Catch me if you can - PostgreSQL and Debezium

Bernd Helmle

PGConf.DE 2024

Who's that guy talking?

- Bernd Helmle
- Senior Database Engineer
- Works since two decades on and with PostgreSQL

Who is Cybertec?

Motivation

- Systems need to process data in nearly realtime.
- Historically batch processing infrastructure was set
- Batch processing doesn’t scale well
- Streaming infrastructure to the rescue

What do we want

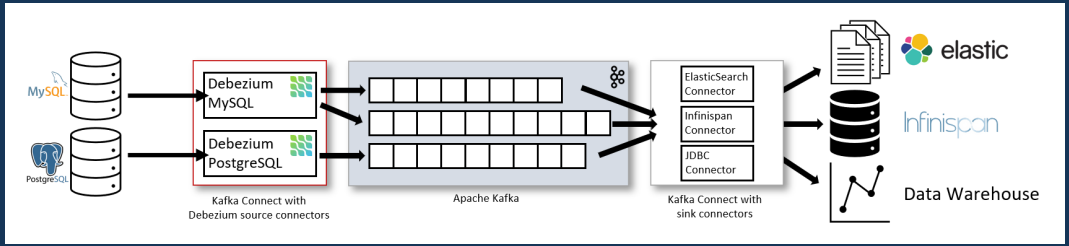
- Capture changes on our databases for further processing
- Replace existing batch infrastructure
- Create infrastructure to support heterogeneous environments
- Low impact
- Easy setup
- Monitoring
- Fun

Debezium

- Builds on top of the Kafka Connect framework
- Connectors for all mature databases (Oracle, DB2, Postgres, . . .)
- Debezium CDC via Kafka Connect, Debezium Server or Embedded API
- Sink Connector for JDBC targets
- Integrates great into streaming infrastructure architectures
- REST API

Debezium CDC via Kafka

- Kafka Broker
- Kafka Connect



PostgreSQL and Debezium

- Leverages Logical Decoding API in PostgreSQL
- Debezium source connector for PostgreSQL
- Two available decoding plugins:
  - `decoderbufs` (maintained and developed by the DBZ team)
  - `pgoutput` (also used by builtin logical replication, since PG 10)

Requirements

- `wal_level=logical`
- `max_replication_slots=<number>`
- Disk space (more on this later)
- `decoderbufs` needs additional installation
  - included in the PGDG APT and YUM repositories:
    - `postgres-decoder_bufs_16`
  - `shared_preload_libraries='decoderbufs`

Installation (1)

Best installation source are containers, e.g.

Debezium Docker Repository

- debezium/connect: Kafka Connect with all Debezium plugins
- debezium/kafka: Kafka Broker
- debezium/tooling: Add-On Tools
- debezium/debezium-ui: Web-GUI for Debezium Connector Administration

Installation (2)

On k8s a good start is Strimzi, see details on

Strimzi

and

Debezium on Kubernetes

Source Connector

The PostgreSQL source connector (and all others including the newer JDBC sink) is included in the debezium/connect image

PostgreSQL connector configuration (1)

```
CREATE TABLE mails
(
        id bigserial PRIMARY KEY,
        label_id  bigint REFERENCES labels(id),
        mail_from text NOT NULL,
        mail_to   text NOT NULL,
        msg       text NOT NULL,
        subject   text NOT NULL,
        msg_id    text NOT NULL,
        date      timestamp NOT NULL
);
```


PostgreSQL connector configuration (2)

```
{
  "name": "mails",
  "config": {
      "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
      "database.hostname": "rl9-test.chufu.vm",
      "database.port": "5432",
      "database.user": "postgres",
      "database.password": "",
      "database.dbname" : "mailstore",
      "topic.prefix": "pg16-rl9",
      "table.include.list": "public.mails",
      "plugin.name": "pgoutput"
  }
}
```

```
curl -X POST -H "Accept:application/json" \
  -H "Content-Type:application/json" \
  cdc-kafka-connect-1:8083/connectors/ -d '{...}'
```

One table, one topic

`<topic.prefix>.<schema.table>`: `pg16-rl9.public.mails`

Topic routing

- If multiple tables should go into a single topic, we can use message routing in our config
- This features Kafka Connect Transformations (SMT)
- Allows to modify messages in transit

```
{
  ...
  "transforms": "Reroute",
  "transforms.Reroute.type": "io.debezium.transforms.ByLogicalTableRouter",
  "transforms.Reroute.topic.regex": "public(.*)",
  "transforms.Reroute.topic.replacement": "all_tables"
}
```

- **NOTE**: Debezium inserts `__dbz__physicalTableIdentifier` as a special field into the event key to guarantuee unique event keys.
- Useful for partitioned tables

Message format (1)

- Basic configuration creates very verbose (and large) message format
- It includes the schema for the event key and values
- If required, consider kafka topic compression

Message format (2)

- Sometimes just plain data is required, so strip the message down

```
{
  "name": "mails",
  "config": {
    ...
    "transforms": "unwrap",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable": "false",
    "value.converter.schemas.enable": "false"
  }
}
```


Message format (3)


```
{
  "id": 615211,
  "label_id": 1,
  "mail_from": "Daniel Gustafsson <daniel@yesql.se>",
  "mail_to": "Heikki Linnakangas <hlinnaka@iki.fi>",
  "msg": "...",
  "subject": "Re: Marking options deprecated in help output",
  "msg_id": "<95693F32-77F5-4587-B47F-11D20BA8B901@yesql.se>",
  "date": 1677629531000000
}
```


Message format - AVRO (1)

- Serialized binary message format
- Very space efficient and compact
- Schema registry required (like Apicurio Registry Server https://www.apicur.io/registry/)
- Special connector configuration
- Message schema by “contract”

Message format - AVRO (2)

We can adjust our example for AVRO like the following:

```
{
  "key.converter": "io.apicurio.registry.utils.converter.AvroConverter",
  "key.converter.apicurio.registry.url": "http://apicurio-registry:8080/apis/registry/v2",
  "key.converter.apicurio.registry.auto-register": "true",
  "key.converter.apicurio.registry.find-latest": "true",
  "value.converter": "io.apicurio.registry.utils.converter.AvroConverter",
  "value.converter.apicurio.registry.url": "http://apicurio-registry:8080/apis/registry/v2",
  "value.converter.apicurio.registry.auto-register": "true",
  "value.converter.apicurio.registry.find-latest": "true",
  "schema.name.adjustment.mode": "avro"
}
```


Message Events (1)

- Generated by the `pg_logical_emit_message()` function
- Useful for outbox pattern without permanent tables
- Writes to the `<topic.prefix>.message` topic
- The 2nd argument of `pg_logical_emit_message()` can be used to identify message contents

Message Events (2)

```
SELECT pg_logical_emit_message(false, 'aggregation',
                               json_agg(m.*)::text)
FROM (SELECT COUNT(*), date_trunc('year', date), mail_from
      FROM mails
      WHERE mail_from
      ILIKE '%Tom Lane%' GROUP BY 2, 3) AS m;
```


Message Events (3)

The message payload has the content encoded as `base64`.

```
kcat -c 4 -q -b cdc-kafka-1:9092 \
  -t  pg16-f39-mailstore.message  | jq '.payload.message.content'

"eyJjb3VudCI6Njc1LCJkYXRlX3RydW5jIjoiMjAyMy0wMS0wMVQwMDowMDowMCJ9"
"eyJjb3VudCI6NDA1NywiZGF0ZV90cnVuYyI6IjIwMDktMDEtMDFUMDA6MDA6MDAifQ=="
"eyJjb3VudCI6NDAyMywiZGF0ZV90cnVuYyI6IjIwMDgtMDEtMDFUMDA6MDA6MDAifQ=="
"eyJjb3VudCI6MTcyLCJkYXRlX3RydW5jIjoiMTk5OC0wMS0wMVQwMDowMDowMCJ9"
```


High Availability Setups

Replication Slots are the main contender here

- State of the slot needs to be replicated
- On reconnect, Debezium connector needs to grab its latest `LSN`
- Debezium doesn't properly handle keepalive on replication connections
- PostgreSQL 17 will have (fingers crossed) synchronized logical replication slots

High Availability Setups - patroni (1)

- Patroni can replicate slots to standby
- Needs some careful configuration in the Debezium PostgreSQL connector

High Availability Setups - patroni (2)

Create a publication manually

```
CREATE PUBLICATION dbz_mails FOR TABLES public.mails;
```


High Availability Setups - patroni (3)

- Configure patroni standbys to replication slot: `use_slots: true`
- Configure the slot names with patroni e.g via dynamic configuration settings:

```
slots:
  dbz_mails:
    database: mailstore
    plugin: pgoutput
    type: logical
```


High Availability Setups - patroni (4)

We need to extend our example connector configuration so that our slot and publication are properly used:

```
{
  "slot.name": "dbz_mails",
  "publication.name": "dbz_mails"
}
```


Streaming from standby servers

- PostgreSQL 16 allows logical decoding from physical standby servers
- Create the logical replication slot on the physical standby
- Execute the `pg_log_standby_snapshot()` function on the **primary** !!
- Configure Debezium, startup connectors

Quirks & Gotchas - Replication Lag (1)

Unfortunately, this is still an existing issue

- Logical Decoding and its architecture can cause high replication lag
- Transactions are decoded at commit
- Long running transactions !
- Tables not in publications can generate huge backlog
- Heartbeat/Keepalive required and mandatory in such situations

Quirks & Gotchas - Replication Lag (2)

A possible solution is implemented inf PgJDBC

PgJDBC Keepalive Patches

and

github PgJDBC PR (merged in 42.7.0)

But even current Debezium `2.6.0` is not shipping fixed `pgjdbc` :()

Issue with pgjdbc 42.7.x

Quirks & Gotchas - Replication Lag (3)

- PostgreSQL connector needs to configured for heartbeats
- Heartbeat tables need to be specified in `table.include.list`

```
{
  "table.include.list": "..., dbz_heartbeat",
  "heartbeat.interval.ms": "30000",
  "heartbeat.action.query": "UPDATE dbz_heartbeat SET last_updated = NOW()"
}
```


Quirks & Gotchas - Record Sizes

```
Caused by: org.apache.kafka.common.errors.RecordTooLargeException:
The message is 2338874 bytes when serialized
which is larger than 1048576, which is the value of the
max.request.size configuration
```

- Happens with large bytea/text/json(b) columns
- Needs adjustments to Kafka Connect parameters in `connect-distributed.properties`

```
"producer.max.request.size": "15000000"
"consumer.max.request.size": "15000000"
```


Tweaks - Query Debezium info from PostgreSQL (1)

- Useful extension https://github.com/pramsey/pgsql-http
- Allows to query and performing actions via Debezium REST API

Tweaks - Query Debezium info from PostgreSQL (2)

```
SELECT jsonb_pretty(content::jsonb)
  FROM http_get('http://localhost:8083/connectors/mails/status');
              jsonb_pretty
-------------------------------------------
 {                                        +
     "name": "mails",                     +
     "type": "source",                    +
     "tasks": [                           +
         {                                +
             "id": 0,                     +
             "state": "RUNNING",          +
             "worker_id": "172.18.0.10:8083"+
         }                                +
     ],                                   +
     "connector": {                       +
         "state": "RUNNING",              +
         "worker_id": "172.18.0.10:8083"  +
     }                                    +
 }
```


Tweaks - Query Debezium info from PostgreSQL (3)

- Pause a connector

```
SELECT http_put('http://localhost:8083/connectors/mails/pause', '','');
```

- Resume the connector

```
SELECT http_put('http://localhost:8083/connectors/mails/resume', '','');
```


Tweaks - the kcctl tool

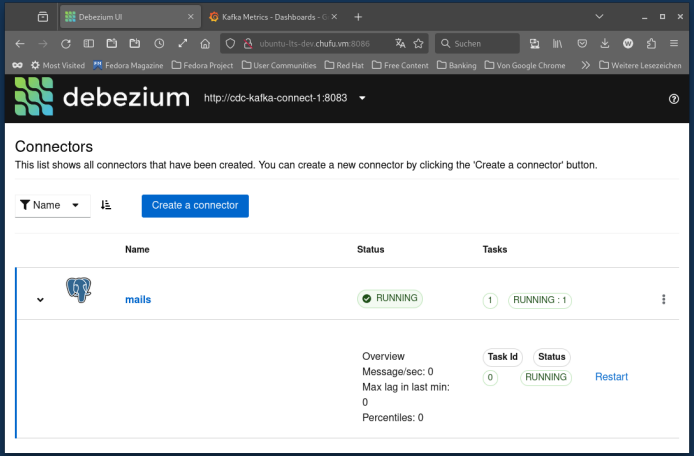
CLI Tool to configure and administer connectors: https://github.com/kcctl/kcctl

```
$ kcctl get connectors

 NAME    TYPE     STATE     TASKS
 mails   source   RUNNING   0: FAILED
 test    source   RUNNING   0: RUNNING

$ kcctl restart task mails/0
```


Tweaks - the Debezium UI



PQfinish(lecture)

Questions?