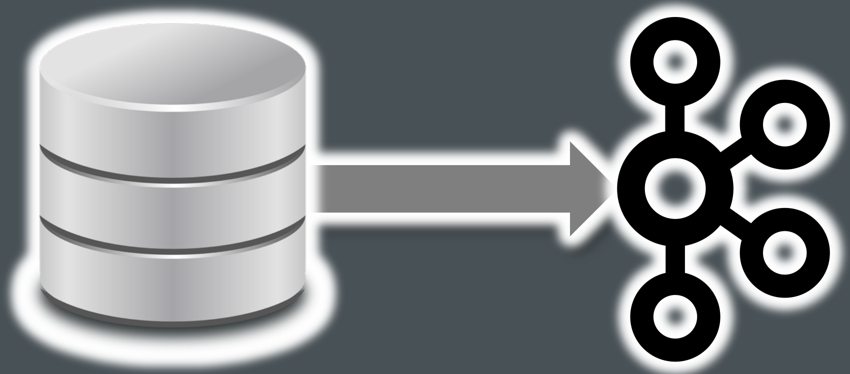
Otto Group Solution Provider

MOVEX Change Data Capture

Lightweight tool for change data capture in relational databases



Peter Ramm, Otto Group Solution Provider (OSP) GmbH

April 2022

Otto Group Solution Provider (OSP)

Founded:
March 1991

Parent company:
Otto Group

Locations:
Dresden, Hamburg, Altenkunstadt, Madrid, Taipei

Number of employees:
> 450

Managing Directors:
Dr. Stefan Borsutzky, Norbert Gödicke, Jens Gruhl

Homepage:
https://www.osp.de

About me

Peter Ramm
Team lead strategic-technical consulting at OSP Dresden

> 30 years of history in IT projects

Main focus:
- Development of OLTP systems based on Oracle databases
- Architecture consulting to trouble shooting
- Performance optimization of existing systems

Tasks to be solved by a requested solution

OSP
Otto Group Solution Provider

- Capture data change events (insert/update/delete) in relational databases and transfer these events in a timely manner in JSON format to a Kafka Event Hub.
  - Set monitoring per per table, column and event type (insert/update/delete)
  - Definition of optional filter conditions as SQL expression
  - Definition of Kafka topics per table as target
  - Authorization concept with named users and rights assignment on schema level
  - Tracking of configuration changes (history)
  - Generation of triggers based on configuration data
  - Initial transfer of existing data when starting up the CDC tracking of a table
  - Execute changes via Web-GUI as well as by http API calls.

Differentiation from established CDC solutions
Why yet another tool for this purpose

OSP
Otto Group Solution Provider

- Several solutions for change data capture exist, both commercial and open source (Oracle Golden Gate, Quest SharePlex, Red Hat Debezium, etc.).
- Most are based on scanning the transaction logs of a DB (late filtering for relevant events)
- This has no impact on the runtime of the original transactions, but:
  - To compensate the potential unavailability of the target (Kafka) in an automated way requires to keep the transaction logs in DB for the maximum assumed target downtime
  - Taking into account response time, weekends, etc., this usually means at least 3 days.
  - For small proportion of change events in a large transaction processing system, there would be a disproportionate effort and complexity in dealing with transaction logs
- Other pull alternatives like Kafka-Connect on JDBC level need individual structural adjustments in the application to work sufficiently performant.

Our solution approach

OSP
Otto Group Solution Provider

- Use of DB triggers to initially capture the change events that take place
- DB-independent, first implementations for Oracle and SQLite
- Own schema for MOVEX-CDC in source DB, no objects or operations outside this schema, => thus no structure impact on the application to be 'skimmed'.
- Buffering of the change events to be transmitted by triggers in local table of the DB in MOVEX-CDCs schema
  => thus no dependency of the event-triggering transactions on external resources like MOVEX-CDC application or Kafka
- Asynchronous transfer of events from Kafka buffer table to triggering transaction
  Scalable number of parallel threads to ensure timely transmission
- Generation of triggers based on the configuration entered via GUI or JSON import
- Provision of relevant functions by a http API for automating processes

Pros and cons of this solution approach

OSP
Otto Group Solution Provider

Pro:

- Saving resources by filtering for relevant events at the time they occur
- No dependencies or complexities for technical DB operation
- No adaptations of existing applications necessary
- Convenient configuration via GUI, but can also be automated via API

Contra:

- Load on original transactions by trigger (double write)
- Possibly downtime needed for trigger deployment and release update
- Possible coupling of operational risks for all participants

Motivation from first use case in a large scale PIM

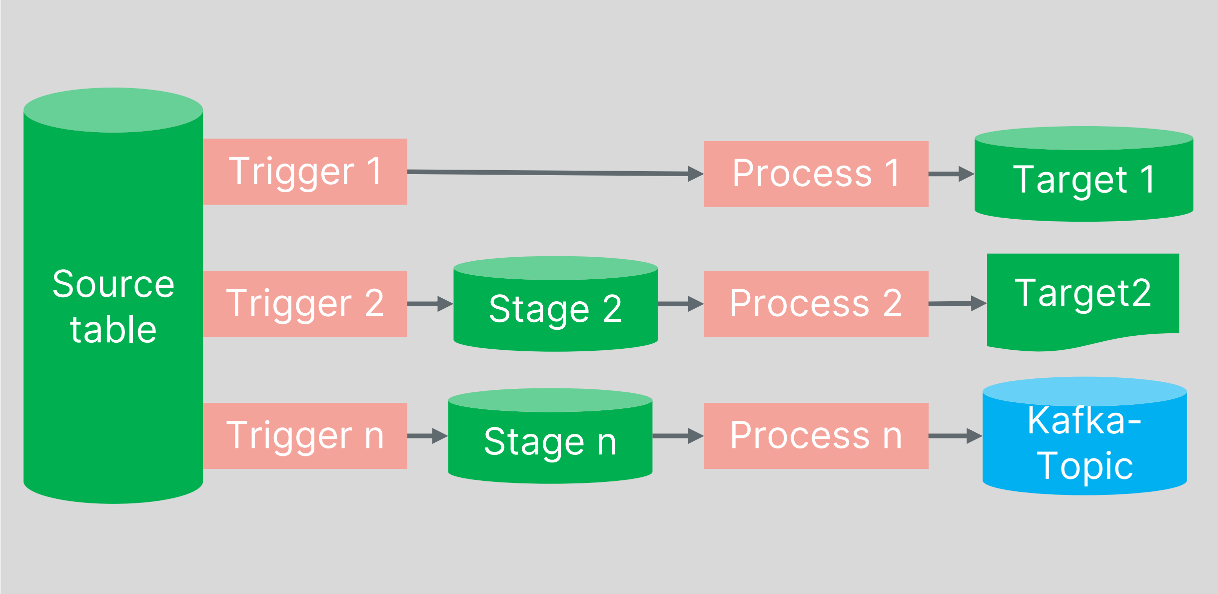
OSP
Otto Group Solution Provider

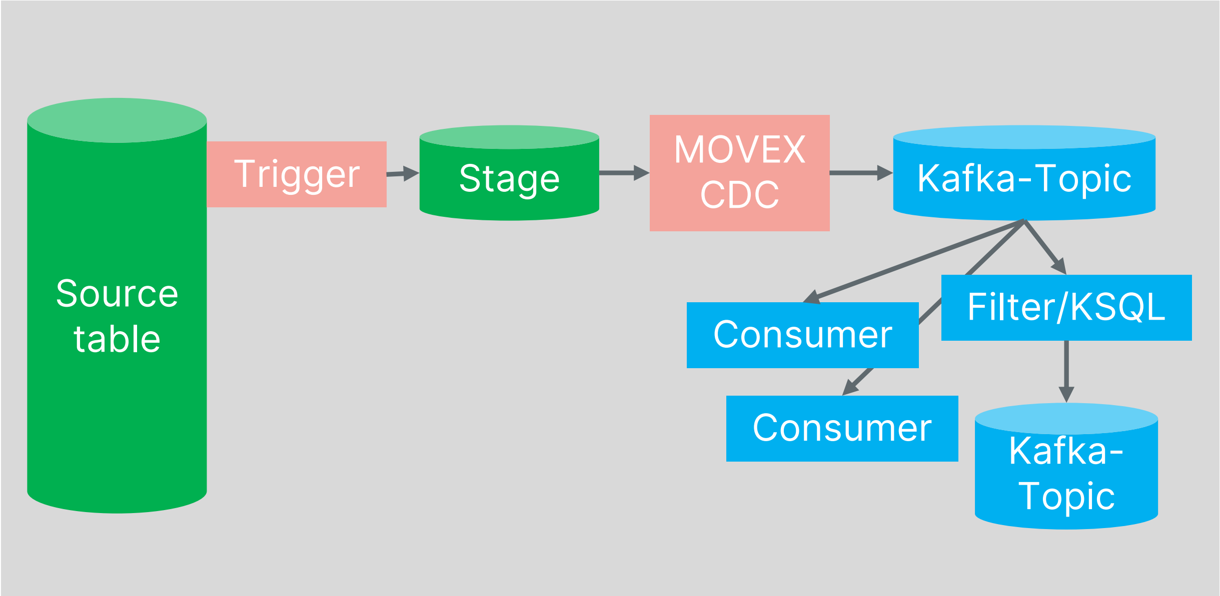
Starting situation:

- Each customer redundantly establishes its own trigger on tables of interest (up to > 100 triggers per table)
- Solutions for further processing of the events in different architecture and quality

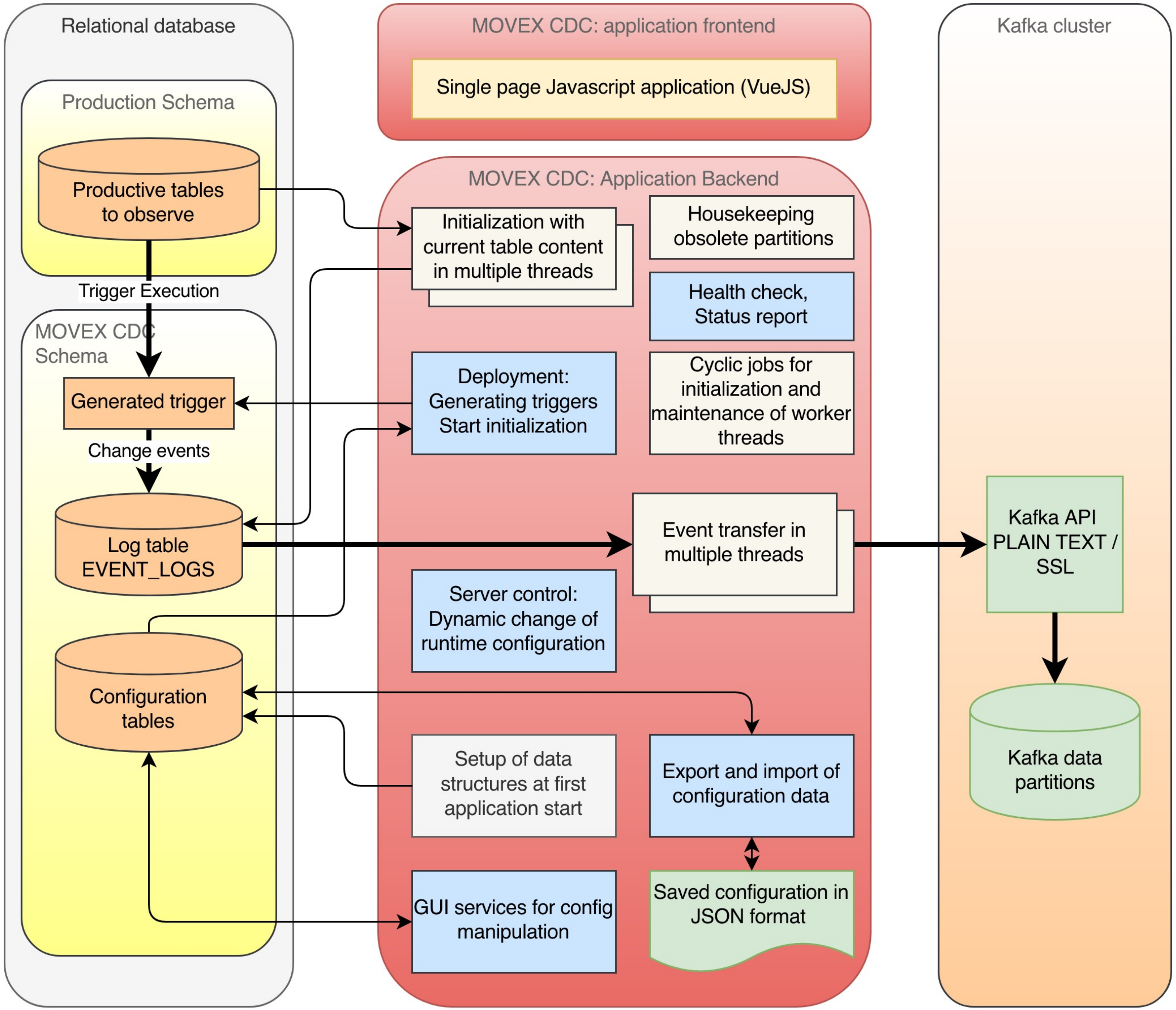
Target scenario with MOVEX-CDC:

- Exactly one trigger per table and event type
- One hardened function for catching events in source DB and transfer to Kafka
- Use publish/subscribe etc. in Kafka

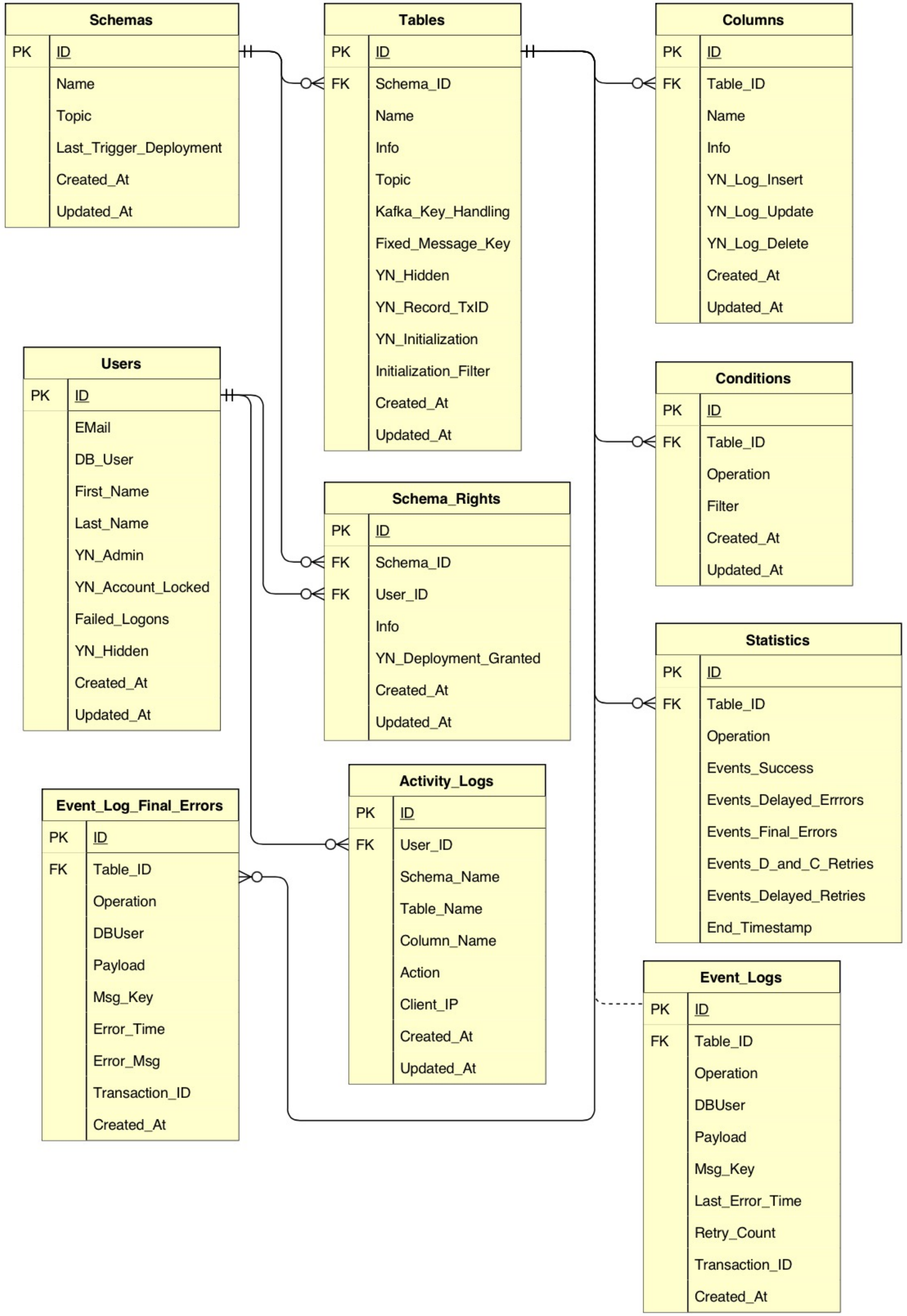




MOVEX-CDC module structure



OSP Otto Group Solution Provider



Entity relationship model

OSP
Otto Group Solution Provider

- Own schema encapsulates all MOVEX-CDC-relevant DB objects
- Export of configuration data to JSON file allows backup outside DB
- Import of configuration data via JSON file e.g. for setup test systems
- Via JSON import generation of MOVEX-CDC configuration from external sources possible
- Separate export/import per DB target schema possible

Supported database systems

OSP
Otto Group Solution Provider

- MOVEX-CDC was developed modular and DB-independent based on Ruby on Rails.

- Runtime environment is a Java VM with jRuby, encapsulated in a Docker container.

- Adaptations are thus in principle imaginable for all relational DB systems with trigger function and available JDBC driver.

Currently supported databases :

- Oracle: all editions with optimization for EE/partitioning
- SQLite: Ensure DB independence in development

Further supported databases planned in the medium term:

- PostgreSQL: Favorite free alternative to Oracle dependency
- MS SQL-Server: For announced use in BI environment
- MySQL / Maria-DB: Possibly if requirements exist

Demo

There's a how-to guide with several steps to install, setup and use MOVEX CDC.

Implement change data tracking on an existing Oracle DB including event transfer to Kafka within 10 minutes:

https://otto-group-solution-provider.gitlab.io/movex-cdc/movex-cdc_demo.html

Implementation: Trigger example

OSP
Otto Group Solution Provider

```
CREATE OR REPLACE TRIGGER T1 FOR INSERT ON SCHEMA.TABLE COMPOUND TRIGGER
… /* Deklariere Memory-Collection payload_tab */

PROCEDURE Flush IS
BEGIN
  … /* Schreibe Memory-Collection payload_tab in Event_Log-table */
END Flush;

BEFORE STATEMENT IS
BEGIN
  payload_tab.DELETE; /* remove possible fragments of previous transactions */
END BEFORE STATEMENT;

AFTER EACH ROW IS
BEGIN
  … /* Schreibe JSON-Record in Memory-Collection, Flush wenn > 1000 Records */
END AFTER EACH ROW;

AFTER STATEMENT IS
BEGIN
  Flush; /* Flush Collection in Table */
END AFTER STATEMENT;

END T1;
```

Implementation: Uniqueness

Otto Group Solution Provider

Uniqueness at the target Kafka:

- Each change event recorded in DB is transferred to Kafka and committed exactly once

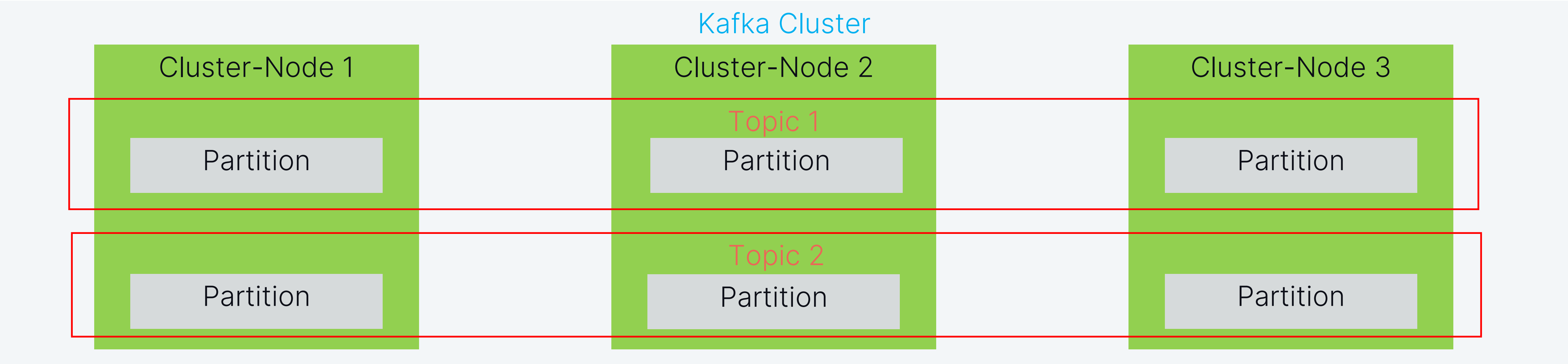
- A non commited transmission to Kafka can occur several times if repeated on error
  - Kafka distinguishes between read_uncommited and read_commited when consuming

- Each event has a unique sequential event ID created by a DB sequence

- Transactional coupling between the two resources DB and Kafka is implemented with two nested transactions in the MOVEX CDC application
- There are no XA or 2-phase commit transactions between them
- Due to this, there is at least a tiny hypothetically risk of double transfers

Implementation: guaranteed sequences

- Kafka guarantees delivery of events in the order of their creation only within a partition
- Events with the same key value always end up in the same partition in Kafka
- MOVEX CDC supports optional keys: no key, primary key, fixed value or transaction ID
- MOVEX-CDC only transmits events with the same key value in an ordered sequence, all others without guarantee of the sequence (conflict of objectives with parallel processing)



Implementation: Horizontal scalability

OSP
Otto Group Solution Provider

- Bottleneck in the transfer between trigger event and Kafka is the transfer of events from the staging table EVENT_LOGS to Kafka

- Scalability is given by configurable number of worker threads in the MOVEX CDC application, each working isolated with own DB and Kafka session
  - Depending on the capacity of the runtime env. (CPU, network) several 100 threads are possible

- The allocation / synchronization of the events from EVENT_LOG to the worker threads is controlled by DB-Locks (SELECT ... FOR UPDATE SKIP LOCKED)

- The guarantee of the order for events with key is ensured by processing events with the same key only by exactly one worker thread in the order of their occurrence
  - Distribution of keys to threads per modulo on a hash value of the key
  - Sequence violation can occur if DB transaction is committed only after successors with the same key from other DB transactions have already been transferred to Kafka.

Implementation: Fail-safe / Instance redundancy

OSP
Otto Group Solution Provider

- Synchronization via DB locks using SELECT ... FOR UPDATE SKIP LOCKED would in principle also allow several MOVEX-CDC instances to be actively operated in parallel.
- However, in this mode of operation the sequence of events can no longer be guaranteed with key
- Hot redundancy with multiple active instances should normally not be necessary, because:
  - An instance has enough potential for throughput optimization by thread scaling
  - A continous gapless operation of the MOVEX-CDC application is not mandatory :
    - For catching the events via DB trigger no running MOVEX CDC instance is needed
    - Suspension of MOVEX-CDCs Docker container does not lead to data loss, but only to delay in transmission to Kafka
    - This allows scenarios such as version updates, changes in runtime environment, etc. to be carried out with short downtimes during ongoing production operation

Implementation: Bulk operations

OSP
Otto Group Solution Provider

The process chain works consistently with bulk operations:

- Trigger implemented as compound trigger with
  - Limitation to max. 1000 JSON records buffered in PL/SQL session memory
  - Bulk operation for insert in stage table EVENT_LOGS

- Read records from EVENT_LOGS with SELECT FOR UPDATE SKIP LOCKED
  - No indexes for EVENT_LOGS means: Full Table Scan for each access to this table
  - Predictable load through interval partitioning with housekeeping of empty partitions
  - Max. size of DB transaction towards Kafka is configurable (default: 10.000)

- Transfer of events to Kafka (Produce) with transaction in Kafka cluster
  - Size of the transaction corresponds to DB transaction
  - Bulk size when transferring to Kafka is limited by Kafka (default 1000, configurable)

Implementation: Fault tolerance

OSP
Otto Group Solution Provider

- In case of transmission errors / rejection of events by Kafka a Divide&Conquer procedure takes effect
- The number of events transmitted by bulk operation is reduced until only a single event is processed. Among other things, this ensures immediate retry several times.
- If an isolated event still remains erroneous, it is marked and retried with a time delay. After x unsuccessful attempts, this event will be sorted out in the error table.
- From error table, events can be manually activated for post-processing, otherwise they will be permanently deleted after a holding period
- Reasons for not broadcasting events can be e.g. :
  - Non-existent Kafka topic
  - Exceeding the permissible event size
  - Configuration without key although log compaction was configured on Kafka side

Implementation: Performance of DB actions (EE)

OSP
Otto Group Solution Provider

- The staging table of the events is an interval-partitioned table without any index. This ensures minimal overhead and maximum availability when inserting the event data in the productive transactions by trigger.
- The partitioning interval as well as the maximum number of simultaneous transactions (INI_TRANS) are controllable via the configuration of MOVEX CDC.
- Fully processed partitions are promptly dropped by a housekeeping process.
- Since the events are stored in a table without indexes, this means that reading the events for transmission to Kafka can only be done via Full Table Scan.
- Interval partitioning ensures a limit to the amount of data to be read via full table scan
- Even with temporarily massive data traffic, the reading effort due to Full Table Scan is reduced again with the next partition change (no problem with non-reducible high water mark).

Implementation: Behaviour of DB actions (SE)

OSP
Otto Group Solution Provider

- For Oracle Standard Edition rsp. Enterprise Edition without Partitioning Option the staging table EVENT_LOGS is implemented as a regular heap table with an index on column ID.

- That means: several optimizations based on partitioning do not take place.

  - The staging table EVENT_LOGS needs an index on column ID for proper performance. This adds additional index maintenance load on triggering transaction and a very tiny risk of blocking between transactions at index block split operations.

  - The high water mark of table EVENT_LOGS is not automatically reduced after peak usage.

  - Additional reorganization activities on staging table EVENT_LOGS can by necessary from time to time depending on type and frequency of usage:
    - ALTER TABLE Event_Logs MOVE; to reduce the high water mark
    - ALTER INDEX Event_Logs_PK REBUILD; to reduce the size of the index

Performance parameters / limitations

OSP
Otto Group Solution Provider

The achievable throughput depends in reality strongly on:

- Database performance
- Performance of the Kafka cluster
- Network latency and throughput / distance between DB, MOVEX-CDC instance and Kafka cluster
- Number of worker threads
- Size of the JSON structure of the events, from 4 KB on, Oracle stores in significantly slower CLOB structures instead of heap tables.
- Example throughput for small distance between DB, MOVEX CDC and Kafka with 3 worker threads and JSON < 4K:
  820,000 events per minute 1.18 billion events per day

Application operation

OSP
Otto Group Solution Provider

- Delivery artifact of MOVEX-CDC is exactly one consistent Docker image
- Configuration is done via a config file or environment variables
- Complete self-initialization of the system in DB schema at the start of the container.
- Logging is done via console output of the Docker container, logging level can be changed dynamically via GUI or API
- Operating status can be queried via HealthCheck API / monitored externally
- Operating statistics (throughputs, error rates, etc.) are collected in table "Statstistics" every minute, condensed after some time
- Temporary downtime or inaccessibility of DB or Kafka will be tolerated. After resources are available again, operation will be resumed without further external activity.
- Several configuration parameters can be adjusted at runtime, like no. of worker threads

Thank you for your interest

Resources:

Project root:	https://gitlab.com/otto-group-solution-provider/movex-cdc
Documentation:	https://otto-group-solution-provider.gitlab.io/movex-cdc/movex-cdc.html
Quick start howto:	https://otto-group-solution-provider.gitlab.io/movex-cdc/movex-cdc_demo.html

Otto Group Solution Provider (OSP) GmbH
Freiberger Str. 35 01067 Dresden
Telefon +49 (0) 351 49723 0
www.osp.de