Logical decoding

The door to a new world of data exchange and integration applications for PostgreSQL

8K data

Álvaro Hernández <aht@8kdata.com>

About 8K data

- Research & Development in Databases
- Consulting, Training and Support in PostgreSQL
- Founders of PostgreSQL España, 4th largest PUG in the world (~500 members)
- Myself: Álvaro Hernández, 8Kdata CTO
  Twitter: @ahachete
  LinkedIn: http://linkd.in/1Jm4tAx

Logical Decoding in 1 slide

- Extract changes (INSERT, UPDATE, DELETE) from PostgreSQL in a database-independent way
- Changes are idempotent and ordered.
  Changes can be streamed from PostgreSQL
- Reply the state of the database externally:
  - ✔ Replication solutions
  - ✔ Materialized databases
  - ✔ Third-party data-processing applications

Logical Decoding:
How it works

Logical Decoding: the basics

"Logical decoding is the 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"

- Changes are decoded row by row, even if they were produced by a single command
- Unlogged and temp tables are not decoded

Logical Decoding: the basics

- As of today, no DDL is decoded
  (empty tx may appear in the stream)
- Requires superuser or replication permissions
- Logical decoding works on logical replication slots (based on physical slots): fine control the amount of WAL to be kept at the server

Logical Decoding Plugins

- The output (database-independent representation of the change) format is controlled by an output plugin
- Loaded dynamically (shared library)
- Text or binary output
- A default one is in contrib ('test_decoding')

Logical Decoding interfaces

SQL

- Poll for changes
- SQL interface (function calls)
- Primarily meant for testing/debugging

Streaming Replication

- Changes are pushed by PostgreSQL
- Exports the snapshot while connection open
- Allows for synchronous (logical) replication

Configure PostgreSQL for LD

- wal_level = logical
- max_replication_slots = <x>
- If accessed over the replication interface:
  - ➔ max_wal_senders = <y>
  - ➔ Configure pg_hba.conf to allow replication
- synchronous_commit = on
  (decoding starts as soon as data is flushed)

SQL interface

```
SELECT * FROM
pg_create_logical_replication_slot
('<slot_name>', 'test_decoding');

– do changes in the db

SELECT * FROM pg_logical_slot_get_changes
('<slot_name>', null, null);
```

SQL interface

- Obtain the changes:
  - get vs peek (consume / not consume changes)
  - {get,peek}_binary: output is bytea

- Output plugin options: control output format
  test_decoding options:
  - …, 'include-timestamp', 'on', …
  - …, 'include-xids', '1', …

- Other arguments: upto_lsn, n_entries

Slots lifecycle

- Drop the slot when finished using it:
  select pg_drop_replication_slot('<slot_name>');
- If slot is not consumed, all WAL since slot creation are retained!
- If logical decoding client crashes, your database may end up stopping if pg_xlog fills!
- Slots may be consumed by more than one client

Checking status

- pg_replication_slots
  View with information about both physical and logical replication slots

- pg_stat_replication
  View replication statistics (logical decoding only if connected via the replication interface)

REPLICA IDENTITY

- On UPDATE or DELETE the old row information is decoded depending on REPLICA IDENTITY:
  - DEFAULT: values from PK (if any)
  - FULL: all values
  - NOTHING: none
  - USING INDEX <index_name>: values covered by the index (not null, not partial)
- ALTER TABLE … REPLICA IDENTITY …

Replication interface

- Send commands over replication protocol

- Or test with psql:

psql "dbname=postgres replication=database" -c "CREATE_REPLICATION_SLOT slotname LOGICAL test_decoding <options>"

- Or use pg_recvlogical!

pg_recvlogical --slot slotname --create -d db
pg_recvlogical --slot slotname --start -f - -d db

Logical Decoding: the door to a new world of data exchange and integration applications for PostgreSQL

8K data

Data exchange, data integration

- Logical Decoding is primarily used for replication (UDR, BDR, Slony, others?)

- But it is much more than that. Much more!
  - ➔ Extract data from PostgreSQL, reproduce in other systems
  - ➔ Create externally-controlled data applications
  - ➔ Integrate with other systems (even PostgreSQL!), like in a sharding environment

Output plugins

- test_decoding
  Text output, not easily parseable, but works.
  Included in contrib

- https://github.com/michaelpq/pg_plugins/tree/master/decoder_raw
  Decodes to SQL

- https://github.com/xstevens/decoderbufs
  Decodes to protocol buffers

Bottled Water

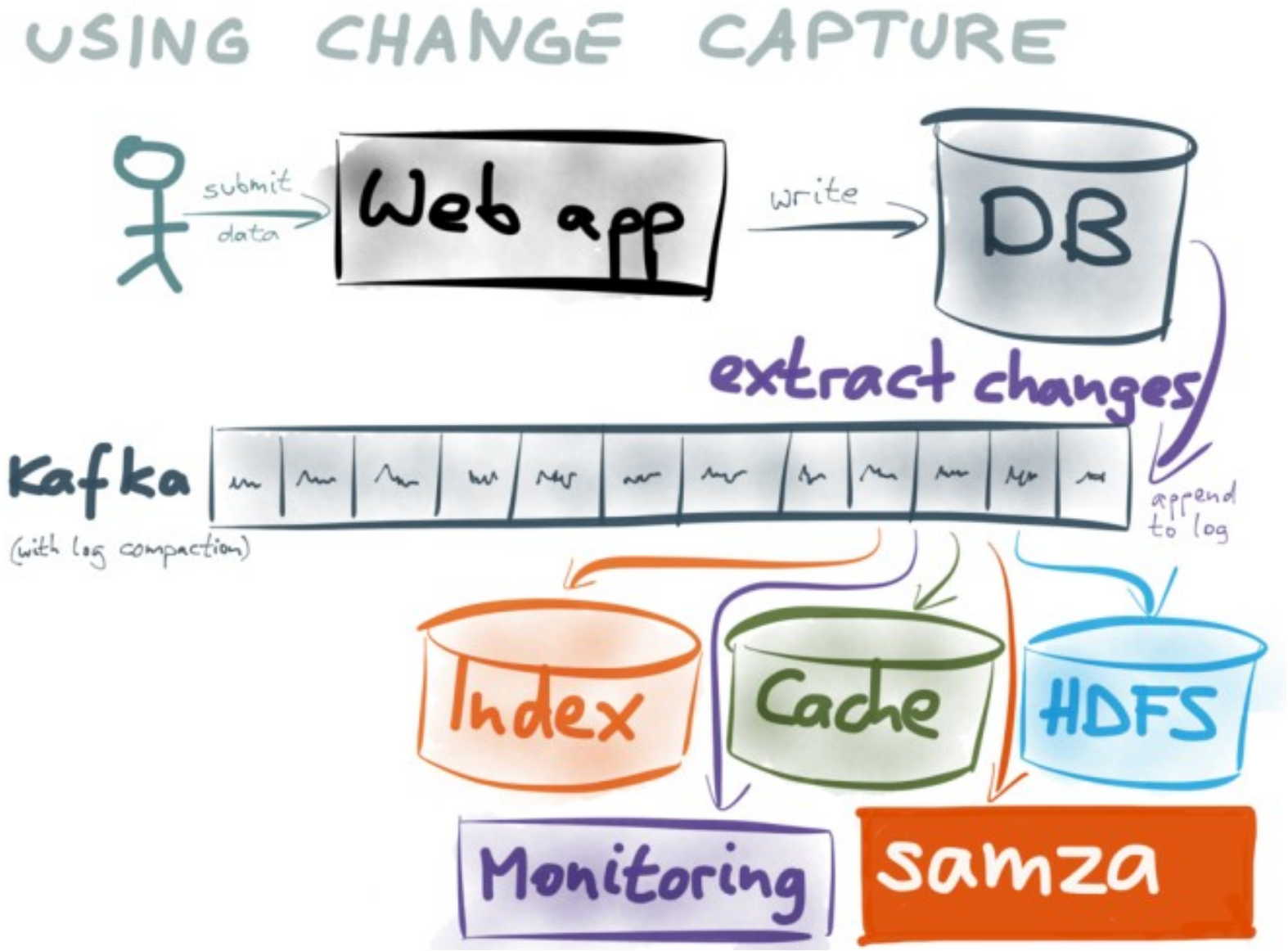
- https://github.com/eulerto/wal2json
  Decodes to JSON
- https://github.com/confluentinc/bottledwater-pg
  Decodes to avro, injects into Kafka!
- Your plugin?

Event sourcing

- Logical Decoding is event sourcing for Postgres

- Extract your changes, process them and:
  - Perform real-time processing (in-memory dbs)
  - Materialize databases
  - Invalidate caches
  - Audit systems
  - Replicate (of course)
  - Distribute changes via WAN for DR

Example architecture



Source: http://www.confluent.io/blog/bottled-water-real-time-integration-of-postgresql-and-kafka/

External Logical Decoding 101

- Implement the replication protocol
- Open replication connection. Get snapshot
- Open new SQL connection. SET TRANSACTION SNAPSHOT, repeatable read tx. Dump all data
- COPY BOTH (replication connection). Receive changes
- Send feedback to the server!

More information

- PostgreSQL docs
- http://www.pgcon.org/2014/schedule/attachments/326_logical-decoding-pgcon-2014-05-23.pdf
- http://thebuild.com/presentations/fosdem-2015-logical-decoding.pdf
- http://michael.otacoo.com/content/materials/20140919_pgopen_logirep.pdf


8K
data