KAFKA AND CQRS

117

117

Kafka for CQRS

- Free audit log
- Loosely coupled
- Availability – Sender doesn't require receiver
- Immutable log ⇒ less need to encapsulate access to the data
  - Emphasis on sharing the data
  - Data more important than API

118

118

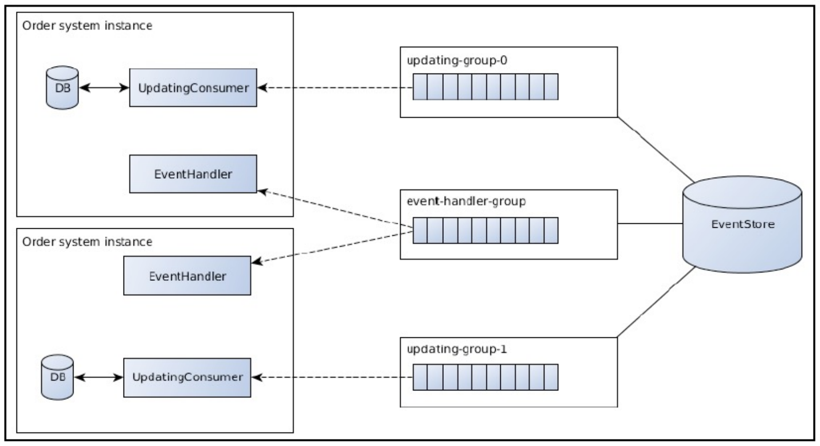
Kafka Message Broker

- Publish-subscribe
  - Messages organized into topics
  - Transactional event producers
  - In-order event consumption
  - Message published to a topic
  - Consumed once per consumer group
- CQRS
  - One consumer group per app (event notifications)
  - One handler group per topic (event handling)

119

119

Kafka Message Broker



120

120

Meal Event

```
public abstract class MealEvent {
   private final Instant instant;
   protected MealEvent() { instant = Instant.now(); }
   protected MealEvent(Instant instant) {
      this.instant = instant;
   }
   ...
}
public class OrderPlaced extends MealEvent {
   private final OrderInfo orderInfo;
   public OrderPlaced(OrderInfo orderInfo) { this.orderInfo = orderInfo; }
   public OrderPlaced(OrderInfo orderInfo, Instant instant) {
      super(instant);
      this.orderInfo = orderInfo;
   }
   ...
}
```

121

121

Event Handler

```
@Singleton
public class OrderEventHandler {

   @Inject
   MealPreparationService mealService;

   public void handle(@Observes OrderPlaced event) {
      mealService.prepareMeal(event.getOrderInfo());
   }
}
```

122

122

Event Processing

```
public class MealPreparationService {
   @Inject
   EventProducer eventProducer;

   @Inject
   IngredientStore ingredientStore;

   public void prepareMeal(OrderInfo orderInfo) {
      // use ingredientStore to check availability
      if (...)
         eventProducer.publish(new
                           OrderFailedInsufficientIngredients());
      else
         eventProducer.publish(new MealPreparationStarted(orderInfo));
   }
}
```

123

Persistent Event State

```
@RequestScoped
public class MealOrders {
   @PersistenceContext EntityManager entityManager;
   public MealOrder get(UUID orderId) {
      return entityManager.find(MealOrder.class, orderId.toString());
   }
   public void apply(@Observes OrderPlaced event) {
      MealOrder order = new MealOrder(event.getOrderInfo());
      entityManager.persist(order);
   }
   private void apply(@Observes OrderStarted event) {
      MealOrder order = entityManager.find(MealOrder.class,
                                           event.getOrderId().toString());
      if (order != null)
         order.start();
   }
}
```

124

Persistent Event State

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;

@ApplicationScoped
public class EventProducer {
   private Producer<String, MealEvent> producer;
   private String topic;
   @Inject Properties kafkaProperties;

   @PostConstruct
   private void init() {
      producer = new KafkaProducer<>(kafkaProperties);
      topic = kafkaProperties.getProperty("topics.order");
      producer.initTransactions();
   }
```

125

125

Persistent Event State

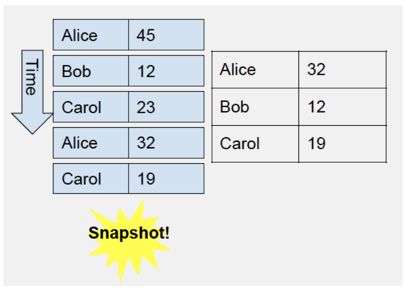
```
   public void publish(MealEvent event) {
      ProducerRecord<String, MealEvent> record = new
                                ProducerRecord<>(topic, event);
      try {
         producer.beginTransaction();
         producer.send(record);
         producer.commitTransaction();
      } catch (ProducerFencedException e) {
         producer.close();
      } catch (KafkaException e) {
         producer.abortTransaction();
      }
   }

   @PreDestroy
   public void close() { producer.close(); }
}
```

126

126

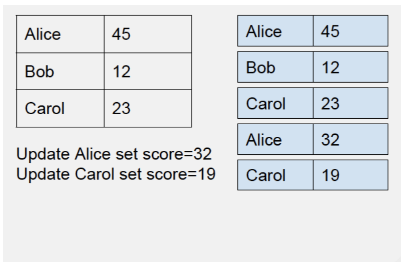
Events and Tables

- Table = stream snapshot
- Stream = table changelog



Alice	45
Bob	12
Carol	23
Alice	32
Carol	19

Alice	32
Bob	12
Carol	19



Alice	45
Bob	12
Carol	23

Update Alice set score=32
Update Carol set score=19

Alice	45
Bob	12
Carol	23
Alice	32
Carol	19

127

127

Kafka Streams

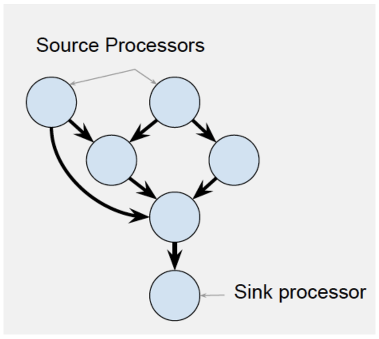
- Higher level API than "topics"
- Operations on streams of records instead of records
  - filter
  - map
  - etc

128

128

Kafka Streams

- Application = directed graph
- Processor = node
- Source processor: stream from topic
- Sink Processor: stream to topic

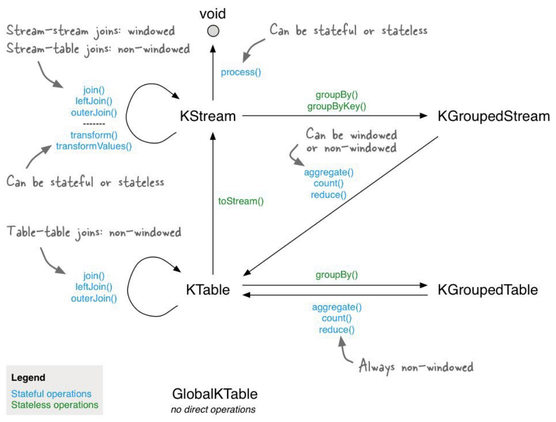


129

129

Streams DSL

- Operations on streams:
- Stateless: Filter, Map, GroupBy etc
- Stateful: Join, Aggregation, etc
  - Table results!
- Compose operations to perform computation



130

130