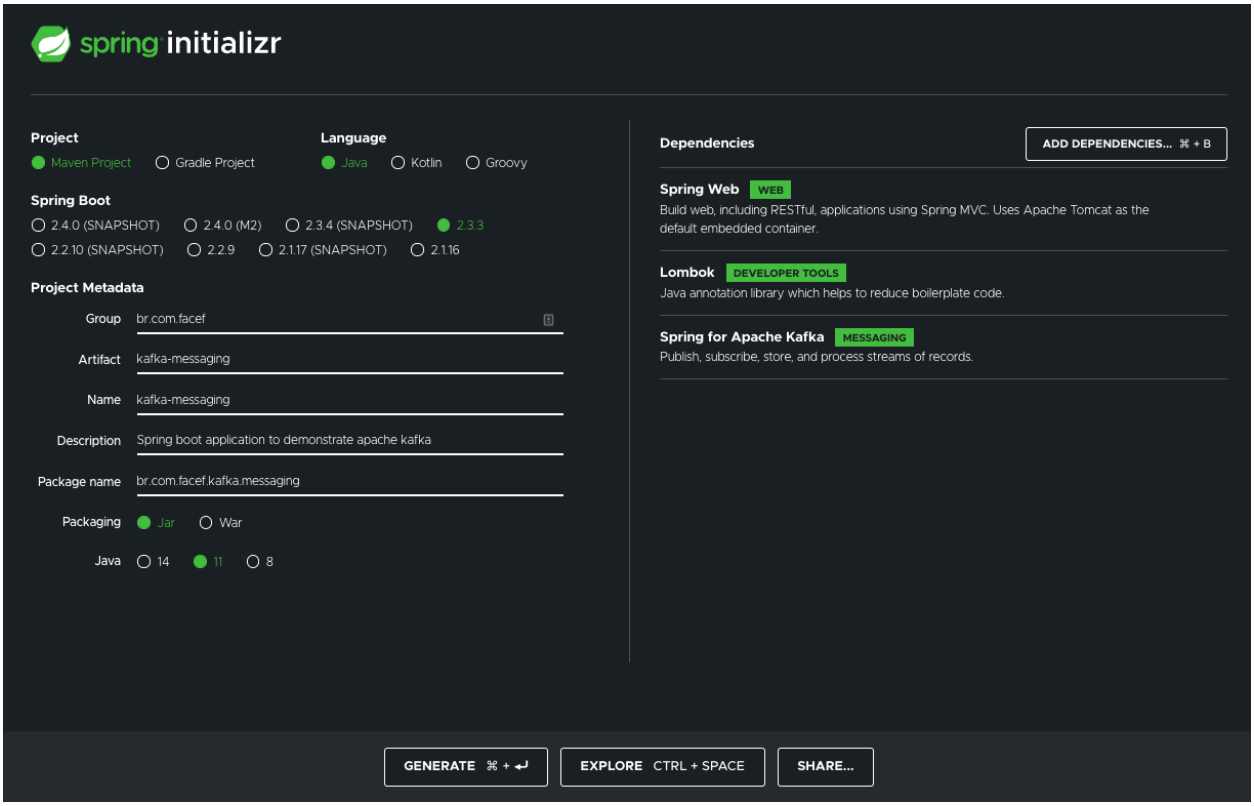
Mensageria com Kafka

Exemplo

Para gerar o projeto podem acessar a **URL**, nessa url iremos utilizar o **Spring Initializr** para gerar a estrutura padrão do projeto já adicionando a biblioteca do Apache Kafka.



Primeiramente precisamos iniciar o Apache Kafka:

- Iniciando via docker-compose

```
docker-compose stop && docker-compose rm -f && docker-compose up -d
```

Para acompanhar a inicialização do container pode-se utilizar o comando:

```
docker-compose logs -f
```

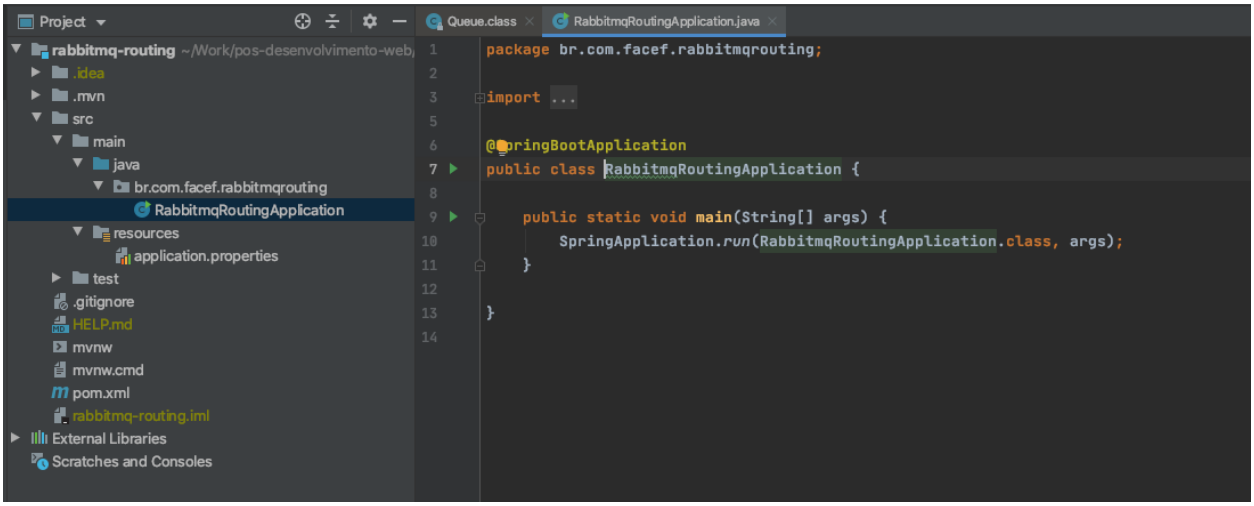
Verificando os logs:

```
kafka_1      | [2020-09-11 01:25:09,984] INFO [Controller id=1] Processing automatic preferred replica leader election (kafka.controller.KafkaController)
kafka_1      | [2020-09-11 01:25:09,986] TRACE [Controller id=1] Checking need to trigger auto leader balancing (kafka.controller.KafkaController)
kafka_1      | [2020-09-11 01:25:09,996] DEBUG [Controller id=1] Topics not in preferred replica for broker 1 Map() (kafka.controller.KafkaController)
kafka_1      | [2020-09-11 01:25:10,002] TRACE [Controller id=1] Leader imbalance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)
kafka_1      | [2020-09-11 01:25:10,274] INFO Successfully submitted metrics to Confluent via secure endpoint (io.confluent.support.metrics.submitters.Conflue
ntSubmitter)
kafka_1      | [2020-09-11 01:30:09,852] INFO [Controller id=1] Processing automatic preferred replica leader election (kafka.controller.KafkaController)
kafka_1      | [2020-09-11 01:30:09,853] TRACE [Controller id=1] Checking need to trigger auto leader balancing (kafka.controller.KafkaController)
kafka_1      | [2020-09-11 01:30:09,854] DEBUG [Controller id=1] Topics not in preferred replica for broker 1 Map() (kafka.controller.KafkaController)
kafka_1      | [2020-09-11 01:30:09,854] TRACE [Controller id=1] Leader imbalance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)
```

Para demonstrar o funcionamento, irei fazer um passo a passo com um exemplo funcional, implementando uma API Rest que recebe uma mensagem via POST com um body que será enviado para a exchange criada, e com base no dado enviado no body a mensagem será roteada para a queue em específico.

Passo a passo:

- **git checkout 4f9bdd3 - Initial project**



- **git checkout 2f607ed - Include docker-compose**

```
version: '3'
services:
# https://hub.docker.com/r/confluentinc/cp-zookeeper
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  # https://hub.docker.com/r/confluentinc/cp-kafka
  kafka:
    # "`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-
    # An important note about accessing Kafka from clients on other machines:
    # -----------------------------------------------------------------------
    #
    # The config used here exposes port 9092 for _external_ connections to the broker
    # i.e. those from _outside_ the docker network. This could be from the host machine
```

```
    # running docker, or maybe further afield if you've got a more complicated setup.
    # If the latter is true, you will need to change the value 'localhost' in
    # KAFKA_ADVERTISED_LISTENERS to one that is resolvable to the docker host from those
    # remote clients
    #
    # For connections _internal_ to the docker network, such as from other services
    # and components, use kafka:29092.
    #
    # See https://rmoff.net/2018/08/02/kafka-listeners-explained/ for details
    # "`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-'"`-._,-
    #
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - 9092:9092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

- **git checkout 00349f6 - Include producer class**

```
package br.com.facef.kafka.messaging.producer;

import java.util.UUID;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
public class OrderProducer {

  @Value("${order.topic}")
  private String orderTopic;

  @Autowired private KafkaTemplate kafkaTemplate;

  public void sendToTopic(String order) {
    final var key = UUID.randomUUID().toString();
    this.kafkaTemplate.send(this.orderTopic, key, order);
  }
}
```

- **git checkout 74efa13 - Include controller class**

```
package br.com.facef.kafka.messaging.controller;

import br.com.facef.kafka.messaging.producer.OrderProducer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/v1/orders")
public class OrderController {

  @Autowired private OrderProducer producer;

  @PostMapping
  public ResponseEntity placeOrder(@RequestBody String order) {
    producer.sendToTopic(order);
    return ResponseEntity.accepted().build();
  }
}
```

- **git checkout 2bbf4fd - Configuring kafka and topic on application.properties**

```
spring.kafka.producer.bootstrap-servers=localhost:9092

order.topic=order-topic
```

- **Iniciando a aplicação**

```
./mvnw spring-boot:run
```

- **Acessando o bash do container do Kafka**

```
docker-compose exec kafka bash
```

- **Realizando uma chamada via API para incluir um novo pedido**

```
curl --location --request POST 'http://localhost:8080/v1/orders' \
--header 'Content-Type: text/plain' \
--data-raw 'ORDER_123456'
```

- **Listando o tópico criado no Kafka - order-topic**

```
./usr/bin/kafka-topics --zookeeper zookeeper:2181 --list
```

```
root@aca4f2e084be:/# ./usr/bin/kafka-topics --zookeeper zookeeper:2181 --list
__confluent.support.metrics
order-topic
root@aca4f2e084be:/#
```

- **Verificando a gravação das mensagens no tópico**

```
./usr/bin/kafka-topics --zookeeper zookeeper:2181 --topic order-topic --describe
```

```
root@aca4f2e084be:/# ./usr/bin/kafka-topics --zookeeper zookeeper:2181 --topic order-topic --describe
Topic: order-topic      PartitionCount: 1       ReplicationFactor: 1    Configs:
        Topic: order-topic      Partition: 0    Leader: 1       Replicas: 1     Isr: 1
root@aca4f2e084be:/#
```

- **git checkout 344c881 - Include a consumer**

```
package br.com.facef.kafka.messaging.consumer;

import lombok.extern.slf4j.Slf4j;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
@Slf4j
public class OrderConsumer {

  @KafkaListener(topics = "${order.topic}", groupId = "${spring.kafka.consumer.group-id}")
  public void consume(String order) {
    log.info("Order: " + order);
  }
}
```

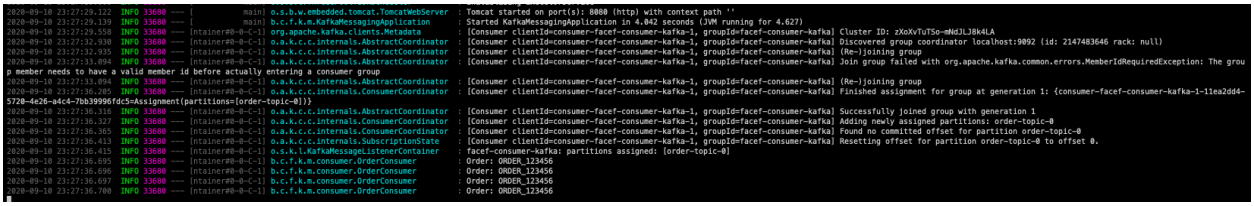
- **git checkout bccac86 - Add consumer configuration on application.properties**

```
spring.kafka.producer.bootstrap-servers=localhost:9092

order.topic=order-topic

spring.kafka.consumer.group-id=facef-consumer-kafka
spring.kafka.consumer.auto-offset-reset=earliest
```

- **group-id**: É o identificador do grupo de consumo do tópico, responsável pelas configurações de consumo em paralelo do tópico, mais sobre grupo de consumos pode ser visto na documentação.
- **auto-offset-reset**: É a configuração da posição inicial que será consumida do tópico, no caso foi configurado como **earliest**, então será do início do tópico.
- **Iniciando a aplicação novamente**

```
./mvnw spring-boot:run
```

As mensagens foram consumidas



Enviando e consumindo mensagens tipadas

Em alguns momentos temos a necessidade de trafegar dados que não serão somente strings e sim objetos complexos com muito mais informações, para isso o Kafka fornece a possibilidade de envio de mensagens tipadas que serão **serializadas** pelo producer e **deserializadas** pelo consumer.

- **git checkout 0c4edeb - Include configuration to serialize/deserialize Order**

```
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.consumer.properties.spring.json.add.type.headers=false
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages=br.com.facef.kafka.messaging.dto
```

Vamos adicionar três propriedades:

- **key-serializer**: É o tipo de serialização da chave da mensagem, no caso vamos manter como String;
- **value-serializer**: É o tipo de serialização do conteúdo da mensagem, que vamos alterar para um formato Json;
- **add.type.headers**: Como vamos enviar a mensagem como Json e não sabemos qual o tipo da mensagem, desabilitamos a adição tipo no header da mensagem.
- **key-deserializer**: É o tipo de deserialização da chave da mensagem, no caso vamos manter como String;
- **value-deserializer**: É o tipo de deserialização do conteúdo da mensagem, que vamos alterar para um formato Json;

Iremos adicionar a dependência do Jackson para o processo de serialização / deserialização

- **git checkout ba3abc0 - Include jackson-databind as dependency**

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.11.2</version>
</dependency>
```

- **git checkout 708668a - Create a dto to transfer order**

```
package br.com.facef.kafka.messaging.dto;

import java.io.Serializable;
import java.math.BigDecimal;
import lombok.Data;

@Data
public class Order implements Serializable {

  private String orderId;
  private String paymentType;
  private BigDecimal value;
}
```

- **git checkout def08e6 - Change a controller to receive a Order**

```
package br.com.facef.kafka.messaging.controller;

import br.com.facef.kafka.messaging.dto.Order;
import br.com.facef.kafka.messaging.producer.OrderProducer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/v1/orders")
public class OrderController {

  @Autowired private OrderProducer producer;

  @PostMapping
  public ResponseEntity placeOrder(@RequestBody Order order) {
    producer.sendToTopic(order);
    return ResponseEntity.accepted().build();
  }
}
```

- **git checkout b392e81 - Change Producer to send to Kafka a Order Object**

```
package br.com.facef.kafka.messaging.producer;
```

```
import br.com.facef.kafka.messaging.dto.Order;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
public class OrderProducer {

  @Value("${order.topic}")
  private String orderTopic;

  @Autowired private KafkaTemplate kafkaTemplate;

  public void sendToTopic(Order order) {
    this.kafkaTemplate.send(this.orderTopic, order.getOrderId(), order);
  }
}
```

- **git checkout 4a98675 - (HEAD → master) Change a consumer to desserialize a order**

```
package br.com.facef.kafka.messaging.consumer;

import br.com.facef.kafka.messaging.dto.Order;
import lombok.extern.slf4j.Slf4j;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Component
@Slf4j
public class OrderConsumer {

  @KafkaListener(topics = "${order.topic}", groupId = "${spring.kafka.consumer.group-id}")
  public void consumer(ConsumerRecord<String, Order> consumerRecord) {
    log.info("key: " + consumerRecord.key());
    log.info("Headers: " + consumerRecord.headers());
    log.info("Partion: " + consumerRecord.partition());
    log.info("Order: " + consumerRecord.value());
  }
}
```

Também foi realizado a alteração para receber como parâmetro do método o **ConsumerRecord** pois através dele é possível recuperar várias outras informações através do tópico.

- **Apagando o tópico existente**

```
./usr/bin/kafka-topics --zookeeper zookeeper:2181 --delete --topic order-topic
```

- **Iniciando a aplicação novamente**

```
./mvnw spring-boot:run
```

- **Realizando uma chamada via API para incluir um novo pedido**

```
curl --location --request POST 'http://localhost:8080/v1/orders' \
--header 'Content-Type: application/json' \
--data-raw '{
    "orderId": "839128391283",
    "paymentType": "creditCard",
    "value": 1249.99
}'
```

- **Visualizando o consumo através dos logs do console**