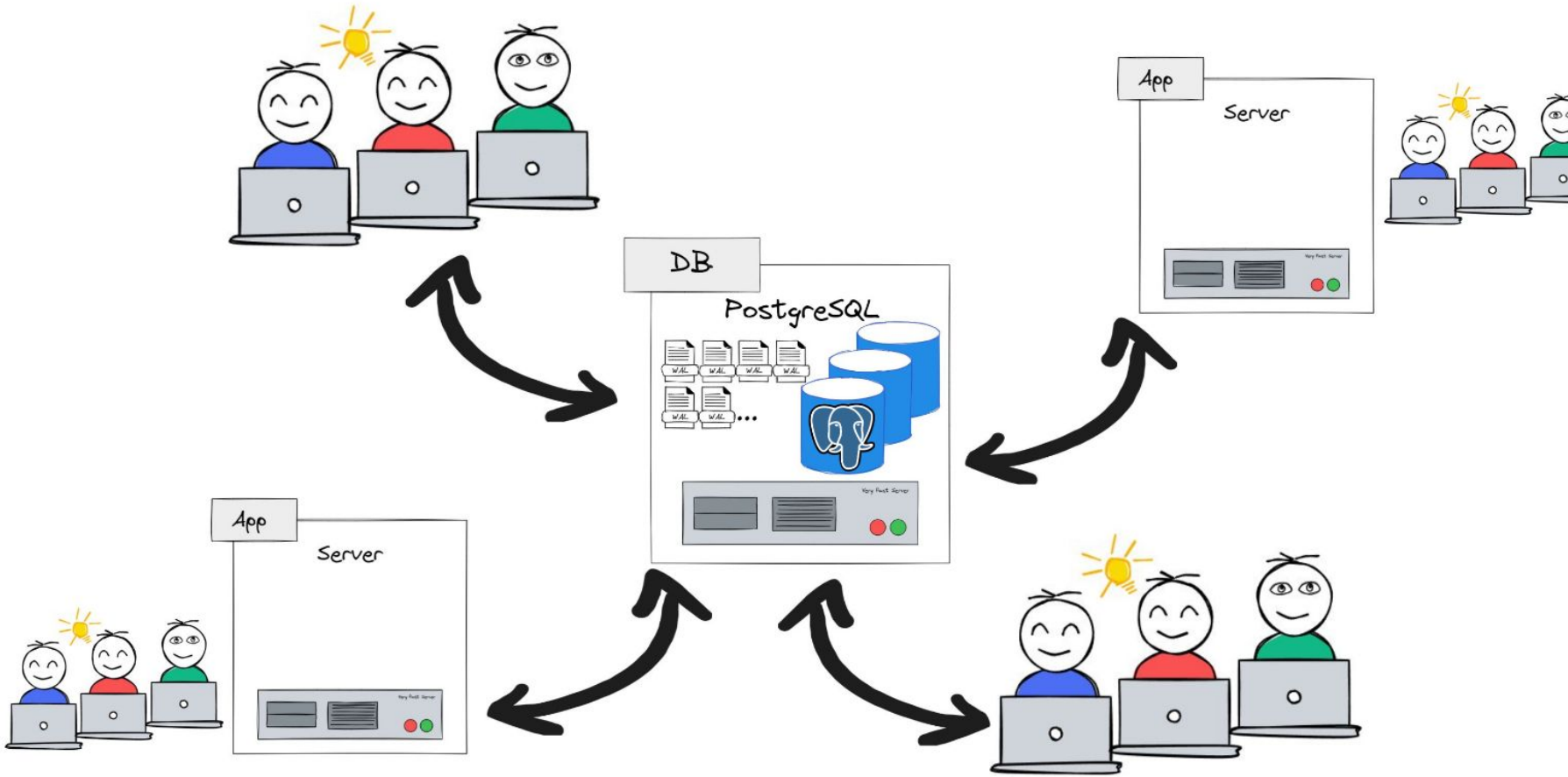
Short primer on how it all works.

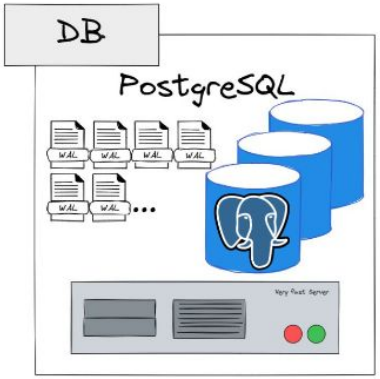
03. **Trying it out**

Quick demo

04. **Profit?**

Showcasing a couple of use-cases.


App
Server
DB
PostgreSQL
WAL
App
Server



DATA MIGRATION

What is replication?

“Replication [...] refers to maintaining multiple copies of data, processes, or resources to ensure consistency across redundant components”

https://en.wikipedia.org/wiki/Replication_(computing)

Replication

- Physical
- Logical

How did we get here

- PG 7.1 introduced the write-ahead-log (WAL) in 2001
- PG 8.0 added point-in-time-recovery (PITR) in 2005
- PG 8.3 “replication” with pg_replay in 2008
- PG 9.0 streaming replication/hot-standby in 2010
- PG 9.4 replication slots/logical replication in 2015
- PG 10.0 publication/subscription support in 2017

Source: https://peter.eisentraut.org/blog/2015/03/03/the-history-of-replication-in-postgresql, PG changelogs

How it works

- Publication
- Subscription
- (replication slot)

Publication

- Per database
- Describes which tables/columns are published. Plus optional filters on rows (WHERE condition).
- Can be multiple/all tables

Publication

```
CREATE PUBLICATION name
    [ FOR ALL TABLES
      | FOR publication_object [, ... ] ]
    [ WITH ( publication_parameter [= value] [, ... ] ) ]
```

where *publication_object* is one of:

```
    TABLE [ ONLY ] table_name
          [ * ]
          [ ( column_name [, ... ] ) ]
          [ WHERE ( expression ) ]
          [, ... ]
    TABLES IN SCHEMA { schema_name | CURRENT_SCHEMA } [, ... ]
```

https://www.postgresql.org/docs/current/sql-createpublication.html

Subscription

- Per database
- Describes how to connect to the publication
- Lots of options

Subscription

```
CREATE SUBSCRIPTION subscription_name
    CONNECTION 'conninfo'
    PUBLICATION publication_name [, ...]
    [ WITH ( subscription_parameter [= value] [, ... ] ) ]
```

https://www.postgresql.org/docs/current/sql-createsubscription.html

Important subscription parameters

- `create_slot / slot_name`
- `binary`
- `copy_data`
- `streaming`
- `origin`
- `failover`

Replication slot

- Blocks deletion of WAL files
- Is usually created automatically (unless you prevent it)
- Manual creation:

```
SELECT * FROM pg_create_logical_replication_slot(
    'sub1', 'pgoutput'
);
```

WAL sender

- Handles both physical and logical replication
- Keeps track of transactions
- Sends completed* transactions to a output plugins

WAL sender: output plugins

- pgoutput (default)
- test_decoding
- wal2json
- decoderbufs
- ...

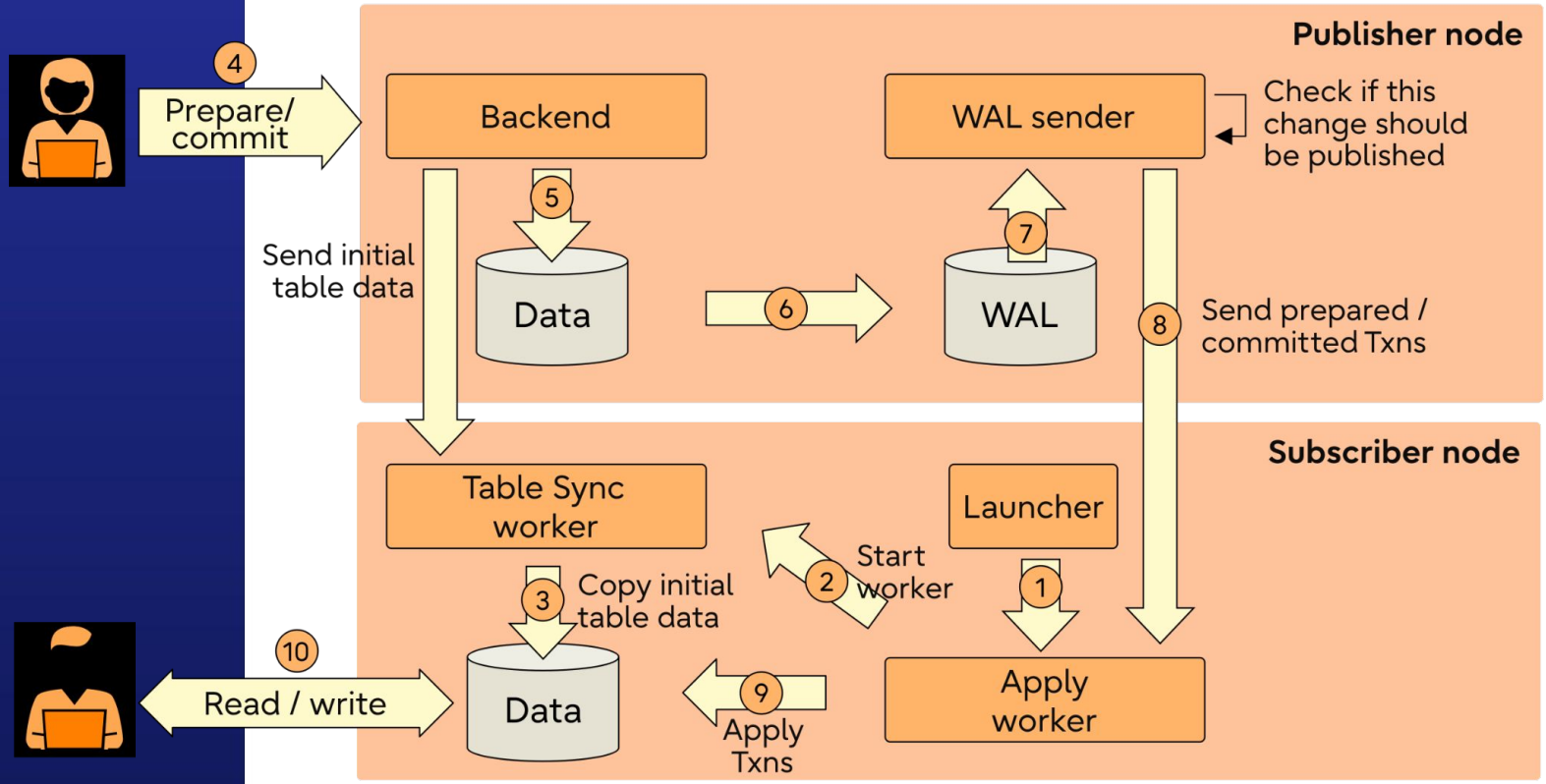
WAL file

- Consists of records, “byte-changes in pages”
- With `wal_level = logical`, the WAL file is augmented with records on what really changed

```
$ /usr/pgsql-16/bin/pg_waldump
testdb6001/pg_wal/000000010000000000000002
rmgr: Sequence   desc: LOG rel 1663/5/16384, blkref #0: rel
1663/5/16384 blk 0
rmgr: Heap       desc: INSERT+INIT off: 1, flags: 0x08,
blkref #0: rel 1663/5/16385 blk 0
rmgr: Btree      desc: NEWROOT level: 0, blkref #0: rel
1663/5/16391 blk 1, blkref #2: rel 1663/5/16391 blk 0
rmgr: Btree      desc: INSERT_LEAF off: 1, blkref #0: rel
1663/5/16391 blk 1
rmgr: Transaction desc: COMMIT 2025-06-24 17:01:46.642249
CEST
rmgr: XLOG       desc: SWITCH
```

Apply worker

- Creates tablesync workers for the initial synchronization
- Once synced up, it applies changes sent to it



https://www.postgresql.fastware.com/blog/inside-logical-replication-in-postgresql

Demo!

- `CREATE TABLE beers(id SERIAL PRIMARY KEY, name VARCHAR);`
- `INSERT INTO beers(name) values ('Feldschlösschen');`
- `CREATE PUBLICATION beer_pub FOR TABLE beers;`

- `CREATE SUBSCRIPTION beer_sub CONNECTION '...' PUBLICATION beer_pub;`

- `INSERT INTO beers(name) values ('Quöllfrisch');`

How does it work

```
 170649 ?      Ss     0:00  |   \_ /usr/pgsql-16/bin/postgres -D testdb6001
 170650 ?      Ss     0:00  |        \_ postgres: checkpointer
 170651 ?      Ss     0:00  |        \_ postgres: background writer
 170653 ?      Ss     0:00  |        \_ postgres: walwriter
 170654 ?      Ss     0:00  |        \_ postgres: autovacuum launcher
 170655 ?      Ss     0:00  |        \_ postgres: logical replication launcher
 170657 ?      Ss     0:00  |        \_ postgres: patrick.staehlin postgres ::1(33850) idle
 176575 ?      Ss     0:00  |        \_ postgres: walsender patrick.staehlin postgres
[local] START_REPLICATION
```

How does it work

```
 176149 ?        Ss     0:00    \_ /usr/pgsql-16/bin/postgres -D testdb6002
 176150 ?        Ss     0:00          \_ postgres: checkpointer
 176151 ?        Ss     0:00          \_ postgres: background writer
 176153 ?        Ss     0:00          \_ postgres: walwriter
 176154 ?        Ss     0:00          \_ postgres: autovacuum launcher
 176155 ?        Ss     0:00          \_ postgres: logical replication launcher
 176157 ?        Ss     0:00          \_ postgres: patrick.staehlin postgres ::1(38860) idle
 176574 ?        Ss     0:00          \_ postgres: logical replication apply worker for
subscription 16394
```

Caveats

- Monitor your replication slots
- DDL statements are not replicated
- Replication slots are not replicated (prior to PG17)
- Replication slots are dropped during major upgrades (prior to PG17)
- No re-mapping of columns or tables (names/types must match)
- Tables without primary keys require REPLICA IDENTITY FULL
- Initial COPY can consume a huge amount of resources
- Large transactions can be an issue (mostly pre PG14)
- Sometimes still rough around the edges

Showcase 1: CDC

- Capture data for 3rd party systems
- Leave GDPR/DSG relevant information in one system
- Put changes in a queue for processing

Showcase 2: Analytics

- Main database needs different indices
- Specialized indices are needed for faster analytics
- ⇒ Schemas can differ so we can have different indices

Showcase 3: Upgrade in stages

- Major database upgrade can be risky
- Logical replication works between different versions
- ⇒ You can replicate parts of your data a new PG version and route your requests there

Showcase 4: Zero downtime PG upgrade

- PG upgrades are scary
- Especially if you can’t afford downtime

Showcase 4: Zero downtime PG upgrade

- You need a load-balancer/pooler
- Two servers, primary and standby
- pg_createsubscriber - new in PG17
  - Creates a logical replica from a physical one
  - You need access to the data-directory
- pg_ctl stop ... standby
- pg_upgrade ... standby
- Take backup
- Initialize new standby from backup

Thank you!

Please monitor your replication slots!

Patrick Stählin

PostgreSQL team

Senior Software Engineer, Aiven

patrick.staehlin@aiven.io

https://www.linkedin.com/in/patrickstaehlin/

@packi.ch (Bsky)

start_server.sh

```
#!/bin/bash
VERSION=$1
PORT=$2

/usr/pgsql-$VERSION/bin/initdb testdb$PORT
printf "\nport = $PORT\nwal_level =
logical\nunix_socket_directories = '/tmp'" >>
testdb$PORT/postgresql.conf
/usr/pgsql-$VERSION/bin/pg_ctl -D testdb$PORT -l logfile$PORT start
/usr/pgsql-$VERSION/bin/psql -h localhost -p $PORT postgres
```