Apache Kafka

Uma introdução para Desenvolvedores e Arquitetos

Este quem vos fala...

Victor Osório
Senior Software Engineer @ Openet

Mais de 15 anos de experiência com desenvolvimento Java

https://twitter.com/vepo

https://www.linkedin.com/in/victorosorio

https://github.com/vepo

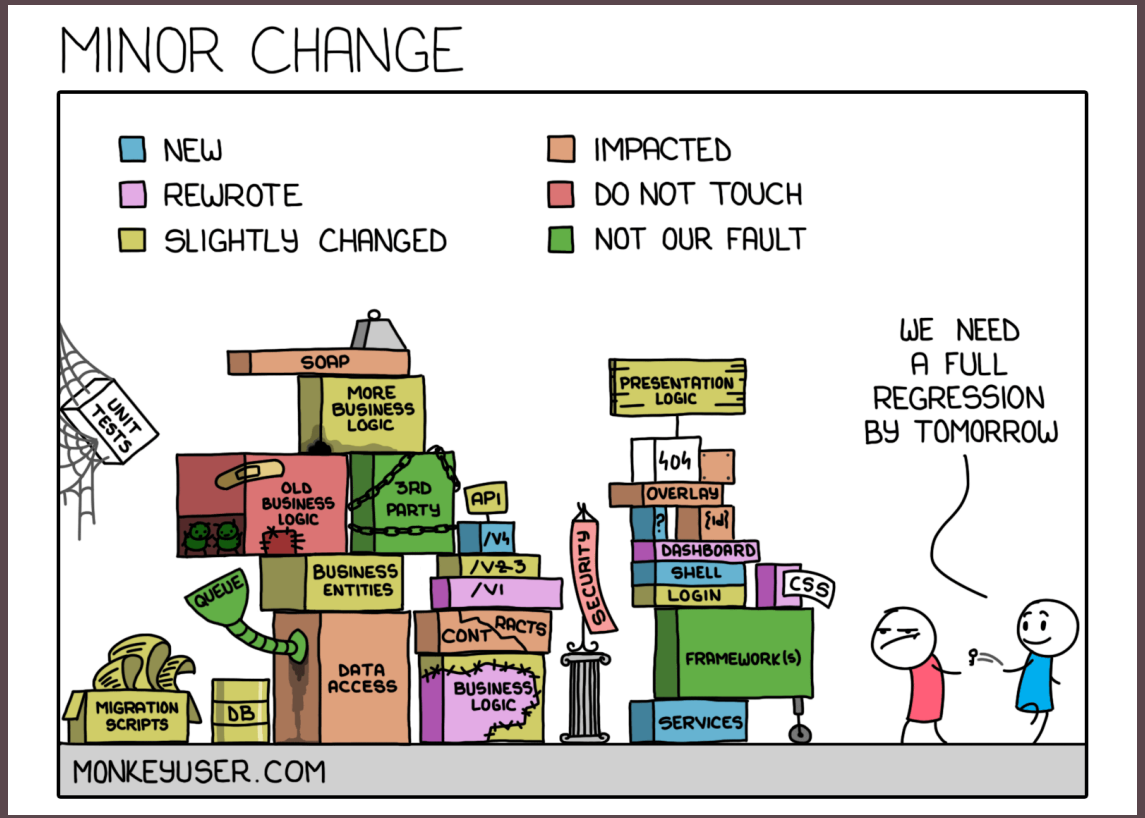
OPENET

Agenda

1. Quais problemas o Kafka Resolve?
2. Como é definida a Arquitetura de um broker Kafka?
3. Como desenhar uma arquitetura usando Kafka?
4. Usando Kafka no Java SE
5. Usando Kafka no Quarkus

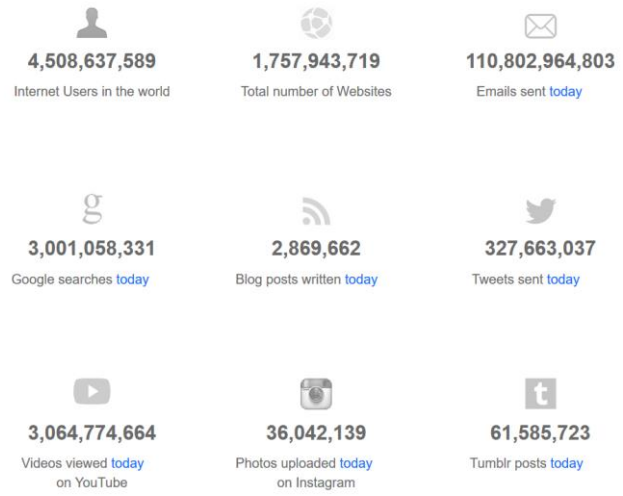
Público Alvo

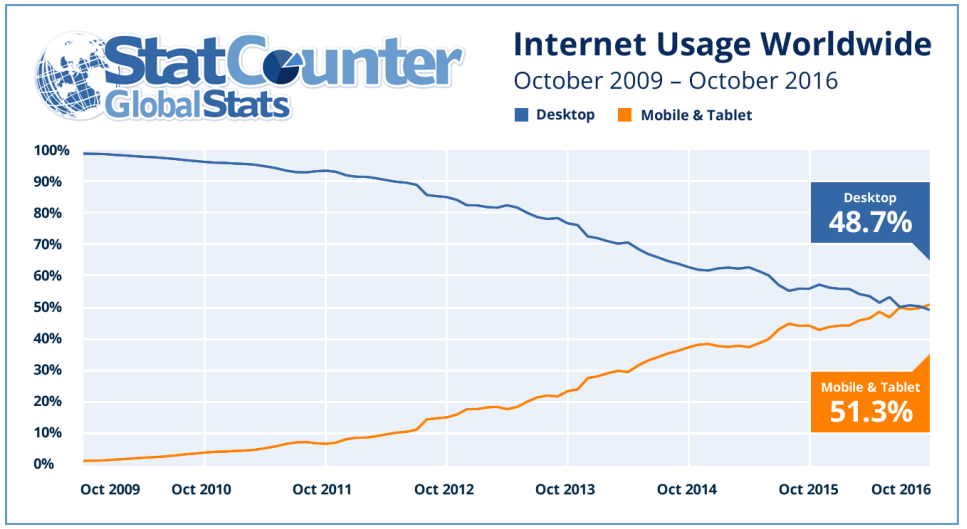
- Desenvolvedor ~~Java~~
- Interesse em Microsserviços
- Interesse em Sistemas Distribuídos
- Interesse em Sistemas Escaláveis

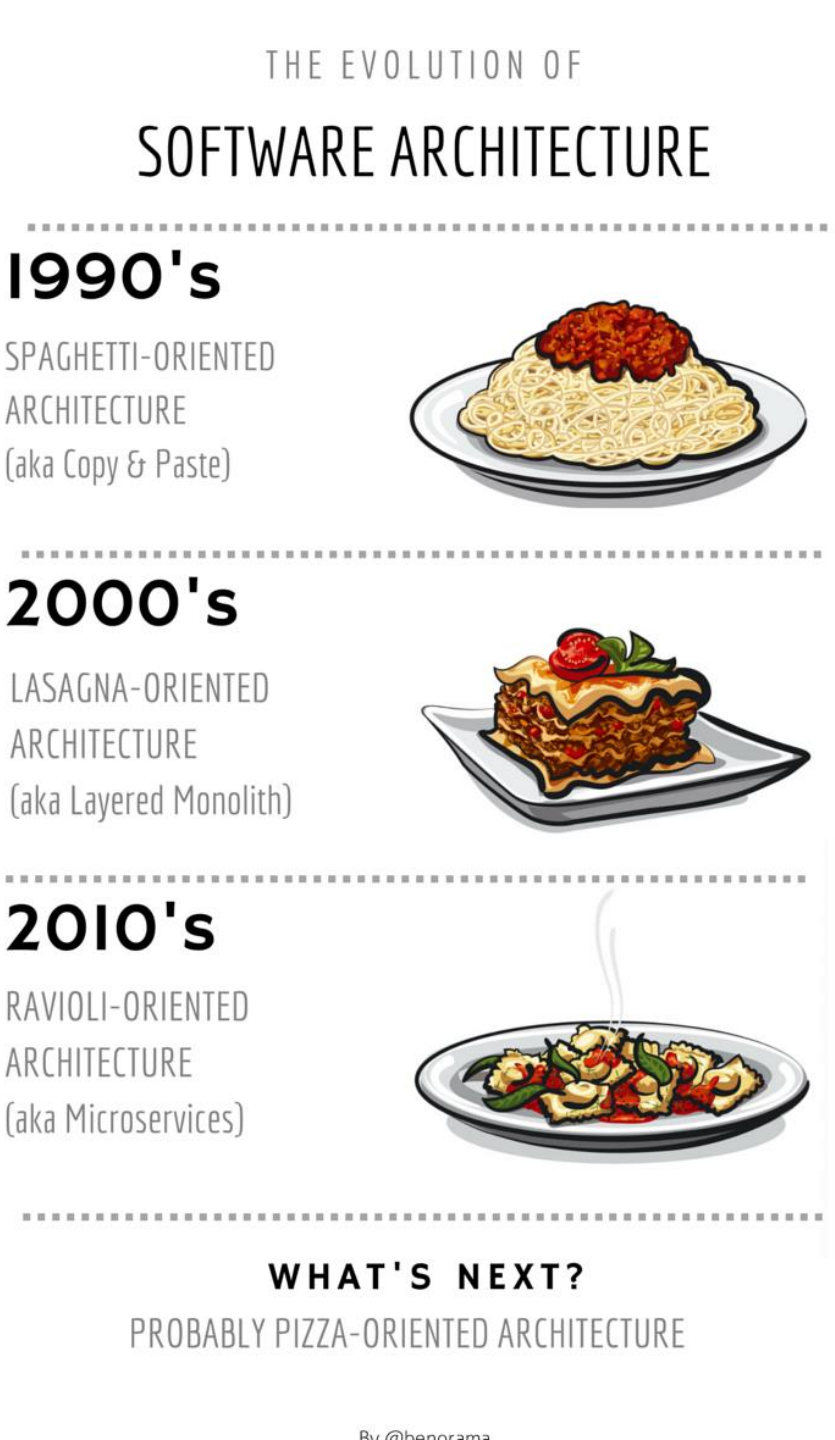


Quais problemas o Kafka Resolve?

2010's – Internet Onipresente





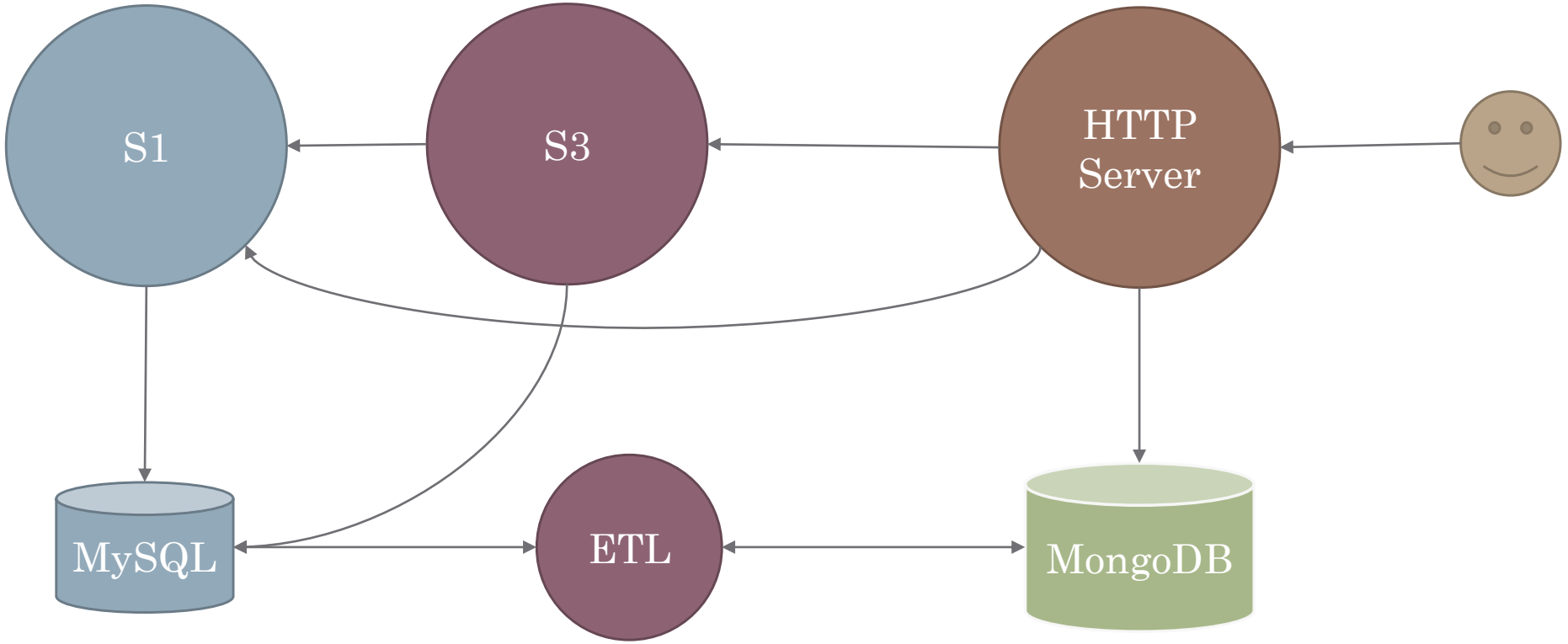


Implicações

- Mais serviços existentes
- Mais acessos simultâneos
- Falência dos modelos de Arquitetura de Software antigos
- Emergência de novos modelos de Arquitetura de Software

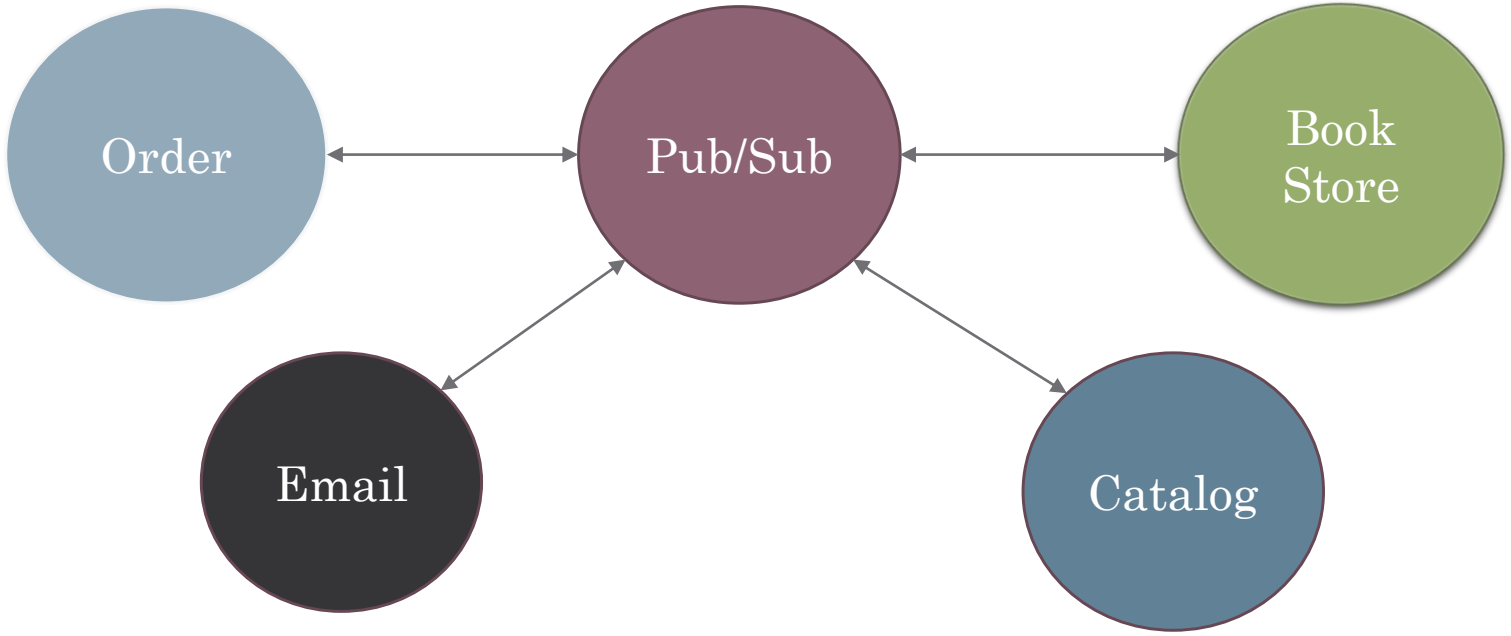
Novos Desafios!

Como integrar Serviços?



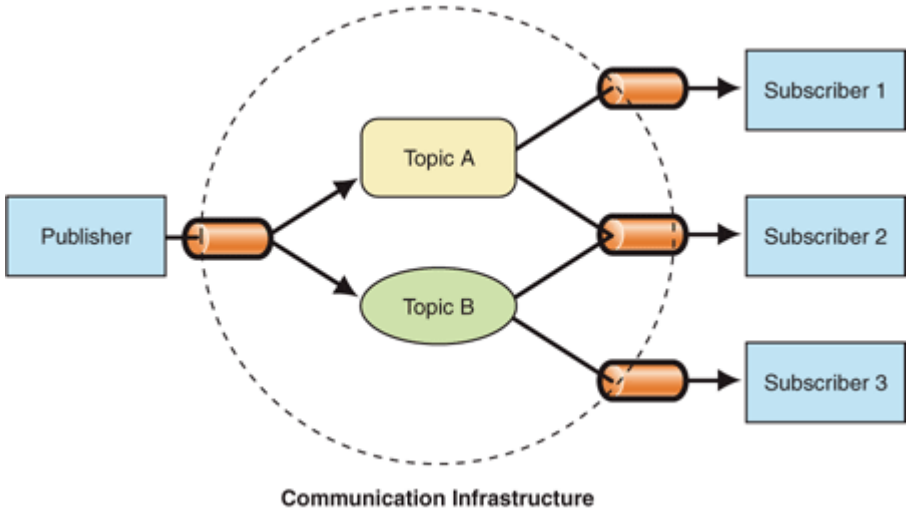
Event Driven Architecture

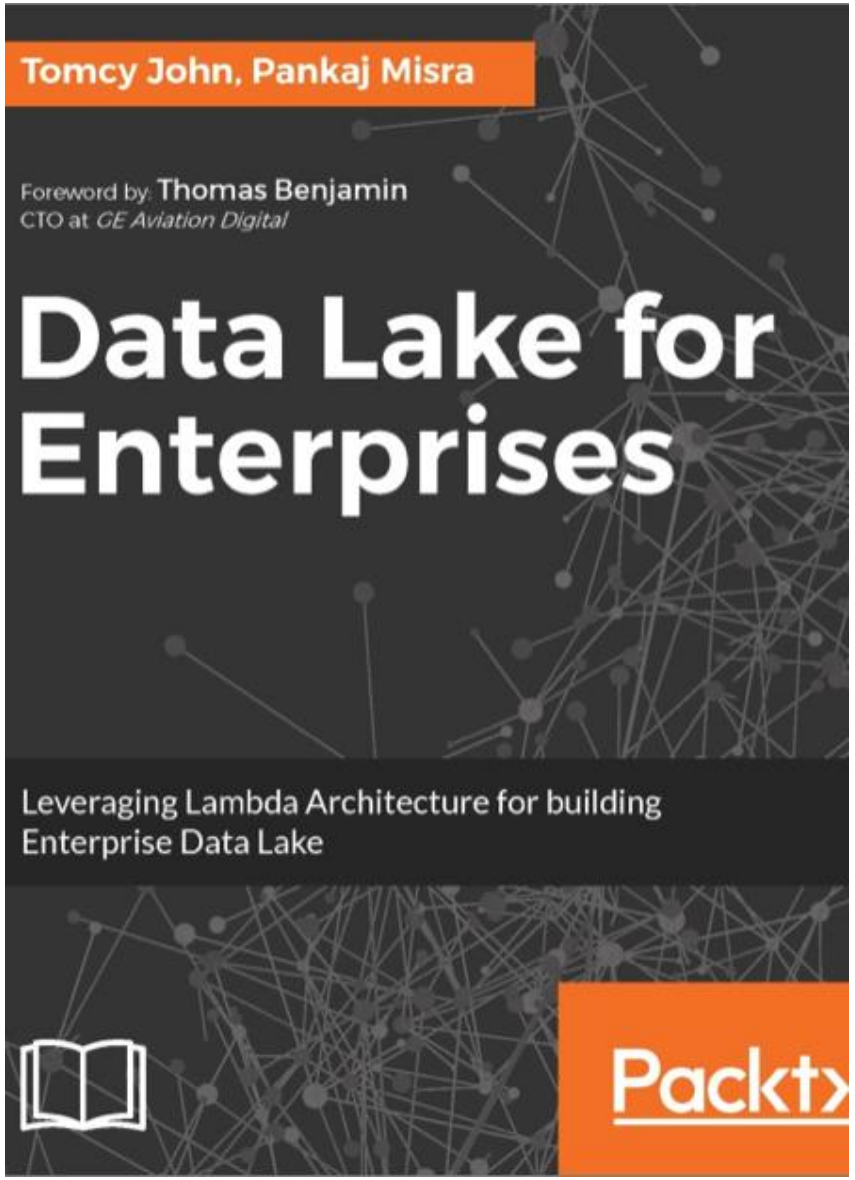
- Cada serviço possui sua Base de Dados *Database per servisse*
- É necessário um mecanismo para garantir a integração dos serviços
- Pub/Sub middlewares



Pub/Sub Alternatives

- Amazon Kinesis.
- RabbitMQ.
- Apache Kafka.
- MuleSoft Anypoint Platform.
- Azure Event Hubs.
- PieSync.
- TIBCO Spotfire.
- IBM MQ.
- Google Cloud Pub/Sub





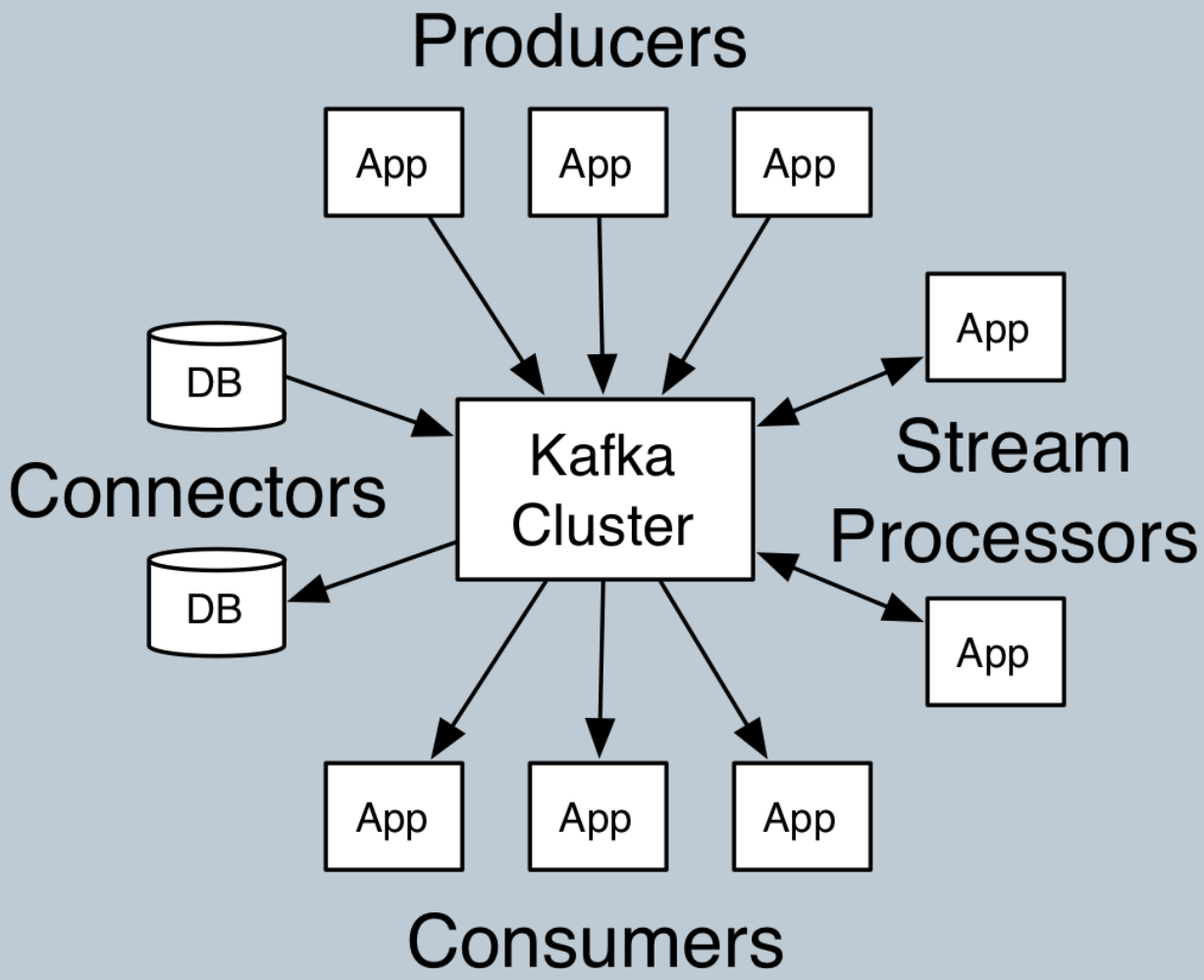
Porque o Apache Kafka?

- Por causa das escolhas arquiteturais é possível entregar:
  - **High-throughput**: Kafka is capable of handling high-velocity and high-volume data using not so large hardware. It is capable of supporting message throughput of thousands of messages per second.
  - **Low latency**: Kafka is able to handle these messages with very low latency of the range of milliseconds, demanded by most of new use cases.
  - **Fault tolerant**: The inherent capability of Kafka to be resistant to node/machine failure within a cluster.
  - **Durability**: The data/messages are persistent on disk, making it durable and messages are also replicated ...

Como é definida a Arquitetura de um broker Kafka?

Definindo os nós

- **Kafka Cluster**
- **Producers**
- **Consumers**
- Stream Processors
- Connectors



APIs Kafka

- Kafka has four core APIs:
  - The Producer API allows an application to publish a stream of records to one or more Kafka topics.
  - The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
  - The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
  - The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

Fonte: https://kafka.apache.org/documentation/#gettingStarted

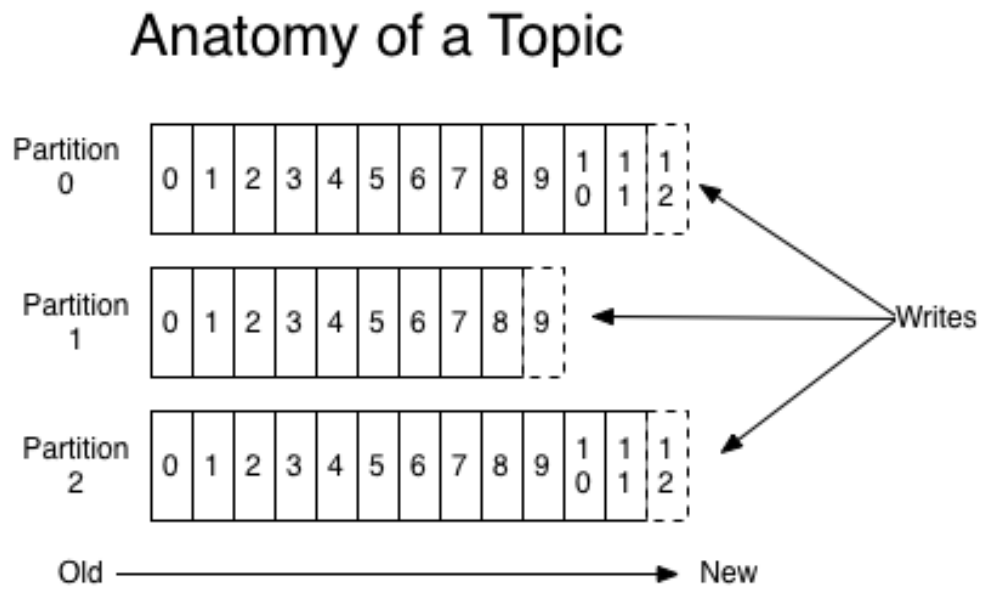
Enviando mensagem

Enviando mensagem

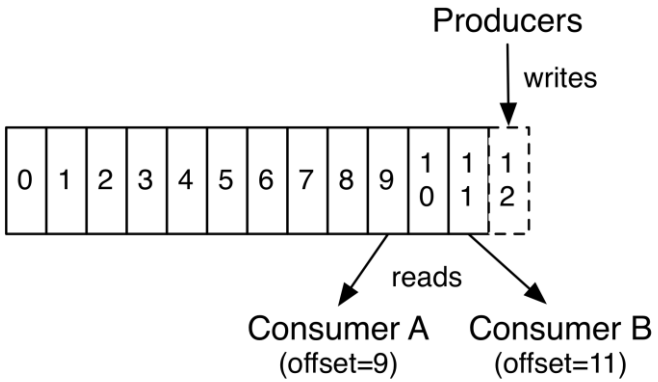
- **Tópico** é uma categoria ou o nome do Feed/Stream
  - Deve ser muito bem definido
- **Chave** é o identificador da Mensagem
  - Pode ser nulo!
  - Deve ser muito bem definido, irá determinar o comportamento do Stream!
- **Mensagem** é um ByteArray enviado pelo Stream
  - Não pode ser nulo!
  - Deve ser definido o método de Serialização e o Formato

Kafka recebendo a mensagem

- Mensagens sempre são escritas no final de um arquivo
- Um mesmo tópico pode ter várias partições
- De acordo com a Chave a mensagem é enviada para uma partição especifica
- Uma mensagem é referenciada por:
  - Tópico
  - Partição
  - Índice



Recebendo mensagem



- Vários Consumers podem apontar para o mesmo Tópico
- Cada tipo de Consumer é escalável
  - Para ser escalável, um Tópico deve ter mais de 1 partição

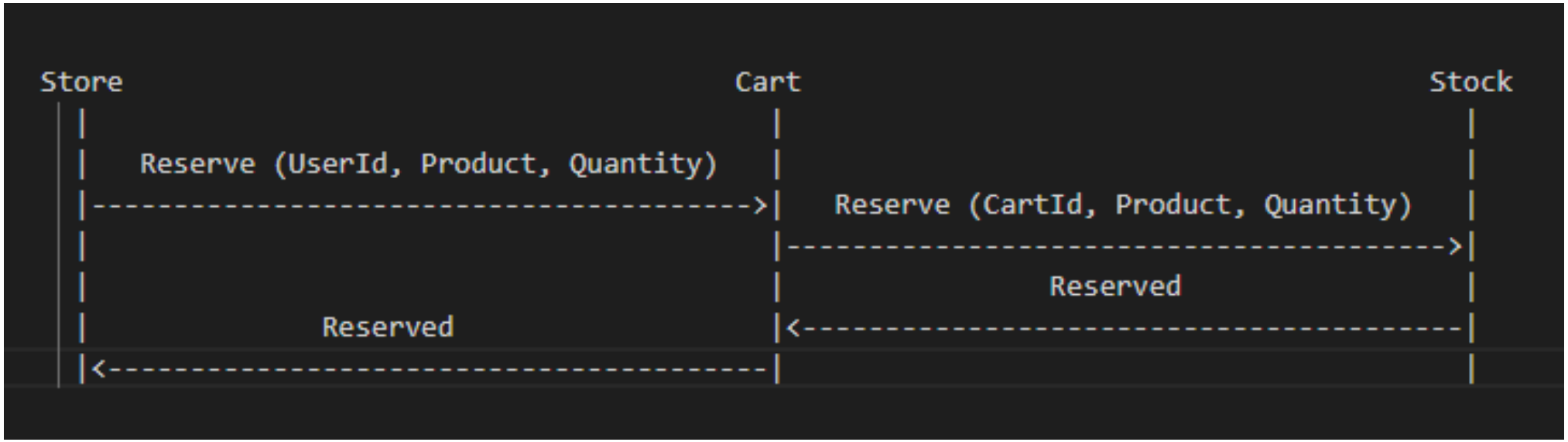
Como desenhar uma arquitetura usando Kafka?

Modelando

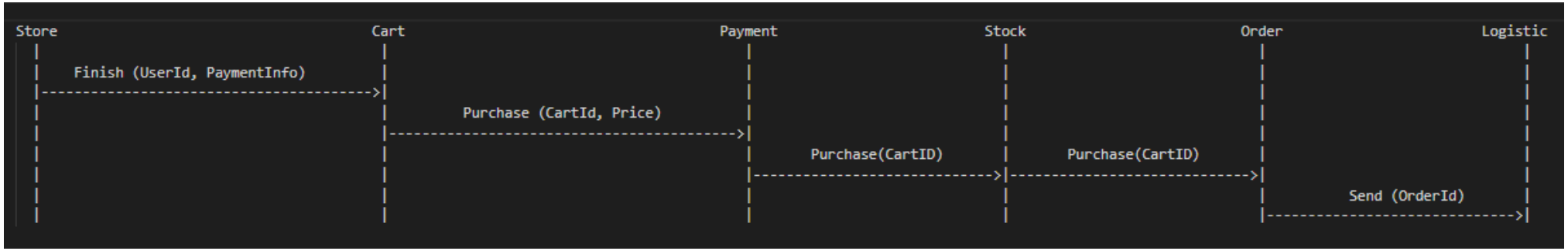
- A modelagem deve focar em
  - Qual a responsabilidade de cada Módulo
  - Quais mensagens cada Módulo irá Emitir
  - Quais mensagens cada Módulo irá Consumir

Requisitos – Alto Nível

- Sistema de *BookStore*
  - Módulos
    - Loja
    - Carrinho
    - Pagamentos
    - Logística
    - Estoque
    - Fiscal
    - Business Inteligence
- A Loja DEVE ter um tempo de resposta máximo em milissegundos por requisição


Store
Cart
Stock
Reserve (UserId, Product, Quantity)
Reserve (CartId, Product, Quantity)
Reserved
Reserved

Eventos


Store
Cart
Payment
Stock
Order
Logistic
Finish (UserId, PaymentInfo)
Purchase (CartId, Price)
Purchase(CartID)
Purchase(CartID)
Send (OrderId)

Eventos

Requisitos

- Cada serviço deve ter
  1. Consumers e Producers
  2. A lógica da Serialização/Envio/Consumo não deve "poluir" o código
  3. Eventos correlatos devem poder ser associados

- Para isso
  - Factory Pattern resolverá (1) e (2)
  - Cada Evento deve ter seu Id, eventos derivados desse devem copiar o seu Id

Usando Kafka no Java SE

Dependências

- Usar Maven
  - **Group Id**: `org.apache.Kafka`
  - **ArtifactId**: `kafka-clientes`

https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients

```
<dependencies>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>2.4.1</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>2.0.0-alpha1</version>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.3.0-alpha5</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.10.3</version>
    </dependency>
</dependencies>
```

Criando Producer

Chave	Descrição
bootstrap.servers	é uma lista separada por vírgula de pares de host e porta que são os endereços dos brokers Kafka em um cluster Kafka "bootstrap" ao qual um cliente Kafka se conecta inicialmente para inicializar.
key.serializer	Classe para Serializar Chave. Deve implementar org.apache.kafka.common.serialization.Serializer
value.serializer	Classe para Serializar Valor. Deve implementar org.apache.kafka.common.serialization.Serializer

```
14 import org.apache.kafka.clients.producer.KafkaProducer;
15 import org.apache.kafka.clients.producer.ProducerConfig;
16 import org.apache.kafka.common.serialization.StringDeserializer;
17 import org.apache.kafka.common.serialization.StringSerializer;
34     public <K, V> KafkaProducer<K, V> createProducer() {
35         // Configure the Consumer
36         var configProperties = new Properties();
37         configProperties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
38         configProperties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
39         configProperties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
40 
41         return new KafkaProducer<>(configProperties);
42     }
```

Criando Consumer

Chave	Descrição
bootstrap.servers	é uma lista separada por vírgula de pares de host e porta que são os endereços dos brokers Kafka em um cluster Kafka "bootstrap" ao qual um cliente Kafka se conecta inicialmente para inicializar.
group.id	Identificador que associa esse consumer a um grupo. Esse valor é extremamente importante para o gerenciamento das mensagens recebidas.
key.deserializer	Classe para Serializar Chave. Deve implementar org.apache.kafka.common.serialization.Deserializer
value.deserializer	Classe para Serializar Valor. Deve implementar org.apache.kafka.common.serialization.Deserializer

```
10
11 import org.apache.kafka.clients.consumer.ConsumerConfig;
12 import org.apache.kafka.clients.consumer.ConsumerRecord;
13 import org.apache.kafka.clients.consumer.KafkaConsumer;
16 import org.apache.kafka.common.serialization.StringDeserializer;
17 import org.apache.kafka.common.serialization.StringSerializer;
18
44     public <K, V> void startConsumer(String groupId, String topicName, Consumer<ConsumerRecord<K, V>>callback){
45         // Configure the Consumer
46
47         var configProperties = new Properties();
48         configProperties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
49         configProperties.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
50         configProperties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
51         configProperties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);
52
53         try (var consumer = new KafkaConsumer<K, V>(configProperties)) {
54             consumer.subscribe(asList(topicName));
55             while (true) {
56                 consumer.poll(Duration.ofSeconds(1)).forEach(callback);
```

QUARKUS

Usando Kafka no Quarkus

Dependências

- Usar Starter: https://code.quarkus.io/
  - Smallrye Reactive Messaging
  - Kafka Clients

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>${quarkus.platform.group-id}</groupId>
            <artifactId>${quarkus.platform.artifact-id}</artifactId>
            <version>${quarkus.platform.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-kafka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-smallrye-reactive-messaging-kafka</artifactId>
    </dependency>
</dependencies>
```

Configuração

- Os valores devem ser preenchidos no application.properties
  - `src\main\resources\application.properties`
- Podem ser configurados por Variáveis de Ambiente
  - https://quarkus.io/guides/config
- Mais informações: https://quarkus.io/guides/kafka

```
# Configure the Kafka sink (we write to it)
mp.messaging.incoming.reserved-product.connector=smallrye-kafka
mp.messaging.incoming.reserved-product.bootstrap.servers=localhost:9092
mp.messaging.incoming.reserved-product.group.id=cart-product-reserved
mp.messaging.incoming.reserved-product.topic=cart.product.reserved
mp.messaging.incoming.reserved-product.key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
mp.messaging.incoming.reserved-product.value.deserializer=io.vepo.bookstore.infra.CartProductReservedDeserializer

mp.messaging.outgoing.reserve-product.connector=smallrye-kafka
mp.messaging.outgoing.reserve-product.bootstrap.servers=localhost:9092
mp.messaging.outgoing.reserve-product.topic=cart.product.reserve
mp.messaging.outgoing.reserve-product.key.serializer=org.apache.kafka.common.serialization.StringSerializer
mp.messaging.outgoing.reserve-product.value.serializer=io.quarkus.kafka.client.serialization.JsonbSerializer
```

Configurando Producer

- Deve ser configurado
  - Em um CDI Bean
  - Usar *Annotations*
    - `@Inject`
    - `@Channel` → Vai associar a configuração correspondente

```
28 @Path("/cart")
29 @Produces(MediaType.APPLICATION_JSON)
30 @Consumes(MediaType.APPLICATION_JSON)
31 public class CartController {
41     @Inject
42     @Channel("reserve-product")
43     Emitter<CartReserveProduct> addCartEmitter;
```

Consumer

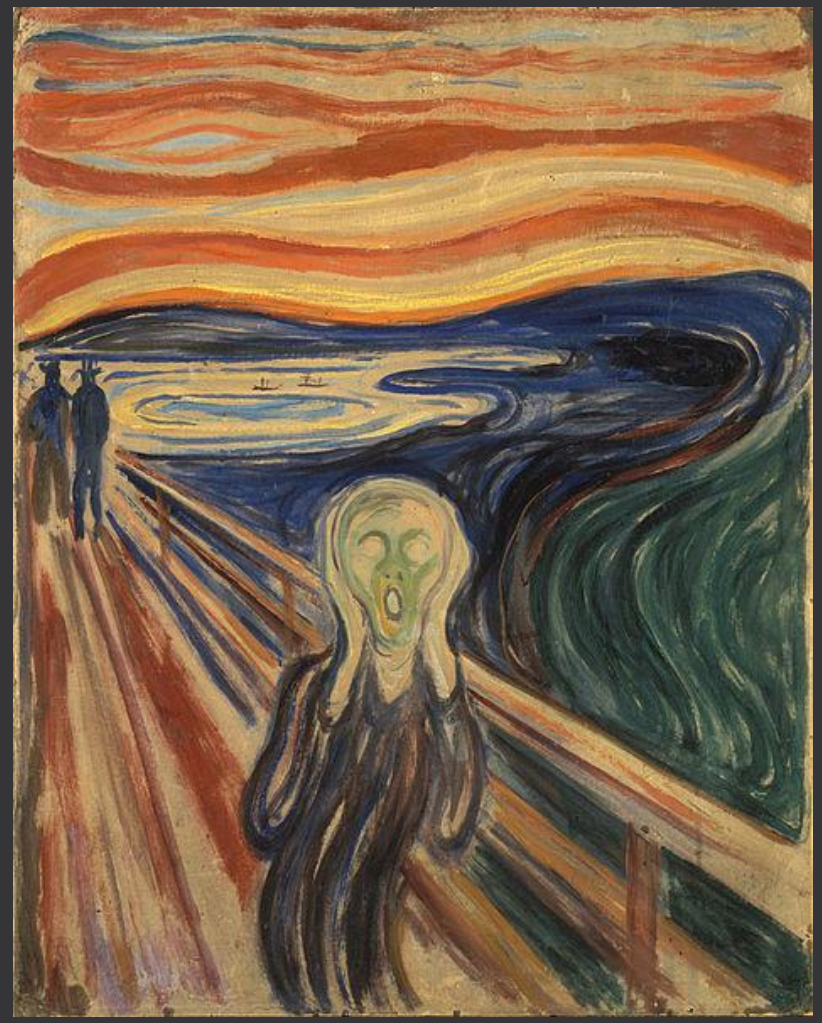
- Deve ser configurado
  - Em um CDI Bean
  - Usar *Annotations*
    - `@Incoming`

```
 9 @ApplicationScoped
10 public class CartProcessor {
11     private final Logger logger = LoggerFactory.getLogger(CartProcessor.class);
12
13     @Incoming("reserved-product")
14     public void process(CartProductReserved message) {
15         logger.info("Receiving final event!\n\nevent={}\n\n", message);
16     }
17 }
```

Demonstração!!!

Duvidas?

Não existe pergunta boba!



Obrigado!!!

- Códigos
  - https://github.com/vepo/kafka-intro