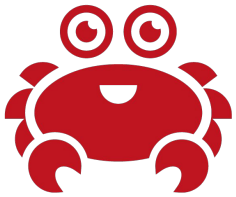

aiven

Change Data Capture for a brave new world

https://aiven.io

Me

Hannu Valtonen, Co-founder of Aiven

Maintainer of PGHoard, pglookout and pgmemcache

PostgreSQL user for the last 18 years

CDC Definition

In databases Change Data Capture (CDC) is a set of software design patterns used to determine (and track) the data that has changed so that action can be taken using the changed data.

- Wikipedia, https://en.wikipedia.org/wiki/Change_data_capture

CDC - the Why

- Data's journey through your company's systems usually just starts with its initial storing
- Real-time change information as it happens
- No need to do bulk updates anymore with all their assorted errors
- Much more efficient, way fewer resources required since only delta is transferred

Foreword on examples

```
CREATE TABLE source_table (
    id SERIAL PRIMARY KEY,
    important_data text NOT NULL,
    create_time TIMESTAMPTZ NOT NULL DEFAULT clock_timestamp(),
    update_time TIMESTAMPTZ NOT NULL DEFAULT clock_timestamp(),
    updated BOOLEAN NOT NULL DEFAULT FALSE
);

INSERT INTO source_table (important_data)
    VALUES ('first bit of very important analytics data');

INSERT INTO source_table (important_data)
    VALUES ('second bit of very important analytics data');
```

CDC - in the age of the Dinosaur

<Imagine a picture of dinosaurs roaming freely through an idyllic landscape>

We're now talking about prehistoric times that predate the early 2000's

CDC - in the age of the Dinosaur

- Nightly database dump of some or all tables often done with pg_dump
- ETL from multiple databases to a single system
- Batch based
- Maybe using some proprietary ETL
- PostgreSQL COPY command made this less onerous

CDC - in the age of the Dinosaur

- Timestamp / sequence / status column based approach
- Add a column `updated_timestamp` to your table which you then read afterwards to try to find changed rows
- Same thing by having an updated boolean column in your table
- Possible limitations for noticing deletes or updates in naive implementations

CDC - in the age of the Dinosaur

```
SELECT * FROM source_table
    WHERE id >= 0
    ORDER BY id ASC LIMIT 1;

SELECT * FROM source_table
WHERE timestamp >= y
    ORDER BY timestamp ASC LIMIT 1;

SELECT * FROM source_table
    WHERE updated IS FALSE
    ORDER BY id ASC LIMIT 1;
```

CDC - in the age of the Dinosaur

```
SELECT * FROM source_table
    WHERE updated IS FALSE
    ORDER BY id LIMIT 1;
UPDATE source_table SET updated = 't'

WHERE id = (SELECT id FROM source_table
    WHERE updated IS FALSE
    ORDER BY id ASC LIMIT 1)
    RETURNING *;
```

CDC - Trigger based approaches

You create change tables that contain the INSERTed UPDATEd or DELETEd rows

- Slony, PGQ (Londiste) and plenty of homespun solutions
- Allows all DML (INSERTs, UPDATEs, DELETEs) to be extracted
- Bad performance as in doubles all writes done to the database
- Doesn't handle DDL (ALTER TABLE etc) gracefully

CDC - Trigger based approaches

- `CREATE TRIGGER store_changes AFTER UPDATE, INSERT, DELETE ON source_table FOR EACH ROW EXECUTE PROCEDURE store_change();`
- And then the trigger just INSERTs the contents of the change to a change table with the information stating whether it was an `INSERT`, `UPDATE` or `DELETE`
- The change table contents are read and applied from start to finish in some other database

CDC - Advent of a new age

<Imagine a picture of something very modern here>

PostgreSQL's built-in logical decoding saga started with the release of PostgreSQL 9.4 at the end of 2014

CDC - Logical decoding - What is it?

- PostgreSQL can keep track of all the changes happening in a database
- Decodes WAL to desired output format
- Multiple logical decoding output plugins exist
- Very performant, low-overhead solution for CDC
- Avoids the multiple write problem with triggers by using the WAL that PG was going to write anyway

CDC - Logical decoding - What can it do?

- Track all DML (INSERT, UPDATE, DELETE) changes
- Unit of Change is a row of data that's already committed
- Allows reading only the wanted subset of changes
- Use cases include auditing, CDC, replication and many more
- Logical replication connections supported in multiple PostgreSQL drivers (JDBC, Python psycopg2)

CDC - Logical decoding - What can't it do?

- Replicate DDL
  - Possible to set up event triggers that write to a table and then have your replication system run the DDL based on it
- Depending on output plugin some data types not supported
- Failovers not handled gracefully as replication slots exist only on master nodes
- Changes tracked are limited to a single logical DB

CDC - Logical decoding - How to set it up?

postgresql.conf:

```
wal_level=logical
max_replication_slots = 10 # At least one
max_wal_sender = 10  # At least one

$ CREATE ROLE foo REPLICATION LOGIN;
```

Before PG 10 also needs changes to pg_hba.conf

CDC - wal2json

- Author: Euler Taveira de Oliveira
- Decodes logical changes into JSON format
- Datatype limitations (JSON doesn't natively support everything)
- Supported by multiple DBaaS vendors (Aiven, AWS RDS, https://github.com/eulerto/wal2json)

CDC - approach - pg_recvlogical

- Receive all logical changes and write them to a file
- Advantage of this is simplicity, great for simple use cases
- Downside is that there's just a single receiver of the data that writes it into a simple file
- If you lose that single file you've lost the change

CDC - wal2json usage

```
$ /usr/pgsql-10/bin/pg_recvlogical -d "host=localhost user=valtha"
--create-slot --start --slot pgdayparis --plugin wal2json -v
--file -

{"change":[{
    "kind":"insert",
    "schema":"public",
    "table":"source_table",

    "columnnames":[
        "id","important_data","create_time","update_time","updated"],
    "columntypes":[
        "integer","text","timestamp with time zone","timestamp with time
zone","boolean"],
    "columnvalues":[
        1,"first bit of very important analytics data","2018-03-15
02:35:22.476475+02","2018-03-15 02:35:22.476476+02",false]
}]}
```

CDC - approach - read changes in an application

- Receive all logical changes and directly receive them in an application
- Really easy to transform the data further
- Acting on the data, e.g. sending an email, is trivial
- Downside is that there's just a single receiver of the data

CDC - wal2json Python replication example

```
slot_name = "pgdayparis"
log_conn = psycopg2.connect(dsn,
    connection_factory=LogicalReplicationConnection)
log_cursor = log_conn.cursor()
log_cursor.create_replication_slot(slot_name,
    slot_type=REPLICATION_LOGICAL, output_plugin="wal2json")
log_cursor.start_replication(slot_name=slot_name,
    slot_type=REPLICATION_LOGICAL)


# Insert data in autocommit mode and wait for the replication message
cursor.execute("INSERT INTO source_table (important_data) VALUES ('first
bit of very important analytics data')")
time.sleep(1)
print("Payload: {}".format(log_cursor.read_message().payload))
```

CDC - approach - streaming platform

- Receive all logical changes and write them to a distributed system
- Change data is immediately replicated *n* times
- Allows multiple readers for the same data and easy-post processing of it
- Upside is that it makes very complex processing possible
- Downside is the (much) increased complexity
- Many large organizations love this

Apache Kafka foreword

- Distributed streaming platform
- Supports publish/subscribe behavior
- Data spread into topics which are further split into partitions
- Data consists of a binary value, timestamp and an optional key
- AWS Kinesis and Google Pub/Sub are close cousins (https://kafka.apache.org/)

Apache Kafka foreword continued

- All data organized into topics and further on into partitions
- Data within a partition is guaranteed to keep order
- Messages can optionally have a key used for choosing which partition and in order to only keep the last value with that key within a topic
- If data is unkeyed it has a retention policy attached to it that prunes it based on time or data size
- Allows writing once and consuming the same data by same or other readers multiple times

Debezium

- Apache Kafka Connect Connector plugin (http://debezium.io/)
- Uses logical replication to replicate a stream of changes to a Kafka topic
- Compacted topics, so only last value kept for a key
- Supports PostgreSQL, MongoDB, MySQL, (Oracle)
- Can run arbitrary transformation code on the data as it's received
- Supports protobuf output plugin or wal2json

CDC - approach - built-in logical replication

- In CDC context usually works best if you just have have the same table in multiple databases
- Doesn’t allow any transforms of the data while in transit
- Commonly used for replicating data to larger data warehouses where analytics is run
- Useful for a lot of other things as well
- Finally built-in!

Logical replication

- With logical replication you can replicate all or a subset of your data
- Replication scope always limited to a single database for a replication connection
- Currently requires superuser privileges or a hassle
- Amazon Database Migration Service (DMS) another common use case (doesn't use built-in PG logical replication)
- Allows no-downtime migrations (in theory)

Logical replication example

```
-- Track only inserts
$ CREATE PUBLICATION cdc_pub FOR TABLE source_table WITH
(publish='INSERT');
CREATE PUBLICATION

$ CREATE SUBSCRIPTION test_sub CONNECTION
'host=127.0.0.1 port=5433 dbname=source user=postgres'
PUBLICATION cdc_pub;
CREATE SUBSCRIPTION

NOTICE:  created replication slot "test_sub" on
publisher
```

Logical replication example

Source database:

```
 source=# INSERT INTO source_table
(important_data) VALUES ('first bit of
important replicated analytics data');
```

Target database:

```
 target=# SELECT * FROM source_table;
```

Demo

<If we have some time left>

aiven

Logical decoding - Recap

- Logical decoding and replication have revolutionized the way CDC can be done with PostgreSQL
- We’re only seeing the very beginnings of its adoption
- Note that logical decoding is not a perfect solution (yet)

Q&A

- Time to ask me anything

aiven

The end

- Thank you for attending

aiven