Kubernetes Native Change Data Capture

Conor Gallagher

What problem is being solved and why?

Problem Statement

Teams across Zalando have adopted streaming architectures, where events are published to our event bus (Nakadi) for entities they manage in data stores (e.g. PostgreSQL or DynamoDB).

Ensuring data store writes and event publishing is atomic is known as the Distributed Commit Problem. Until recently, it had no standard solution at Zalando.

Goals

Standardise event generation across Zalando

Solve the distributed commit problem centrally, so builders can focus on business logic in their applications

Define a Kubernetes Custom Resource Definition to insulate builders from the the complexities of Change Data Capture

Change Data Capture (CDC)

Anatomy of a Postgres Change Record


```
{
  "before": {
    "id": 1,
    "first_name": "Bob",
    "status": "INACTIVE"
  },
  "after": {
    "id": 1,
    "first_name": "Bob",
    "status": "ACTIVE"
  },
  "source": {
    "version": "1.1.0.Final",
    "connector": "postgresql"
    "name": "postgres",
    "ts_ms": 1041325582629,
    "snapshot": "false",
    "db": "postgres",
    "schema": "public",
    "table": "table_two",
    "txId": 560,
    "lsn": 24620824,
    "xmin": null
  },
  "op": "u",
  "ts_ms": 1589455942003,
  "transaction": null
}
```


- `before` is an optional field that if present contains the state of the row *before* the event occurred.
- `after` is an optional field that if present contains the state of the row *after* the event occurred.

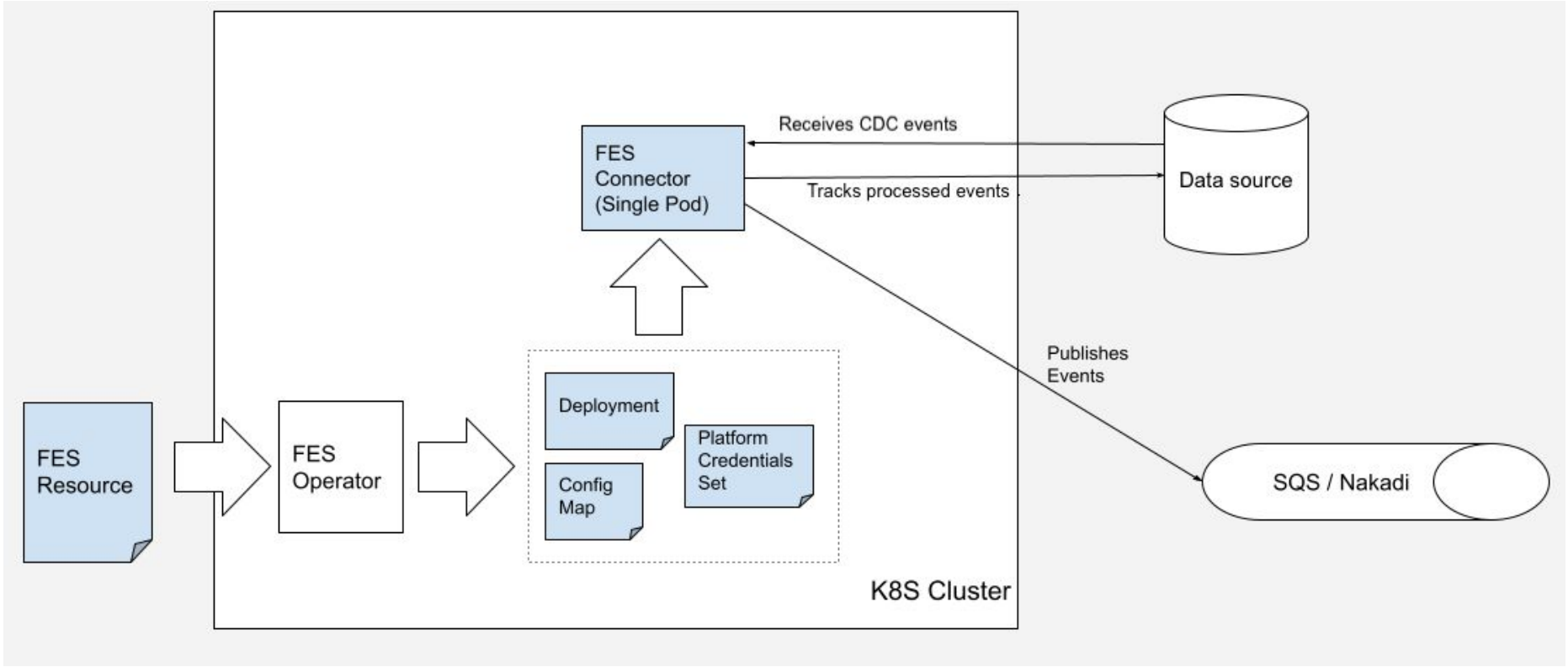
Fabric Event Streams (FES)

FES - Overview

A Kubernetes native Change Data Capture (CDC) eventing solution, consisting of two components:

1. **FES Operator** - The Kubernetes control-plane for FES, containing the `FabricEventStream`Custom Resource Definition (CRD)
2. **FES Connector**:
    - For Postgres, the connector uses Debezium to connect via Logical Replication to a Postgres databases for CDC
    - For DynamoDB, the connector uses the Kinesis Client Library (KCL) to connect to a DynamoDB Stream

FES - Architecture



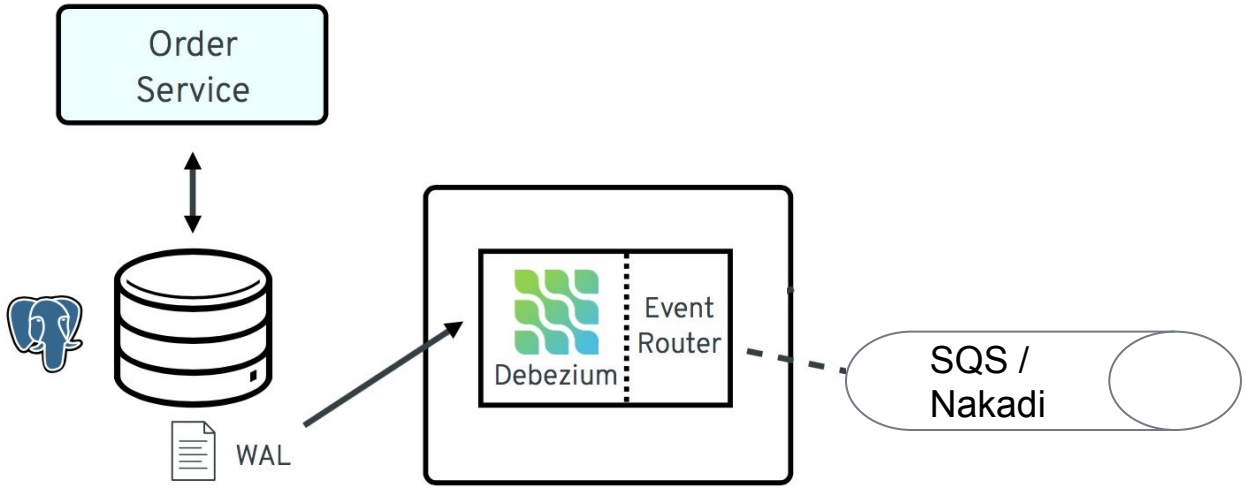
FES - Custom Resource Definition

Inspired by Akka Streams, the `FabricEventStream`CRD is modelled around Sources, Flows, Sinks, and Recovery:

- **Source** - A stage with exactly one output
- **Flow** - An optional stage which has exactly one input and output. The Flow connects the Source and Sink by transforming the events flowing through it
- **Sink** - A stage with exactly one input. The sink defines the terminal point of the stream, a Nakadi event type or SQS queue to publish events to
- **Recovery** - Defines how the stream should handle bad events

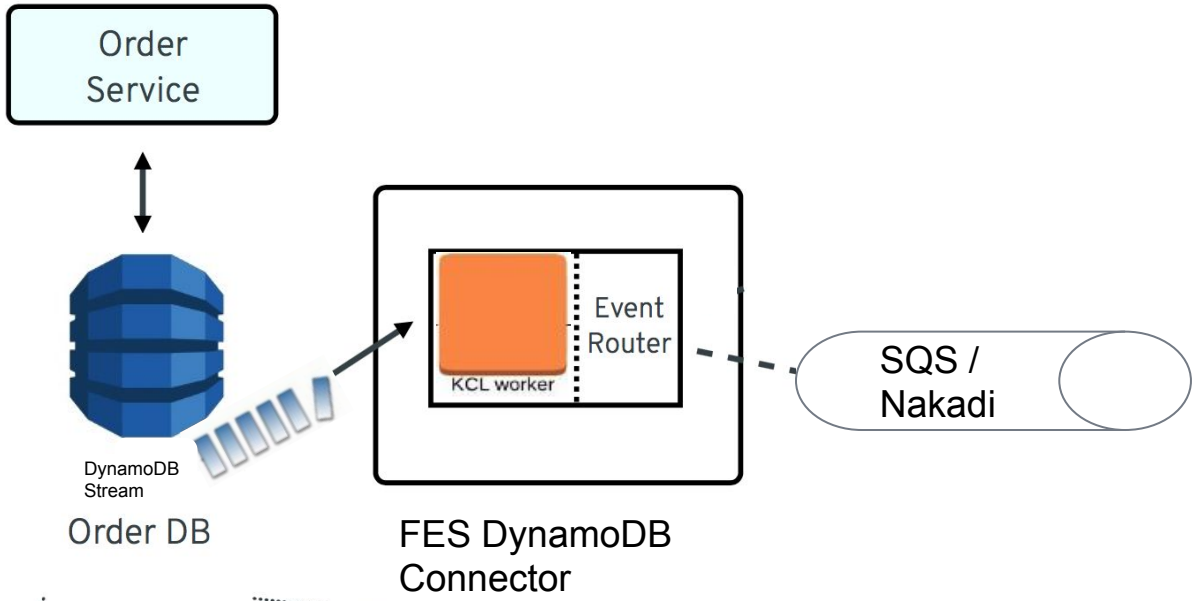
FES - Resource Example

```yaml
spec:
  applicationId: my-application
  eventStreams:
  - source:
      type: PostgresLogicalReplication
      jdbcConnection:
        slotName: fes
        jdbcUrl: "jdbc:postgresql://..."
      table:
        name: my_events_outbox
    flow:
      type: PostgresWalToGenericNakadiEvent
      payloadColumn: "my_event_payload"
    sink:
      type: Nakadi
      eventType: "my-important-business-events"
    recovery:
      type: DeadLetter
      sink:
        type: SqsStandard
        queueName: "my-dead-letter-queue"
```



Id	AggregateType	AggregateId	Type	Payload
ec6e	Order	123	OrderCreated	{ "id" : 123, ... }
8af8	Order	456	OrderDetailCanceled	{ "id" : 456, ... }
890b	Customer	789	InvoiceCreated	{ "id" : 789, ... }

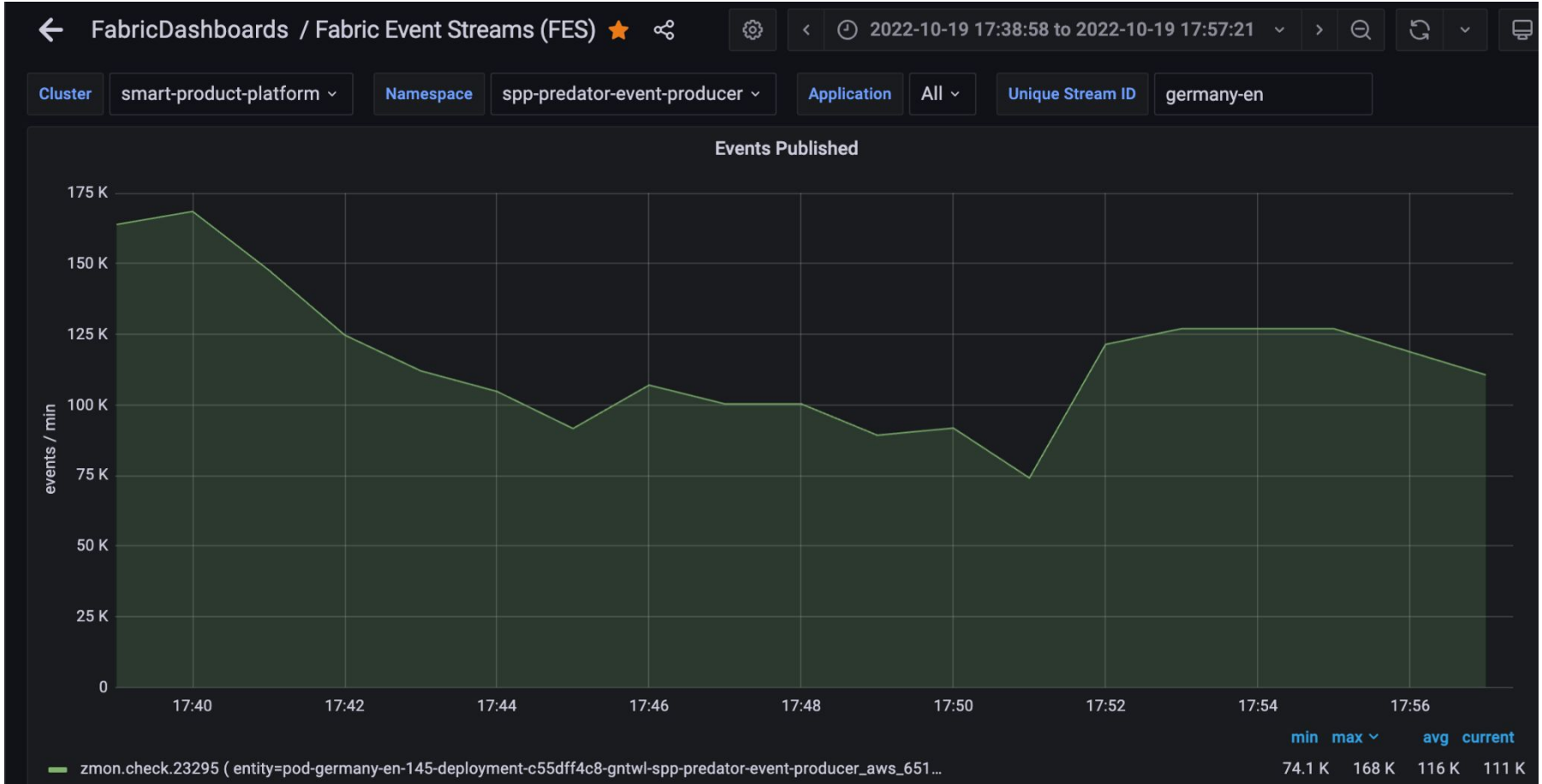
Outbox Table



Id	AggregateType	AggregateId	Type	Payload
ec6e	Order	123	OrderCreated	{ "id" : 123, ... }
8af8	Order	456	OrderDetailCanceled	{ "id" : 456, ... }
890b	Customer	789	InvoiceCreated	{ "id" : 789, ... }

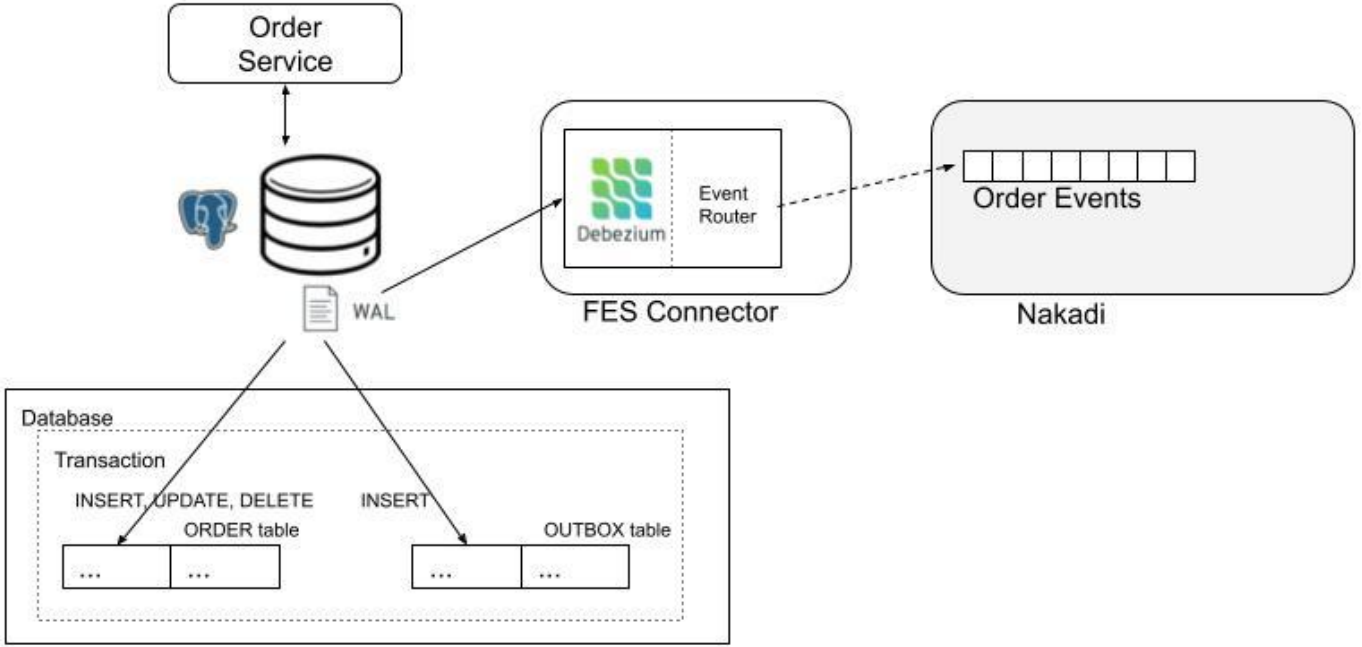
Outbox Table

FES - Standardised Telemetry



Patterns

Transactional Outbox Pattern

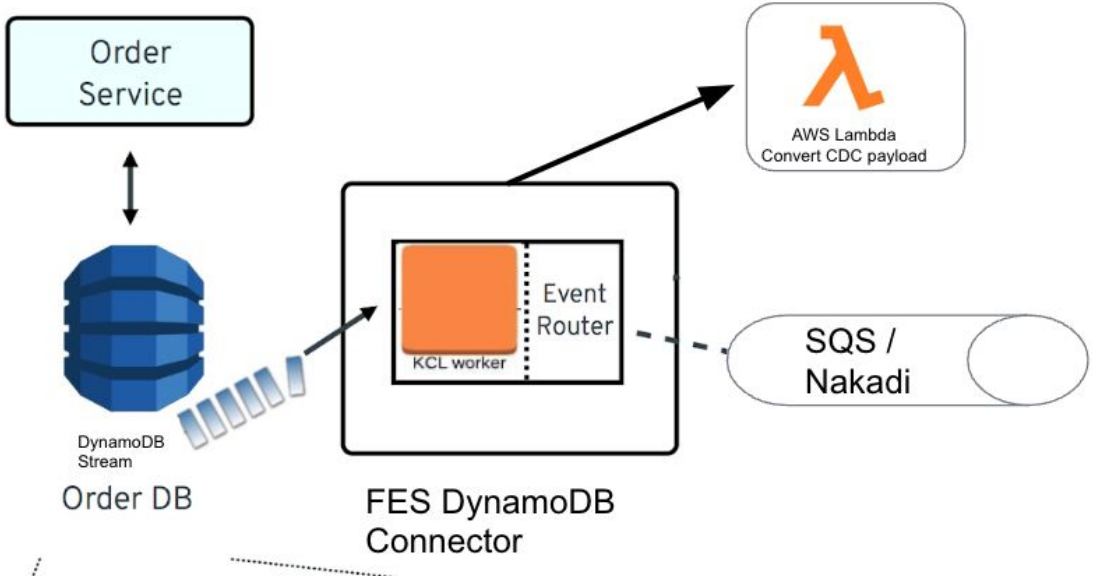


zalando

Transactional Outbox Pattern

```
spec:
  applicationId: my-application
  eventStreams:
  - source:
      type: PostgresLogicalReplication
      jdbcConnection:
        slotName: fes
        jdbcUrl: "jdbc:postgresql://..."
      table:
        name: my_events_outbox
    flow:
      type: PostgresWalToGenericNakadiEvent
    sink:
      type: Nakadi
      eventType: "my-important-business-events"
```

AWS Lambda Flow



Id	AggregateType	AggregateId	Type	Payload
ec6e	Order	123	OrderCreated	{ "id" : 123, ... }
8af8	Order	456	OrderDetailCanceled	{ "id" : 456, ... }
890b	Customer	789	InvoiceCreated	{ "id" : 789, ... }

AWS Lambda Flow

```
spec:
  applicationId: my-application
  eventStreams:
  - source:
      type: DynamoDbStreamsSubscription
      subscription:
        leaseTableName: my_lease_table
        streamArn: "arn:aws:dydb:eu-1:343:my_table/stream/2022-05-"
      filter: "[?(@.OldImage.Status.Id != @.NewImage.Status.Id)]"
    flow:
      type: AwsLambdaConversion
      lambdaFunctionArn: "arn:aws:lambda:eu-1:343:func:cdc-converter"
    sink:
      type: SqsFifo
      queueName: "my-cdc-queue"
    recovery:
      type: None
```

zalando

Questions?