PostgreSQL Berlin March Meetup

Tuesday, March 5, 2024

zalando

Agenda

- **19:00 - 19:10** - "*Introduction to PostgreSQL at Zalando*" by Matthias Adler, Team Lead Database as a Service, Zalando SE
- **19:15 - 19:45** "*PostgreSQL worst practices*" by Ilya Kosmodemiansky, CEO of dataegret.com
- **19:50 - 20:20** "*Peculiarities of Logical Decoding in PostgreSQL*" by Polina Bungina, Senior Database Engineer, Zalando SE
- **20:20** - Q&A, networking, and Pizza & Drinks
- **22:00** - The End

Introduction to PostgreSQL at Zalando

Tuesday, March 5, 2024

Matthias Adler
Team Lead Database as a Service,
Zalando SE

PostgreSQL at Zalando

- More than **3000** database clusters on Kubernetes
- Support for PostgreSQL 12 - 16
- Backups: *wal-e* (archiving) *wal-g* (restoring)
- Notable extensions
  - Monitoring: *bg_mon*
  - Query statistics: *pg_stat_statements*
  - Partitioning: *pg_partman* vs. *Timescale*
  - LLM embeddings: *pgvector*
- Core Contributions
  - Submitting and reviewing patches for PostgreSQL and maintenance of FOSS projects

Patroni

- Industry's high availability solution for PostgreSQL
- Management of PostgreSQL fleets
  - Leader election
  - Manages PostgreSQL configuration
  - Bootstraps PostgreSQL instances
  - Handles K8s API server outages
- Recent improvements in code quality
  - Completely Dropping Python 2.x
  - Adding type hints
  - Attracted new contributors

https://github.com/zalando/patroni

zalando

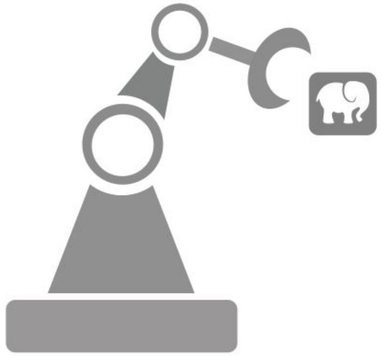
Spilo

- The complete bundle: PostgreSQL and Patroni
- Important utility scripts
  - WAL and log shipping to AWS S3
  - User management for employees
  - Metric helpers for pgView and monitoring
  - Automatic major version upgrades
- Other container images
  - PostgreSQL Docker image for local development
  - Flyway migration template
  - SQL export to AWS S3

https://github.com/zalando/spilo

zalando

Postgres-Operator



- K8s operator that provisions PostgreSQL ("Spilo") clusters
- Designed to help engineers and teams
  - Tuning capabilities (CPU, Memory, IOPS etc.)
  - Automation for database roles and privileges
  - Easy access for owning teams
  - UI to inspect manifest, backups & operator logs
- Clone functionality to easily test in production
  - Testing major version upgrade
  - Query performance analysis

https://github.com/zalando/postgres-operator

zalando

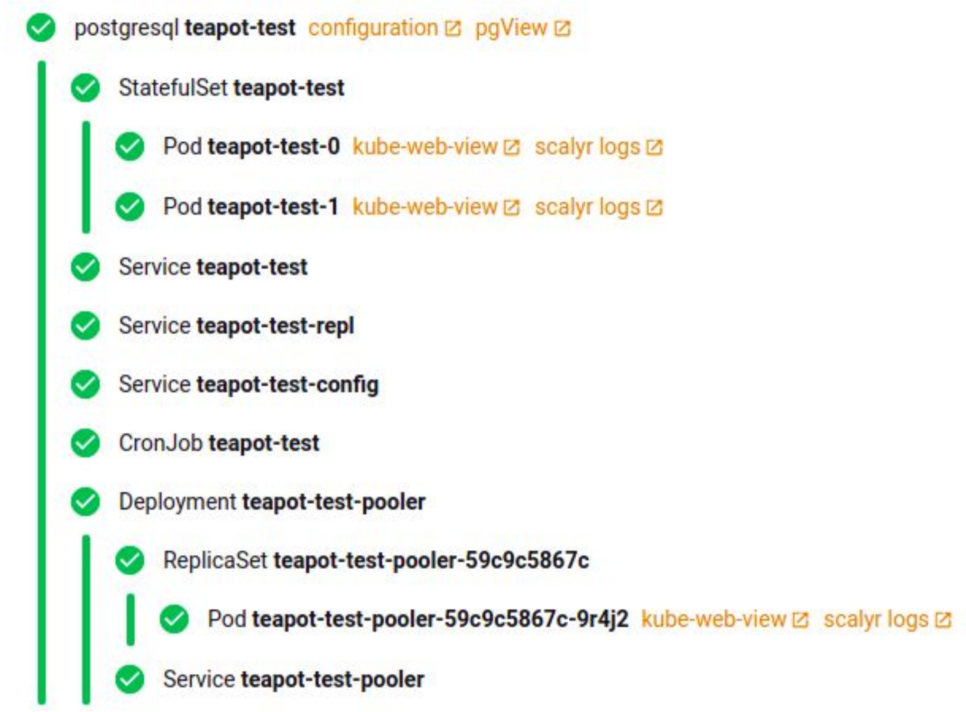
PG-View

- Live monitoring of PostgreSQL clusters
- Users could observe
  - High CPU load and OOM exceptions
  - Running out of free disk space
  - Replication delay and stale WAL consumers
  - Long running and/or blocking queries
- Database browser
  - Schema and table view (to check indexes)
  - Statement statistics (and its history)
  - User logins

Self-Service Database on K8s

```yaml
apiVersion: acid.zalan.do/v1          # Kubernetes Custom Kubernetes Resource (CRD) definition
kind: postgresql
metadata:
  name: teapot-test                    # pick a PostgreSQL cluster name and namespace
  namespace: default
spec:
  numberOfInstances: 2                 # 2 = 1 primary, 1 streaming replica
  enableConnectionPooler: true         # Enable connection pooler (using PGBouncer)
  postgresql:
    version: "15"                      # major version, upgradable by you
  teamId: teapot                       # team members automatically added to database
  users:
    app_user: []                       # application user with privileges to own database
  databases:
    app_db: app_user
  volume:
    size: 50Gi                         # volume size, increasable
```

Continuous Delivery Platform



postgresql **teapot-test** configuration pgView

- StatefulSet **teapot-test**
  - Pod **teapot-test-0** kube-web-view scalyr logs
  - Pod **teapot-test-1** kube-web-view scalyr logs
- Service **teapot-test**
- Service **teapot-test-repl**
- Service **teapot-test-config**
- CronJob **teapot-test**
- Deployment **teapot-test-pooler**
  - ReplicaSet **teapot-test-pooler-59c9c5867c**
    - Pod **teapot-test-pooler-59c9c5867c-9r4j2** kube-web-view scalyr logs
  - Service **teapot-test-pooler**

What you get ?

- Manifest validation and delete protection
- In-place major version upgrade
- HA with automatic leader election and failover
- Continuous backups with point-in-time-recovery (up until last 5 days)
- User provisioning, OAuth2 access for employees with SSO integration
- Databases with pre-configured role setup
- And … built-in monitoring with PG-View & Grafana, etc.

Break

PostgreSQL worst practices

Tuesday, March 5, 2024

Ilya Kosmodemiansky
CEO of dataegret.com

zalando

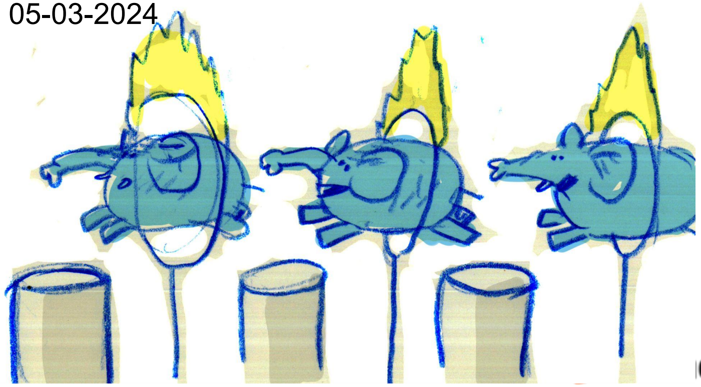
Questions?

zalando

Break

Hazards of logical decoding

Polina Bungina, PostgreSQL Berlin Meetup, 05-03-2024



Agenda

1. Intro
    a. CDC, approaches for PostgreSQL
2. Logical decoding
    a. The hazards
    b. Specific problems

CDC - Change Data Capture

- Is a set of design patterns used to track when and what changes occur in data then notify other systems and services that must respond to those changes.

- Maintain consistency across all systems that rely on data

CDC in PostgreSQL

- **Queries on Timestamp Column**
- **Triggers**
- **Logical decoding**

Logical decoding

Process of extracting all persistent changes to a database’s tables into a coherent, easy to understand format which can be interpreted without detailed knowledge of the database’s internal state.

Decoding the contents of the WAL (changes on the storage level) into the app-specific form (e.g. tuples, SQL statements)

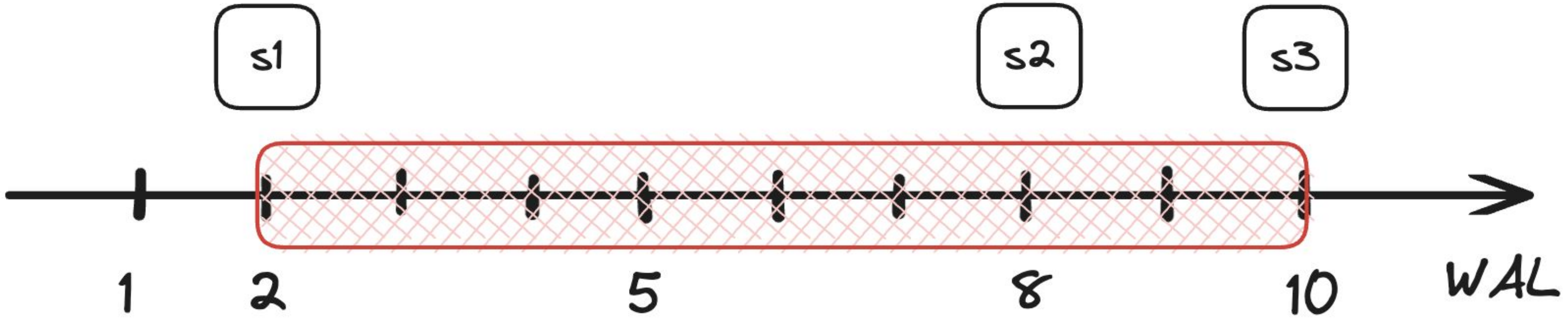
The hazards of logical decoding

Replication slots!

- All resources required for the consumer are defined by the slot’s position in the form of an LSN (log sequence number)
- Resource is not removed if needed by at least one consumer
    - WAL
    - System catalog rows
- Crash-safe - state is persisted to disk
    - only on checkpoint
    - independently of the connection using it

- **confirmed_flush_lsn** - LSN up to which the logical slot's consumer has confirmed receiving data
- **restart_lsn** - LSN of the oldest WAL which still required by the consumer => won't be removed during checkpoints
- **catalog_xmin** - the oldest transaction affecting the system catalogs that this slot needs the database to retain => VACUUM cannot remove catalog tuples deleted by any later transaction

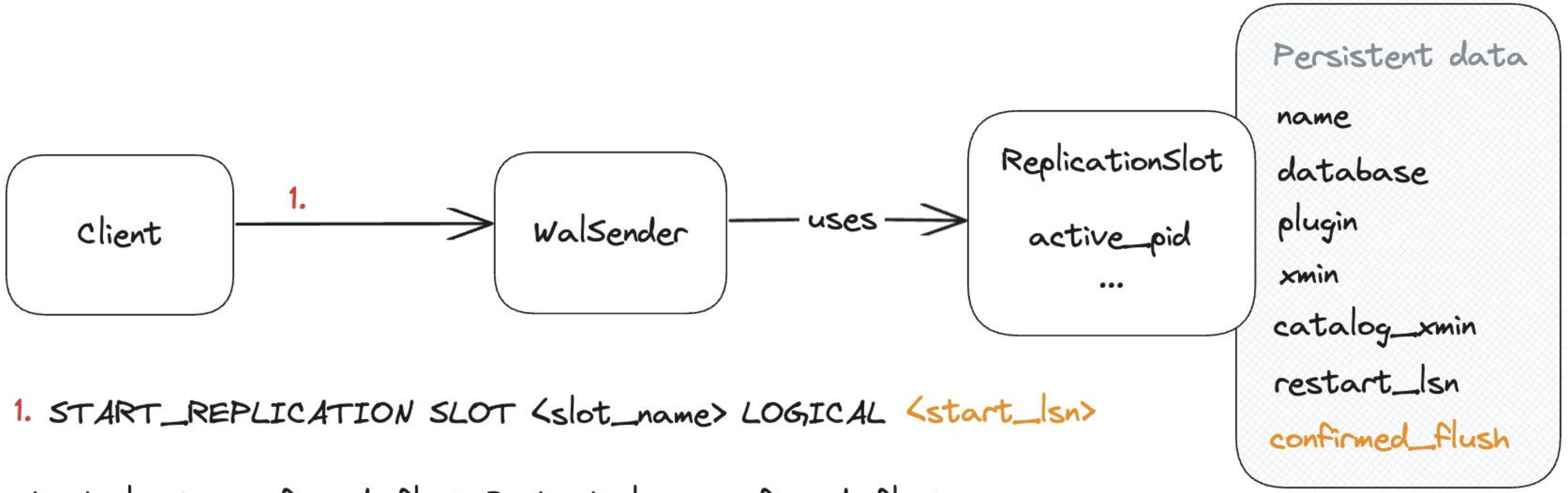
- exposed in pg_replication_slots view



slot1: restart_lsn = 2
slot2: restart_lsn = 8
slot3: restart_lsn = 10

lsn >= 2 not removed

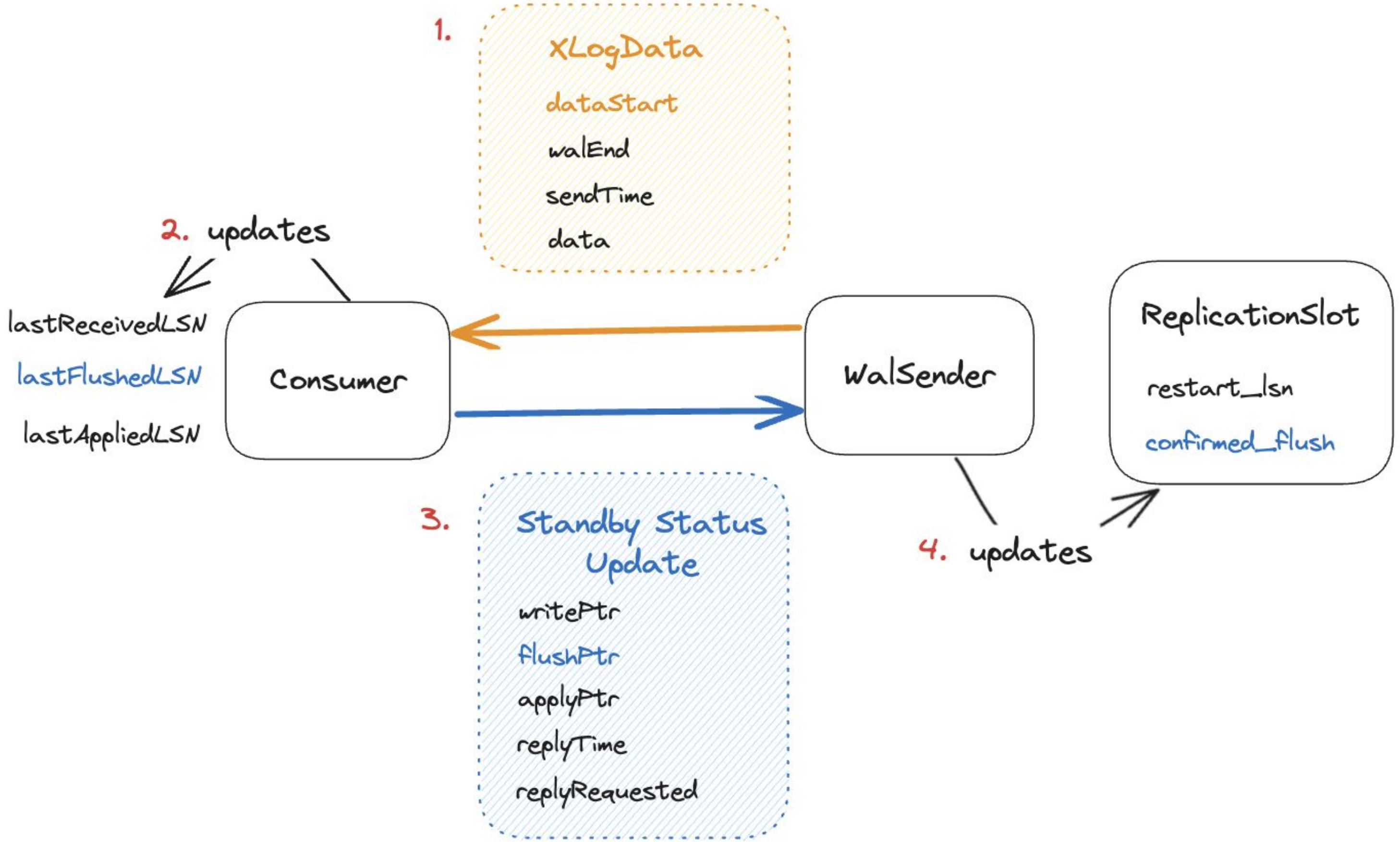
The hazards - replication slots



1. START_REPLICATION SLOT <slot_name> LOGICAL <start_lsn>

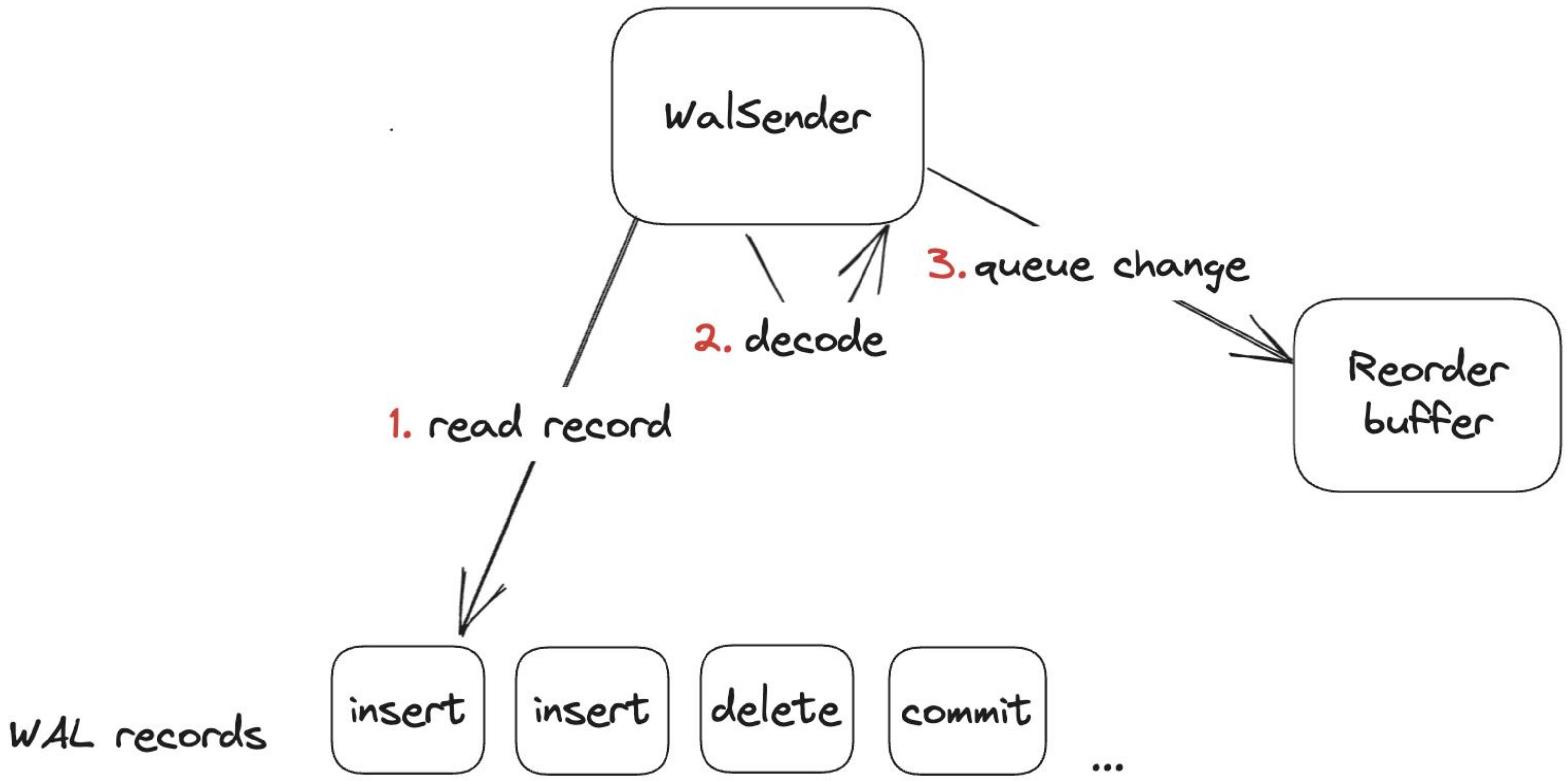
start_lsn >= confirmed_flush ? start_lsn : confirmed_flush

The hazards - replication slots

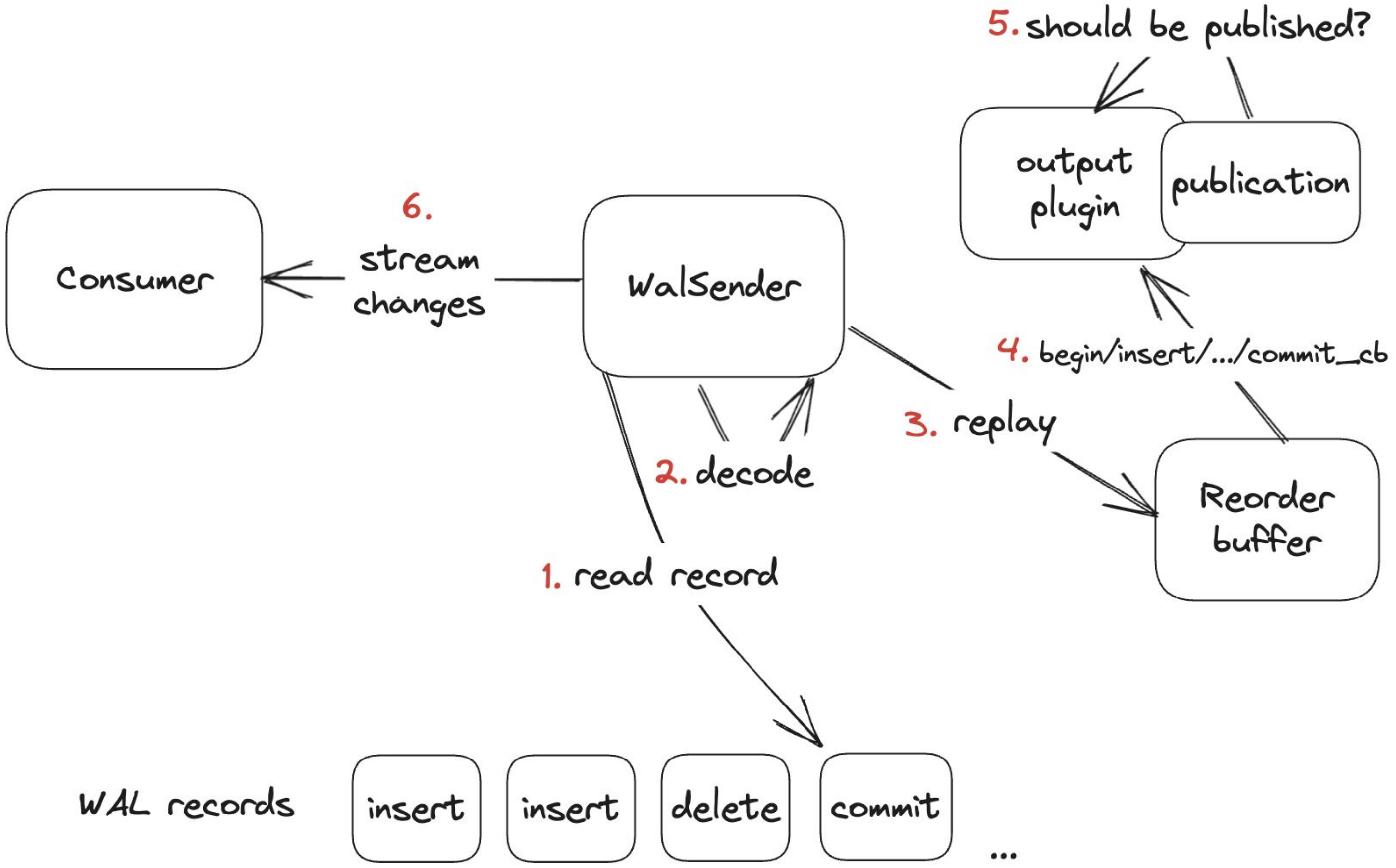


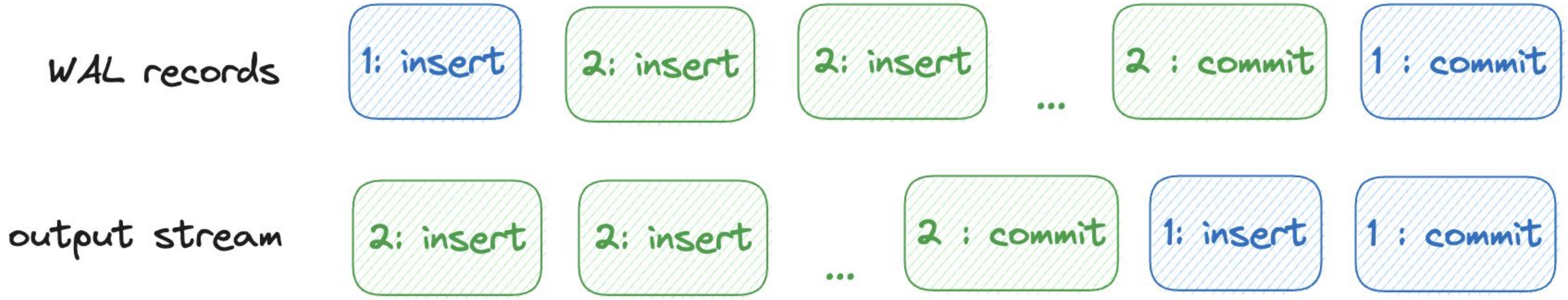
Important things to consider

- Single WAL for the whole cluster, not per-database
- Repeated message handling (after a crash/failover)
- Decoded transactions are sent on commit
    - [PG14] streaming of in-progress transactions


WalSender
1. read record
2. decode
3. queue change
Reorder buffer
WAL records
insert
insert
delete
commit
...

Logical decoding

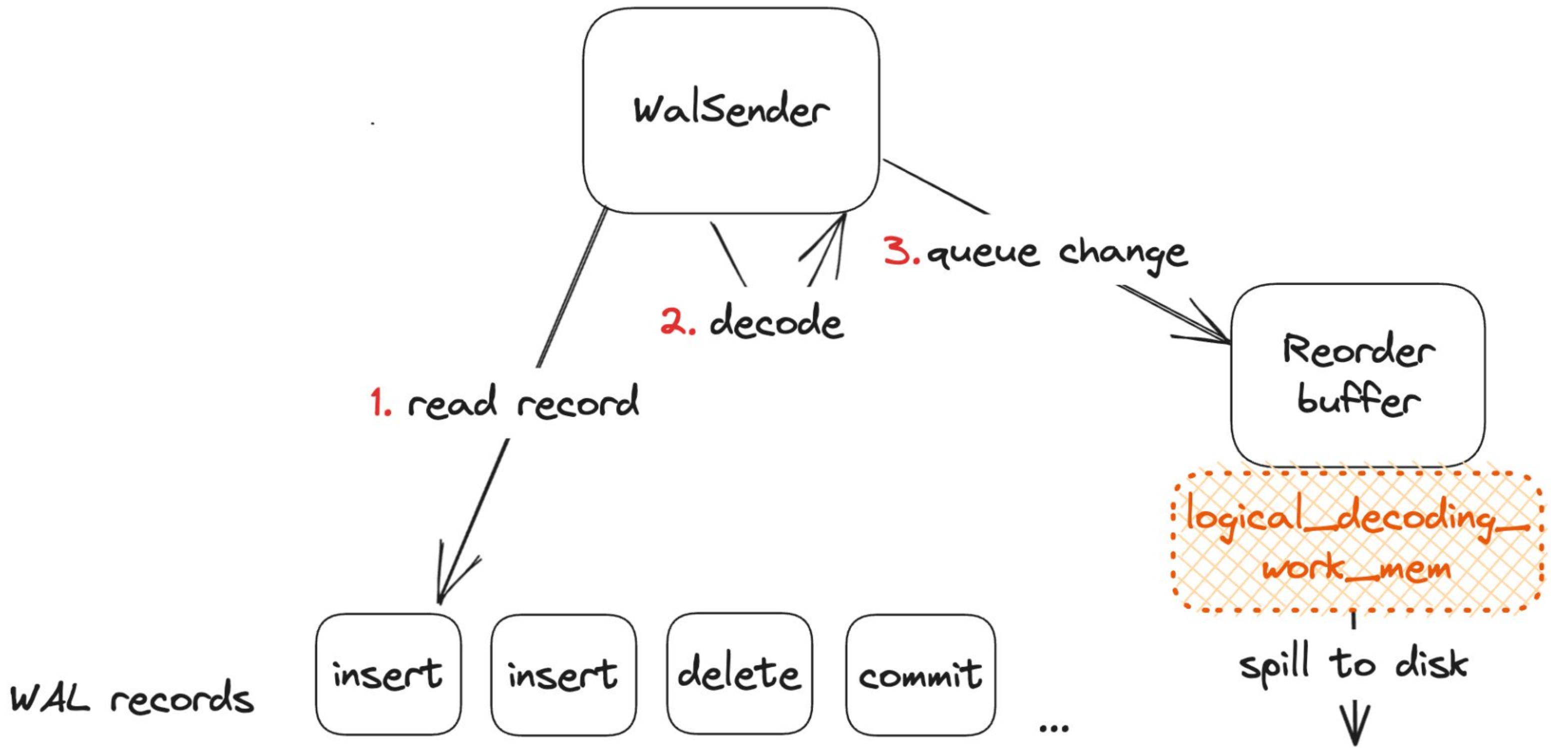



WAL records
1: insert
2: insert
2: insert
...
2 : commit
1 : commit
output stream
2: insert
2: insert
...
2 : commit
1: insert
1 : commit

“Big” transactions

- *logical_decoding_work_mem* - max amount of memory to be used by logical decoding, before some of the decoded changes are written to local disk
- Written to: *$PGDATA/pg_replslot/<slot_name>/**
- [PG14] pg_stat_replication_slots view
    - *spill _txns, spill_count, spill_bytes*

The hazards - “big” transactions



Specific problems

Slot is still there

- Consumer downscaling
  - test the new approach: we go live later and don't expect new events
  - misconfigured setup: "decoding doesn't work as expected… Oh, but it is already 6pm, will handle it tomorrow."
  - test env on the weekends

But wait, I

- do not produce writes for my publication
  - see “WAL is a single stream” for the whole cluster
- do not produce write activity in the db
  - scheduled vacuum/reindex/… ?
  - autovacuum
  - WAL segment switches (archive_timeout) on any database activity

    Segment file size is always the same (16MB by default)

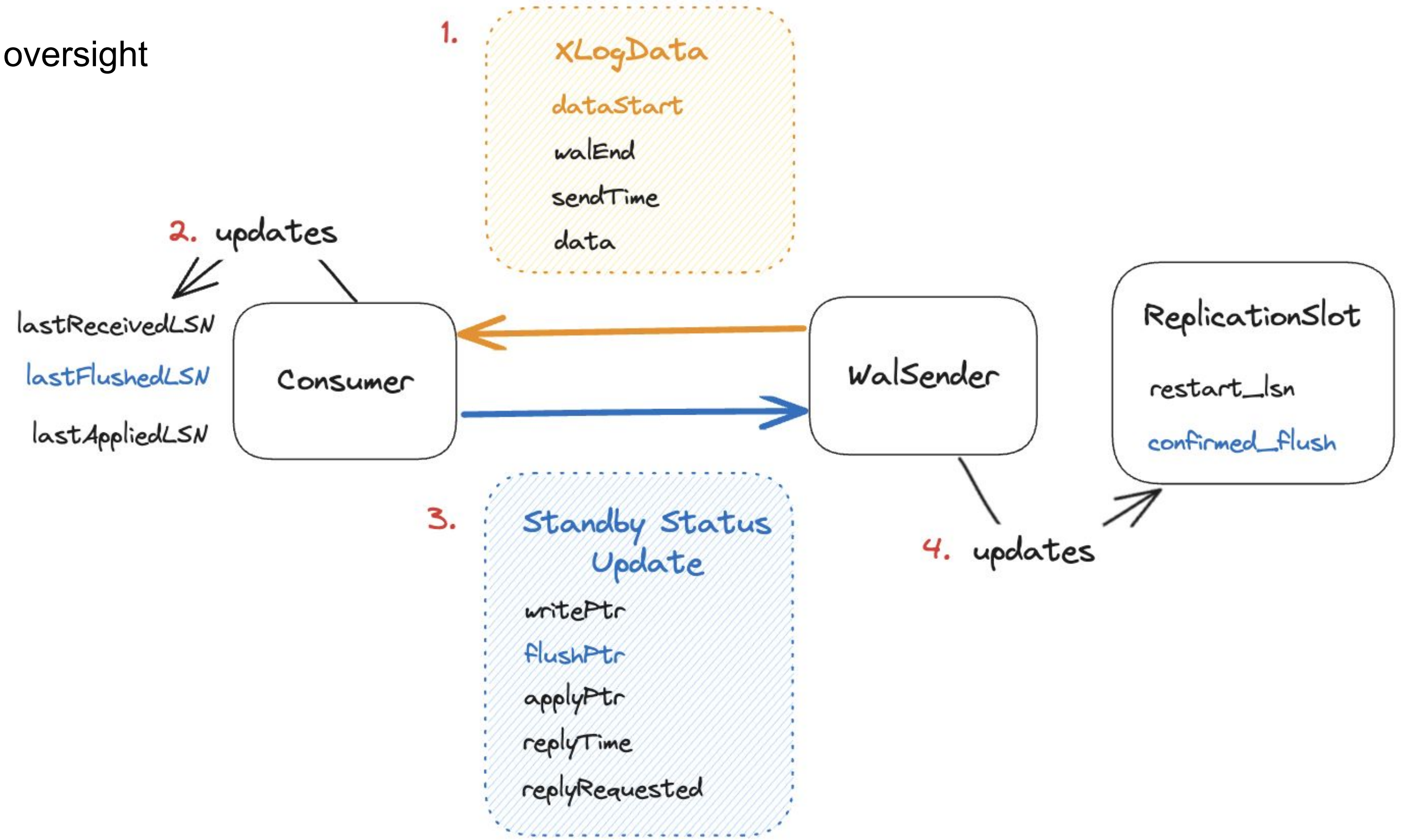
Slot is still there

- Cleanup
  - not just stopping to consume - slot should be deleted once not needed
  - Patroni config: delete from the permanent slots section

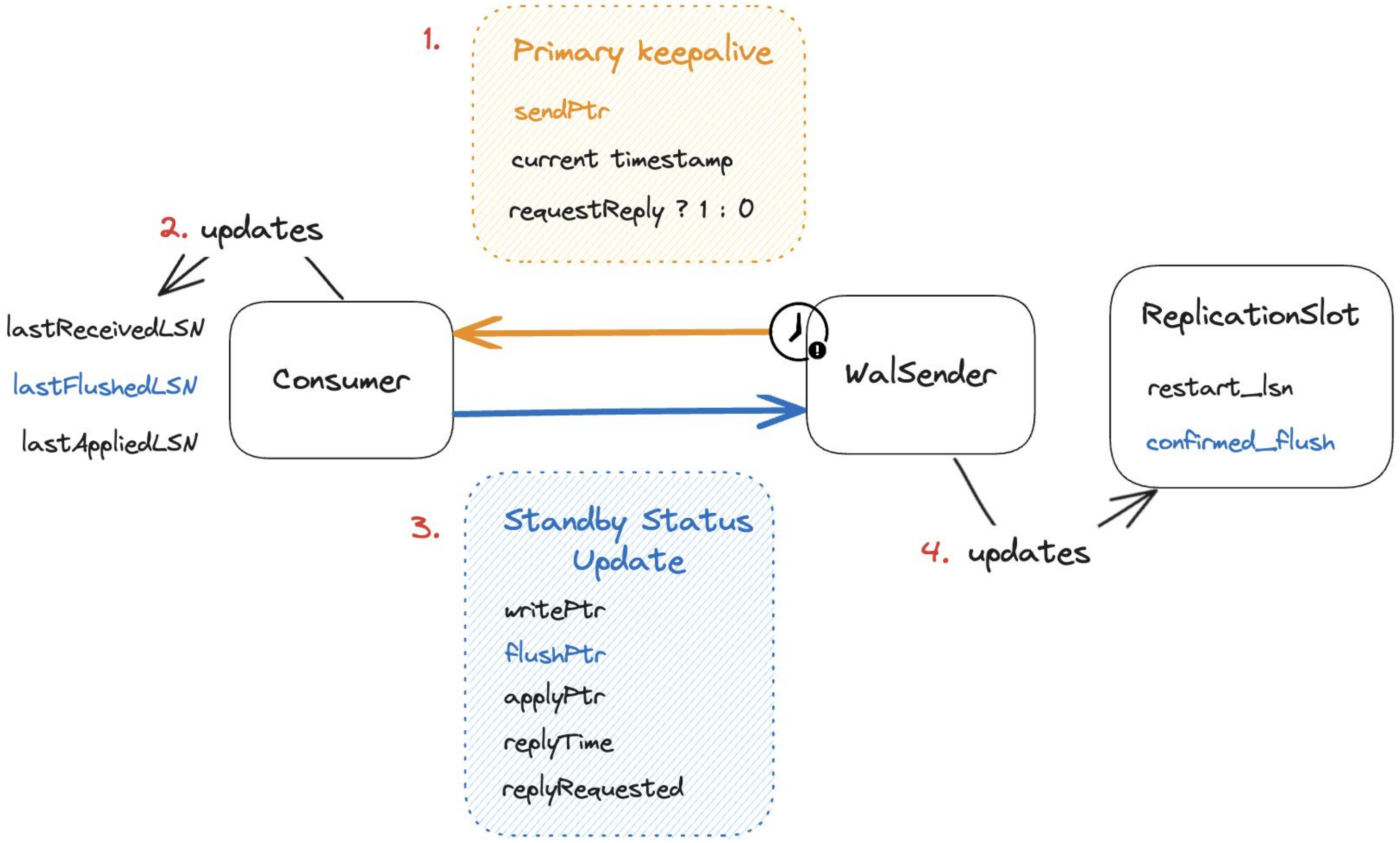
Consumer performance

- Failing/not performant consumer == risk of WAL files piling up
- Make use of publications (filtering):
  - *FOR TABLE / FOR TABLES IN SCHEMA*
  - *with (publish = 'insert, update, delete, truncate')*
  - unrelated events are not sent to the consumer

Pgjdbc oversight



Specific problems - pgjdbc oversight



Specific problems - pgjdbc oversight

- Used by Debezium
- Not properly handling Keepalive messages
  - Keepalive msg with the flag requestReply
  - Local flushPtr is not updated with the received sendPtr position
  - Standby Status Update msg with the old flushPtr
  - Slot is not advanced unless there is a write produced for the respective publication
  - Problem of endless shutdown
    - a clean shutdown == all changes are received by consumers
- Fixed in pgjdbc v42.7.0

“Big” transaction

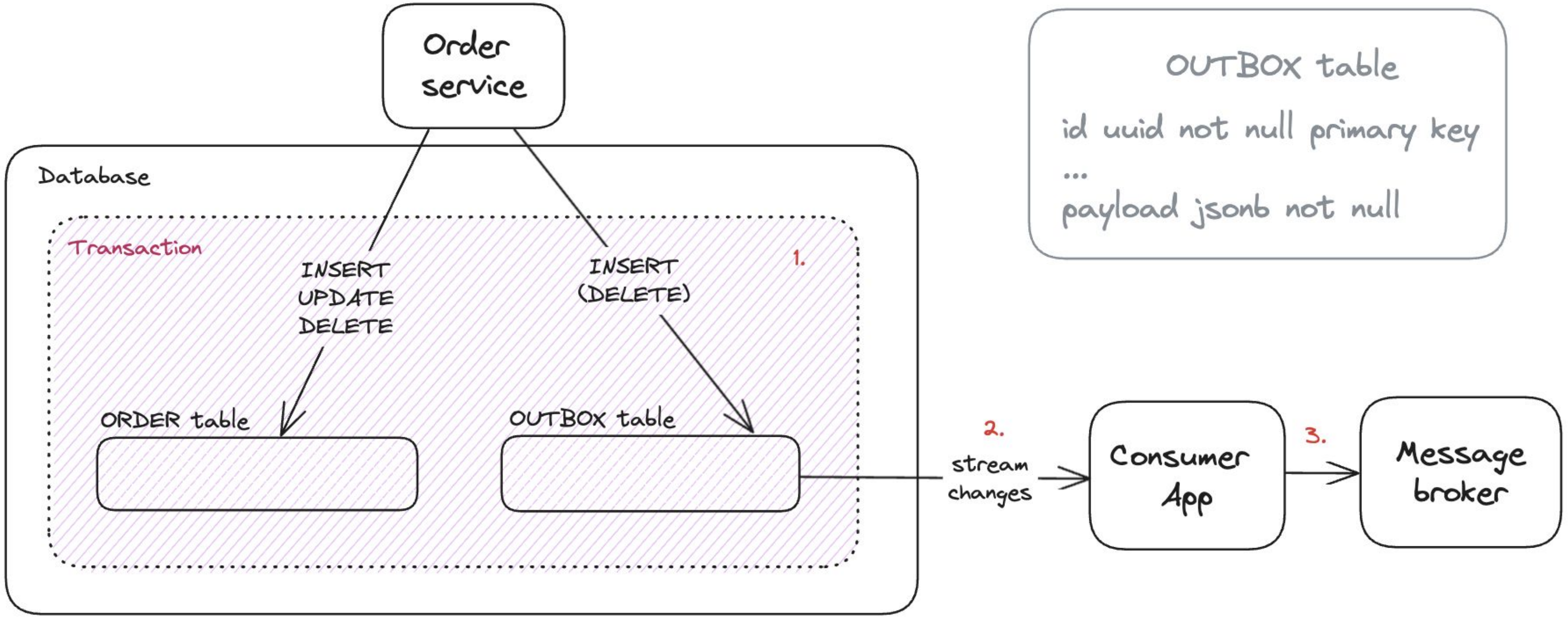
“Transactional outbox” pattern:

- Atomically update the database and send messages to a message broker
- Messages emitted via an “outbox table” in the same transaction
- Improved way (without additional table) -

  pg_logical_emit_message(transactional boolean, prefix text, content text) → pg_lsn

  pg_logical_emit_message(transactional boolean, prefix text, content bytea) → pg_lsn

Specific problems - “Big” transaction



CREATE PUBLICATION name FOR TABLE outbox WITH (publish = 'insert');

Initial situation:

- Outbox table grew to 300GB (no immediate deletion)
- Deletion cron job running for several days, as we are already too big
- Table and indexes became bloated
- Table is actively used for reads - latency spikes observed

Let’s run pg_repack!

- New table containing all the rows of the original one
  - *INSERT INTO <temp_table> SELECT … FROM <origin_table>*
- For 300GB is a long, write-intensive transaction
- full_page_writes = ‘on’
- Huge amount of WAL generated

15:45
```
~/pgdata/pgroot/data$ du -sh pg_replslot/*
70GB			pg_replslot/my_slot
```

15:53
```
~/pgdata/pgroot/data$ du -sh pg_replslot/*
152GB			pg_replslot/my_slot
```

16:03
```
~/pgdata/pgroot/data$ du -sh pg_replslot/*
185GB			pg_replslot/my_slot
```

16:09 - slot dropped
```
~/pgdata/pgroot/data$ du -sh pg_replslot/*
8.0K		pg_replslot/my_slot
```

Thank you!

🍕Pizza & Drinks 🍻

Thank You!

See you again soon …

zalando