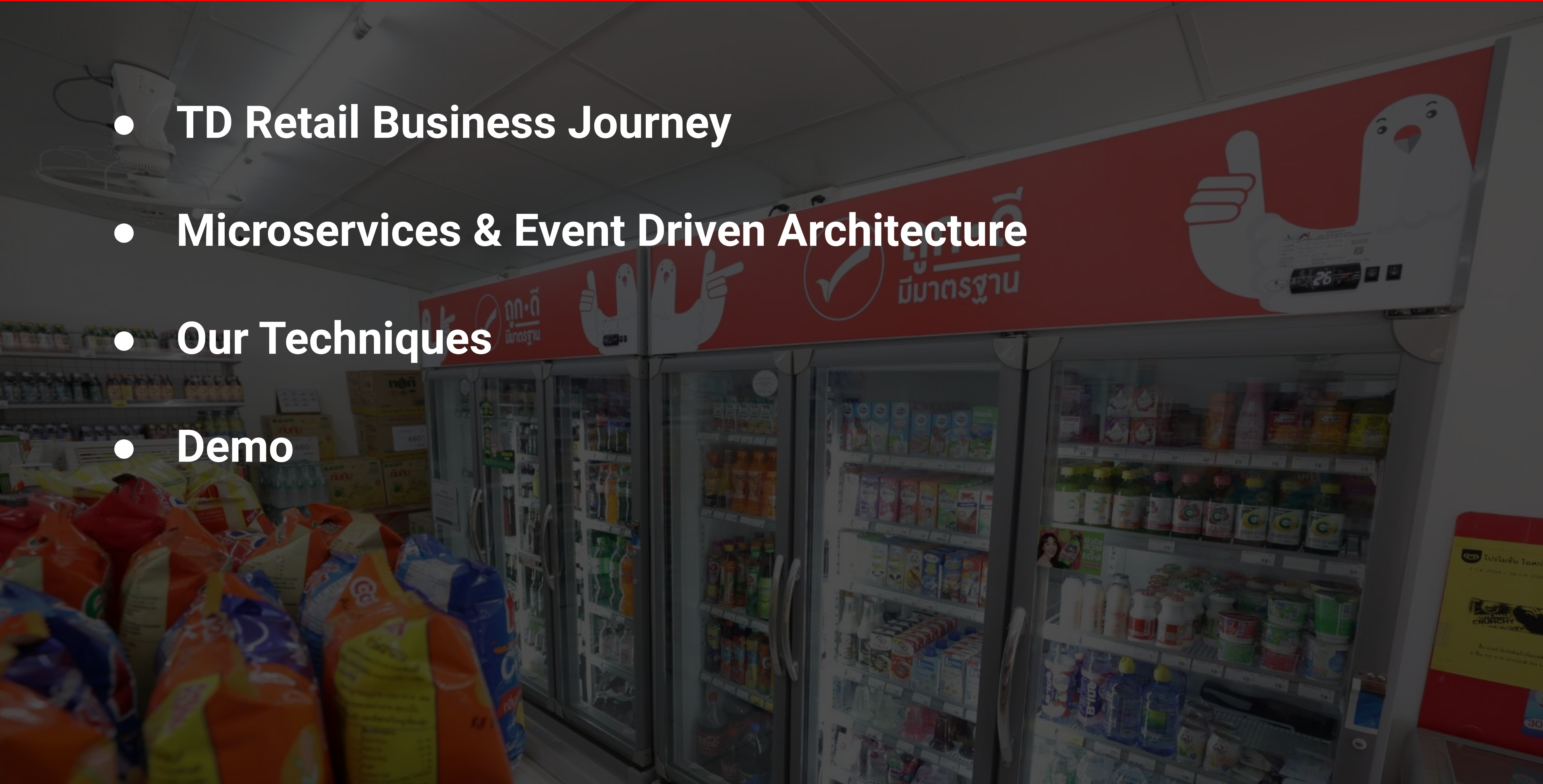
Event Driven Architecture in TD Retail Platform

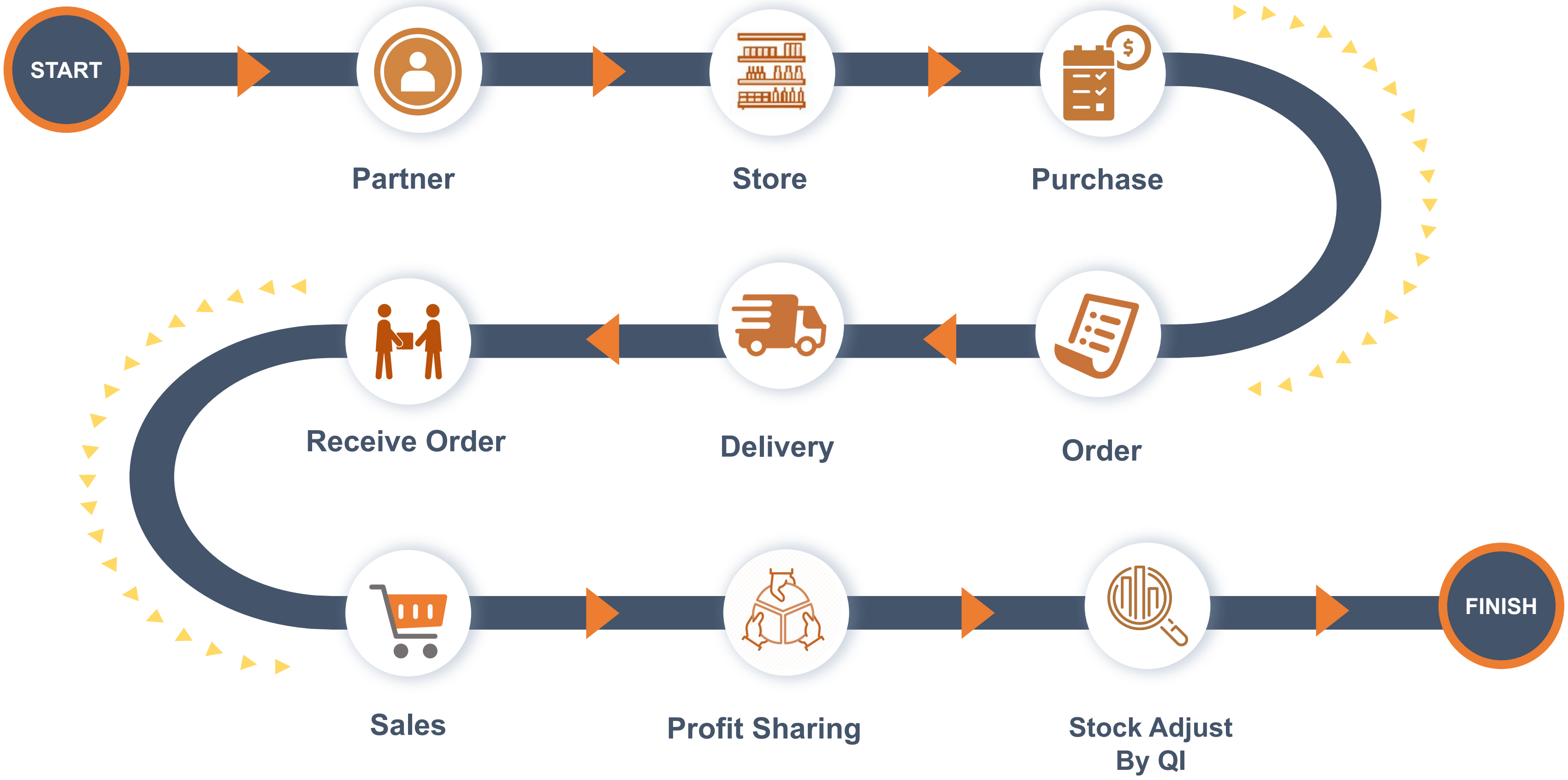
Sattra Sowanit,
Chief Software Architect,
TD Tawandang

Agenda

- TD Retail Business Journey
- Microservices & Event Driven Architecture
- Our Techniques
- Demo

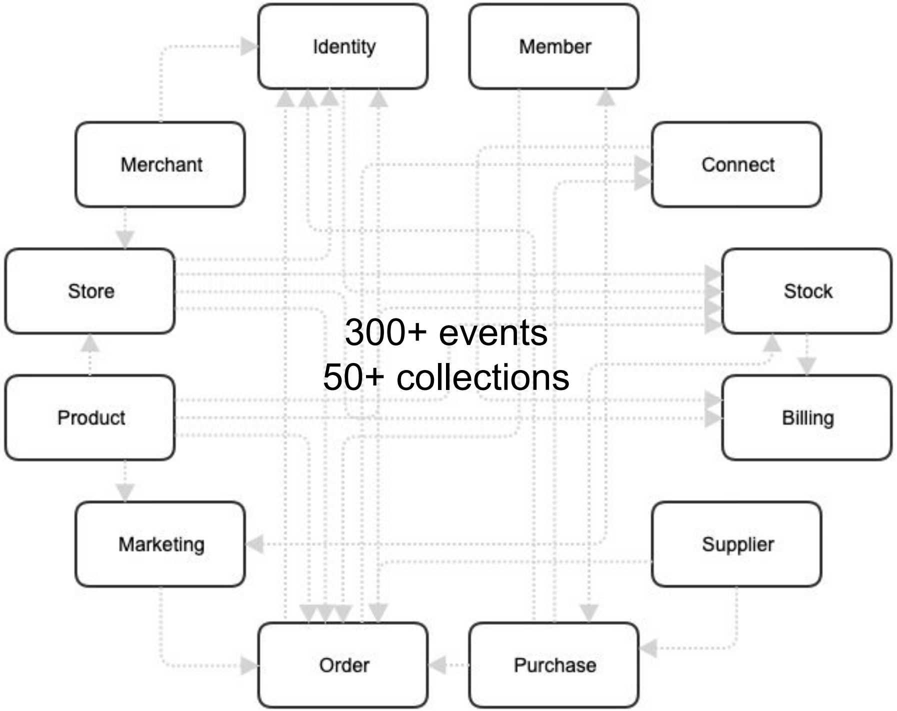


TD Retail Business Journey



Microservices & Event Driven Architecture

- **Microservices**



Clients

- 5K+ POS devices
- 5.5K+ handheld devices

Workloads

- 5K+ stores / 8 DC
- 800K+ bills / day
- 2K+ order / day
- 3.5M+ stock moving / day

Microservices

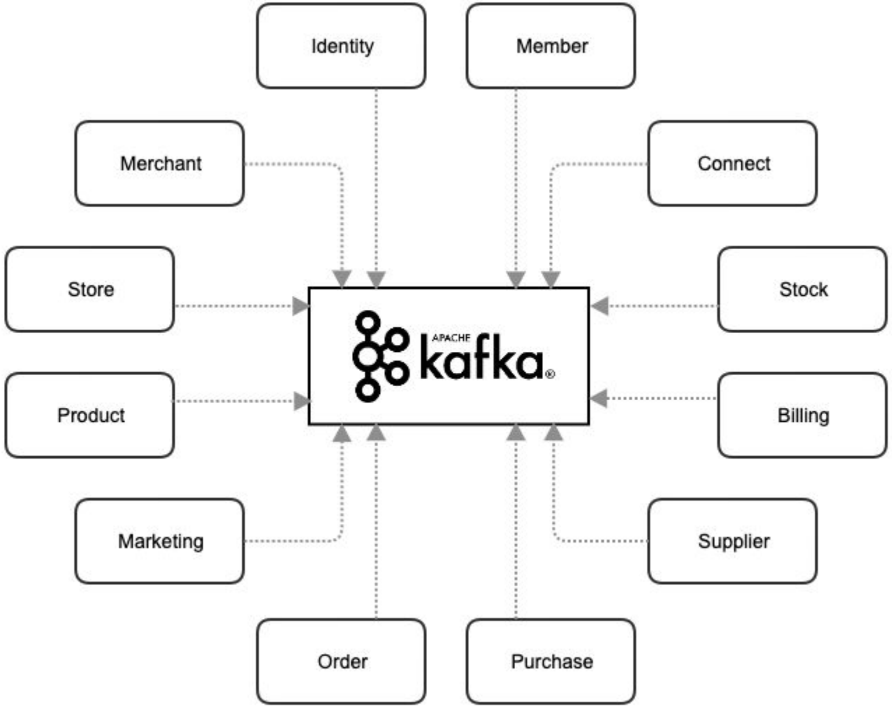
- 12 core modules
- 60+ sub-modules

Runtime Platform

- Google Kubernetes Engine
- 30+ Worker Nodes
- 140+ Deployments
- 300+ Pods

Microservices & Event Driven Architecture

- **Event and Data Streaming**



Event Source

- 300+ event types
- 200+ records / second
- 30+ kilobytes / second
- 20+ connectors

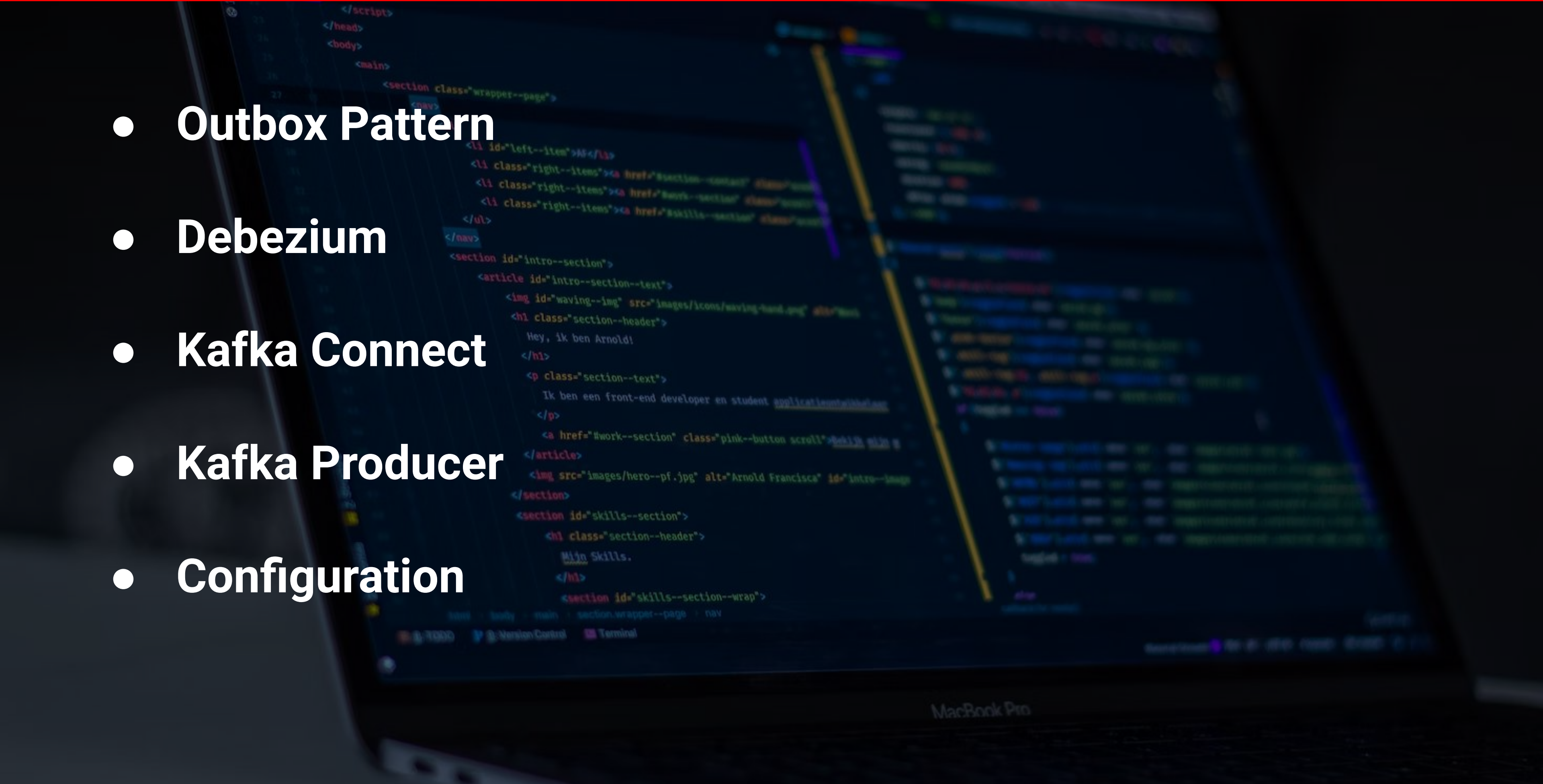
Change Data Capture

- 50+ collections
- 200+ records / second
- 400+ kilobytes / second
- 30+ connectors

Runtime Platform

- Confluent Cloud
- Self-managed Connect Cluster
  - 20+ workers

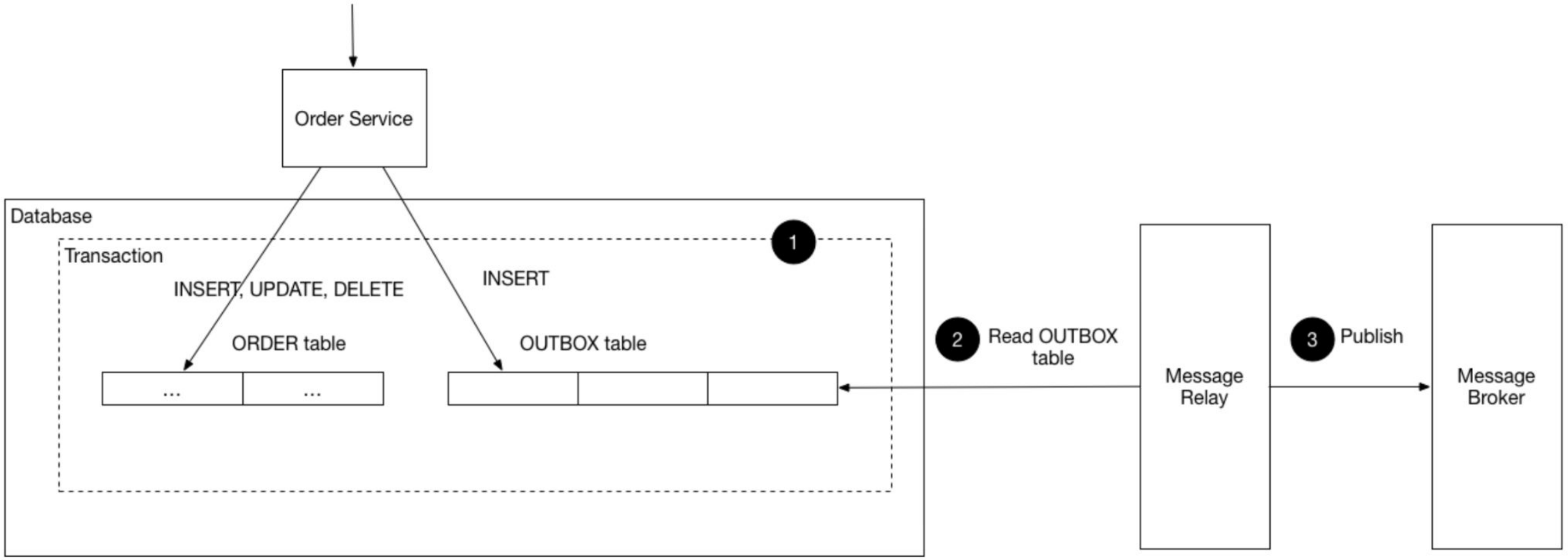
Our Techniques



- Outbox Pattern
- Debezium
- Kafka Connect
- Kafka Producer
- Configuration

Techniques - Outbox Pattern

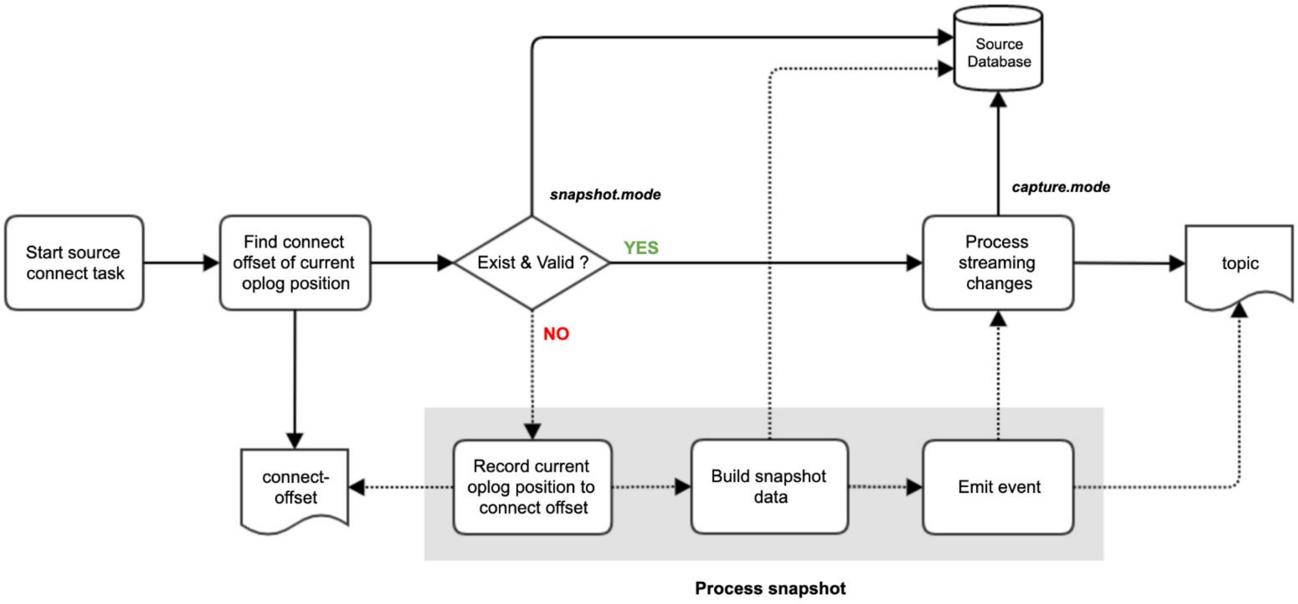
- **Data and Event Consistency**

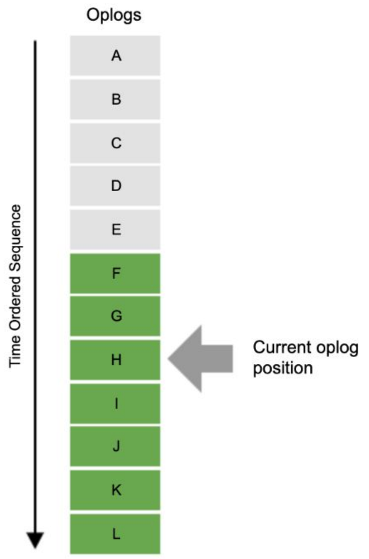


https://microservices.io/patterns/data/transactional-outbox.html

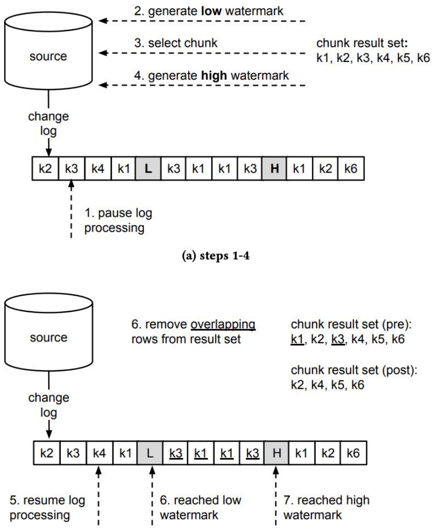
Techniques - Debezium

- **Source and Event Consistency**


Source Database
snapshot.mode
capture.mode
Start source connect task
Find connect offset of current oplog position
Exist & Valid ?
YES
NO
Process streaming changes
topic
connect-offset
Record current oplog position to connect offset
Build snapshot data
Emit event
Process snapshot



Techniques - Debezium



(a) steps 1-4

(b) steps 5-7

Figure 3: Watermark-based Chunk Selection

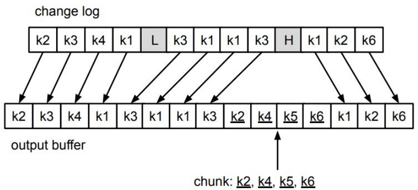

change log
k2 k3 k4 k1 L k3 k1 k1 k3 H k1 k2 k6
k2 k3 k4 k1 k3 k1 k1 k3 k2 k4 k5 k6 k1 k2 k6
output buffer
chunk: k2, k4, k5, k6

Figure 4: Order of output writes. Interleaving log capture with full data capture.

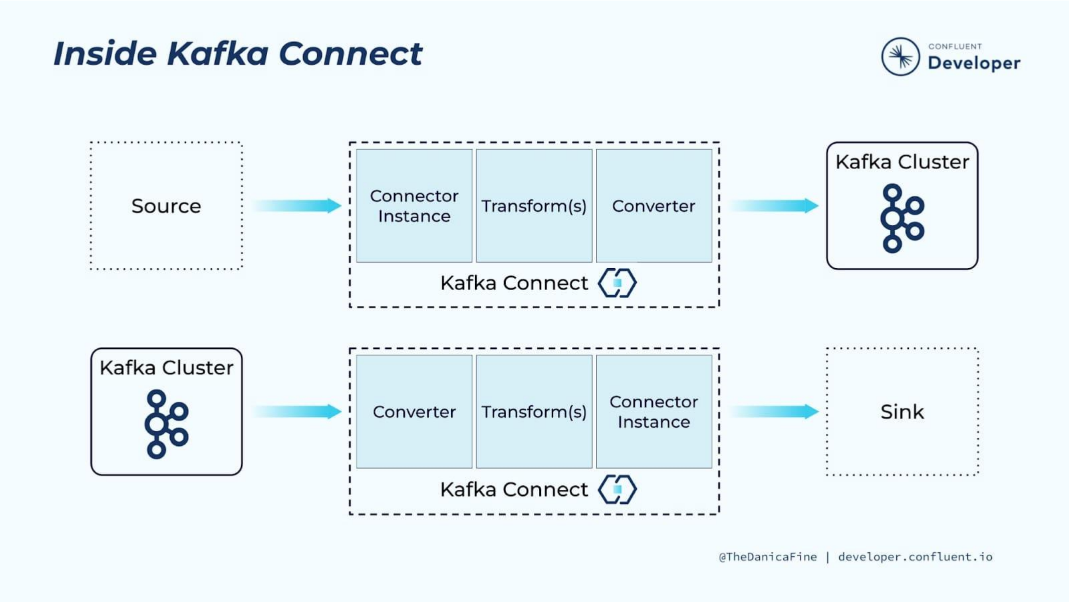
- ## Legacy Snapshot (Debezium <= 1.5)
  - The near-impossibility of adding of additional tables to the captured tables list, if existing data must be streamed
  - A **long-running process** for consistent snapshotting that cannot be terminated or resumed
  - Change data streaming being **blocked** till the snapshot is completed

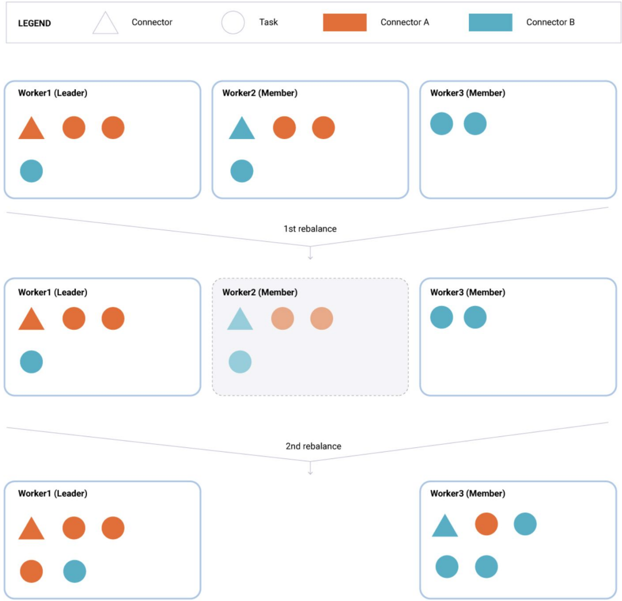
- ## Incremental Snapshot (Debezium >= 1.6)
  - Watermark-based Snapshot (Netflix)
  - Signalling Table

https://arxiv.org/pdf/2010.12597v1.pdf

Techniques - Kafka Connect

- **Scalability and Fault Tolerance**

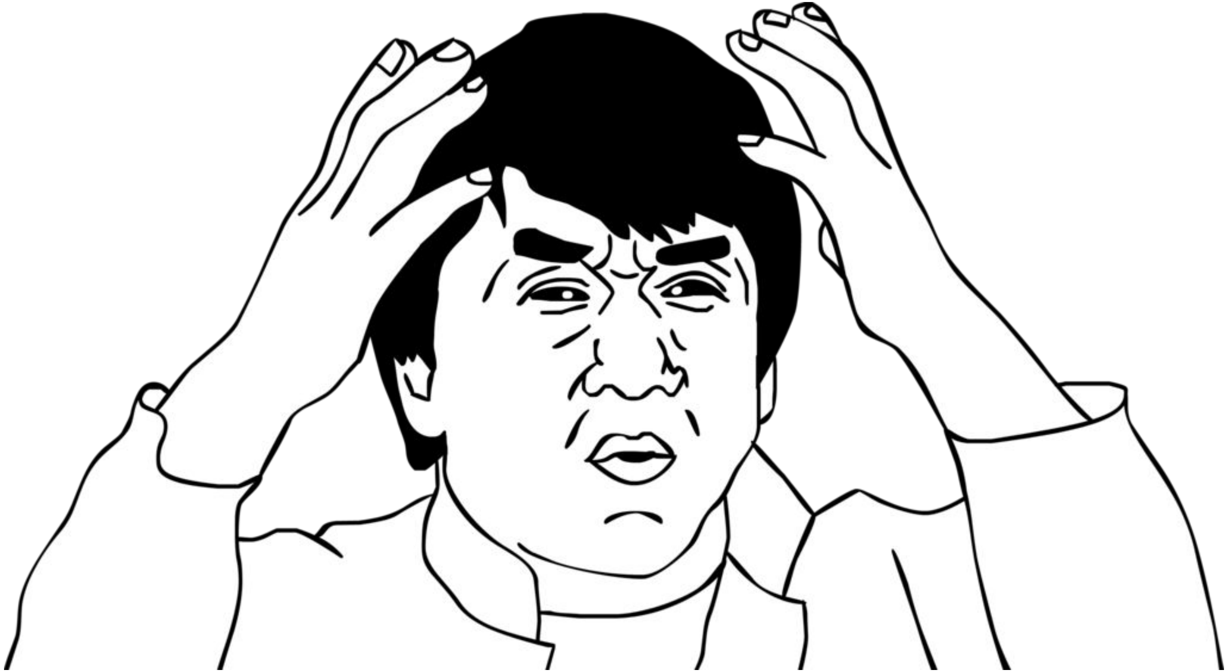




https://developer.confluent.io/learn-kafka/kafka-connect/how-connectors-work/

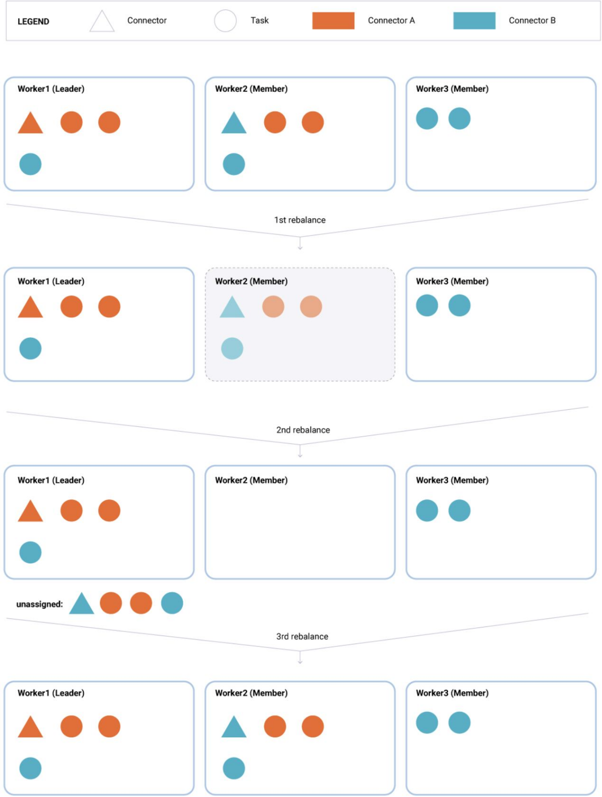
Techniques - Kafka Connect

- Stop the World Rebalancing (Kafka <= 2.2)



Techniques - Kafka Connect

- KIP-415 Incremental Cooperative Rebalancing (Kafka >= 2.3)



Goals

- Address the challenges at **large scale**

Why incremental?

- No need to reach final state within **single rebalance** round
- A **grace period** is configurable
- The protocol coverages smoothly to a state of balance load

Why cooperative?

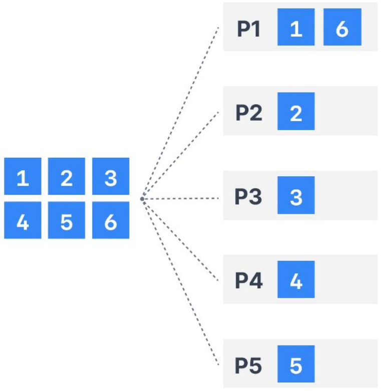
- Resource **revocation** and **release** is graceful

https://www.confluent.io/blog/incremental-cooperative-rebalancing-in-kafka/

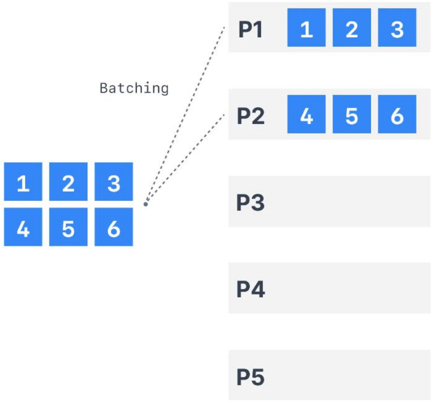
Techniques - Kafka Producer

- **Partition Strategy**

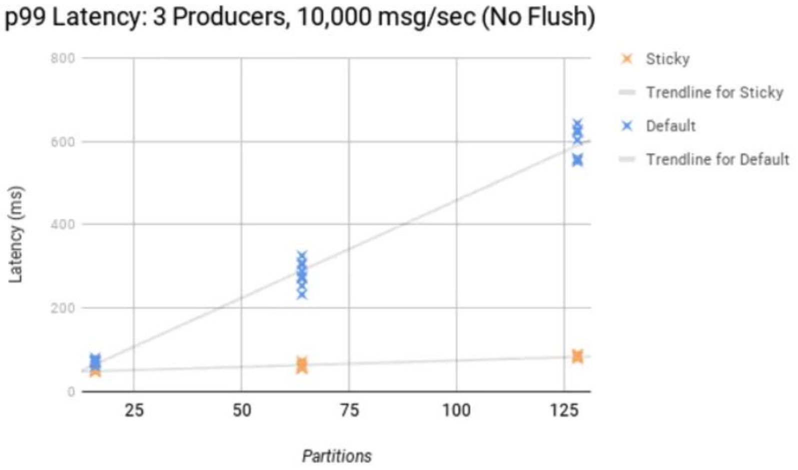
Direct, **Key Hashing**, Round Robin (Kafka <=2.3) and Sticky (Kafka >=2.4)



Round Robin



Sticky
(batch.size, linger.ms)



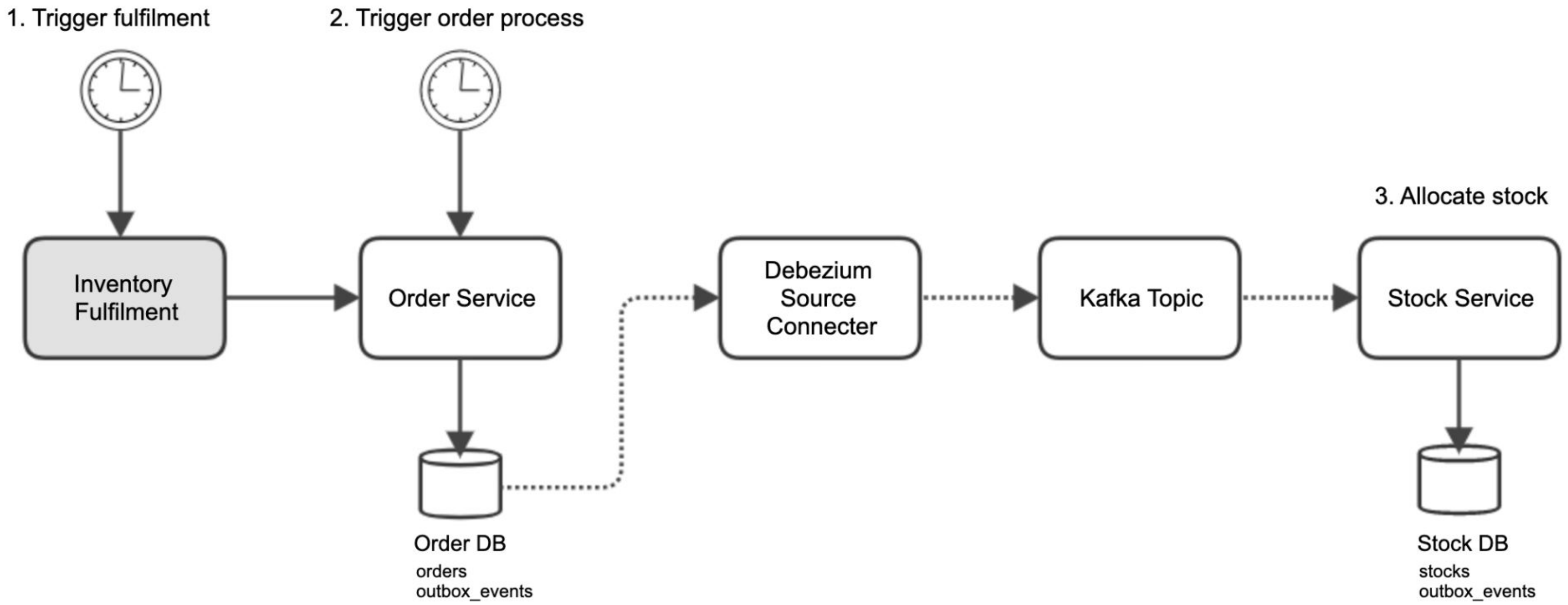
https://www.conduktor.io/kafka/producer-default-partitioner-and-sticky-partitioner

Techniques - Configuration

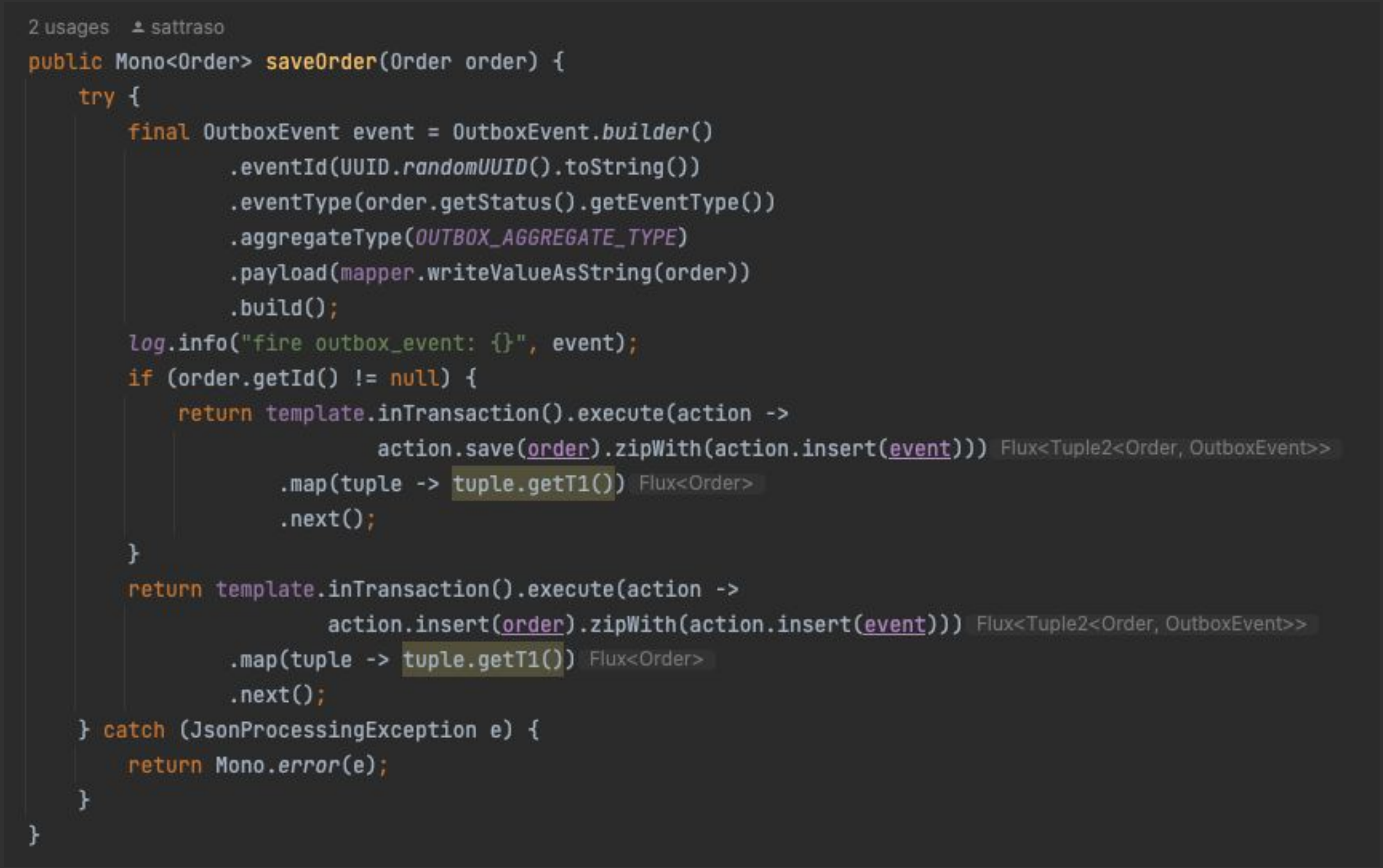
Type	Name	Used By	Default Value	Our Value
Connect	task.shutdown.graceful.timeout.ms	Fault tolerance	5000	15000
Connect	scheduled.rebalance.max.delay.ms	Incremental cooperation rebalance	300000	300000
Connect	offset.flush.timeout.ms	Large snapshot processing	5000	15000
Connect	offset.flush.interval.ms	Large snapshot processing	60000	30000
Producer	batch.size	Sticky partitioning	16384	524288
Producer	linger.ms	Sticky partitioning	0	5000
Producer	max.request.size	Large snapshot processing	1048576	8388608
Consumer	heartbeat.interval.ms	Fault tolerance	3000	5000
Consumer	session.timeout.ms	Fault tolerance	45000	60000
Consumer	max.poll.interval.ms	Fault tolerance	300000	600000
Consumer	max.poll.records	Fault tolerance	500	200

Demo - Use Case

- **Stock is allocated after order has been processed**

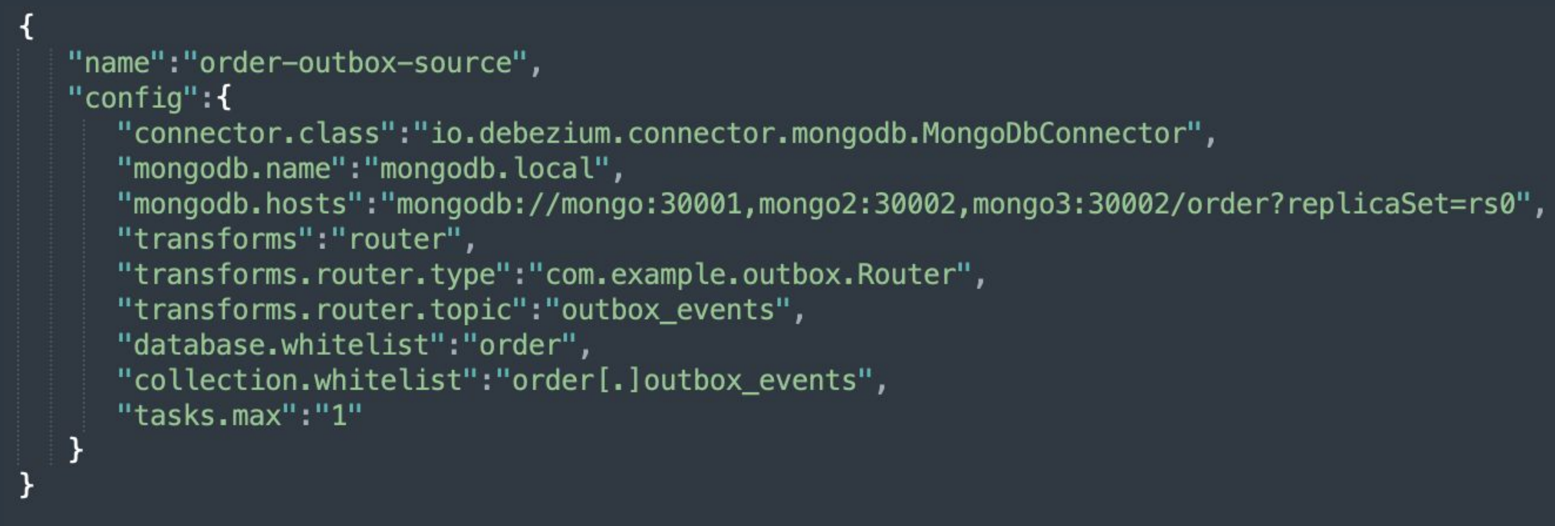


Demo - Event Source



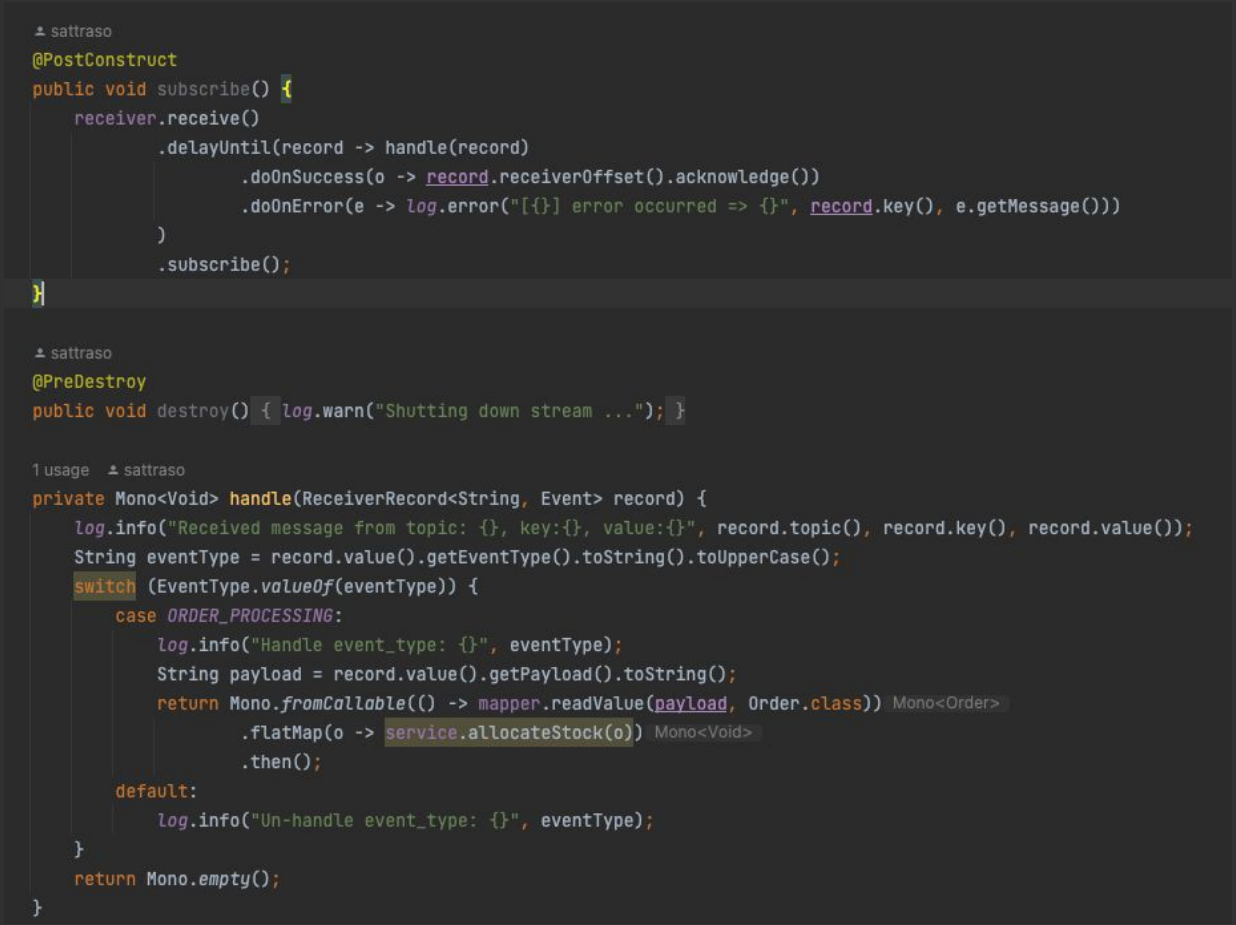
```
2 usages  sattraso
public Mono<Order> saveOrder(Order order) {
    try {
        final OutboxEvent event = OutboxEvent.builder()
                .eventId(UUID.randomUUID().toString())
                .eventType(order.getStatus().getEventType())
                .aggregateType(OUTBOX_AGGREGATE_TYPE)
                .payload(mapper.writeValueAsString(order))
                .build();
        log.info("fire outbox_event: {}", event);
        if (order.getId() != null) {
            return template.inTransaction().execute(action ->
                            action.save(order).zipWith(action.insert(event))) Flux<Tuple2<Order, OutboxEvent>>
                    .map(tuple -> tuple.getT1()) Flux<Order>
                    .next();
        }
        return template.inTransaction().execute(action ->
                        action.insert(order).zipWith(action.insert(event))) Flux<Tuple2<Order, OutboxEvent>>
                .map(tuple -> tuple.getT1()) Flux<Order>
                .next();
    } catch (JsonProcessingException e) {
        return Mono.error(e);
    }
}
```

Demo - Connector



```
{
    "name":"order-outbox-source",
    "config":{
        "connector.class":"io.debezium.connector.mongodb.MongoDbConnector",
        "mongodb.name":"mongodb.local",
        "mongodb.hosts":"mongodb://mongo:30001,mongo2:30002,mongo3:30002/order?replicaSet=rs0",
        "transforms":"router",
        "transforms.router.type":"com.example.outbox.Router",
        "transforms.router.topic":"outbox_events",
        "database.whitelist":"order",
        "collection.whitelist":"order[.]outbox_events",
        "tasks.max":"1"
    }
}
```

Demo - Consumer



```
sattraso
@PostConstruct
public void subscribe() {
    receiver.receive()
            .delayUntil(record -> handle(record)
                    .doOnSuccess(o -> record.receiverOffset().acknowledge())
                    .doOnError(e -> log.error("[{}] error occurred => {}", record.key(), e.getMessage()))
            )
            .subscribe();
}

sattraso
@PreDestroy
public void destroy() { log.warn("Shutting down stream ..."); }

1 usage   sattraso
private Mono<Void> handle(ReceiverRecord<String, Event> record) {
    log.info("Received message from topic: {}, key:{}, value:{}", record.topic(), record.key(), record.value());
    String eventType = record.value().getEventType().toString().toUpperCase();
    switch (EventType.valueOf(eventType)) {
        case ORDER_PROCESSING:
            log.info("Handle event_type: {}", eventType);
            String payload = record.value().getPayload().toString();
            return Mono.fromCallable(() -> mapper.readValue(payload, Order.class)) Mono<Order>
                    .flatMap(o -> service.allocateStock(o)) Mono<Void>
                    .then();
        default:
            log.info("Un-handle event_type: {}", eventType);
    }
    return Mono.empty();
}
```

Demo - Github

https://github.com/Tookdee88/spring-boot-outbox-pattern