Introdução à mensageria utilizando Spring Boot com Apache Kafka e RabbitMQ

Renato Hioji Okamoto Odake e Ian Paschoal Oliveira Belato de Freitas

AGENDA

43 slides 40-50 minutos

1. INTRODUÇÃO À MENSAGERIA
2. ESTRUTURA DA MENSAGERIA
3. RABBITMQ
4. APACHE KAFKA
5. ENCERRAMENTO

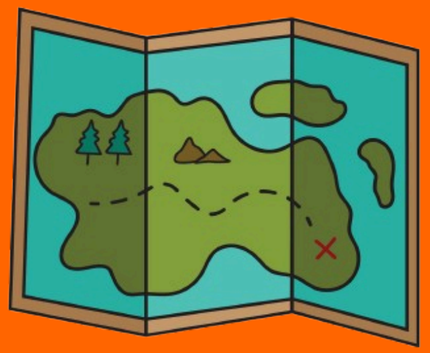
O QUE É A MENSAGERIA?

- Sistema para comunicação entre aplicativos, serviços e outros sistemas, mesmo que em linguagens diferentes;
- Sistema “ás cegas”.
- Tornar os serviços cada vez mais independentes um dos outros (Decoupling);
- Possibilidade de validações dos conteúdos de uma mensagem;

A ESTRUTURA DA MENSAGERIA

Producer
(Aquele que envia)

Consumer
(Aquele que recebe)



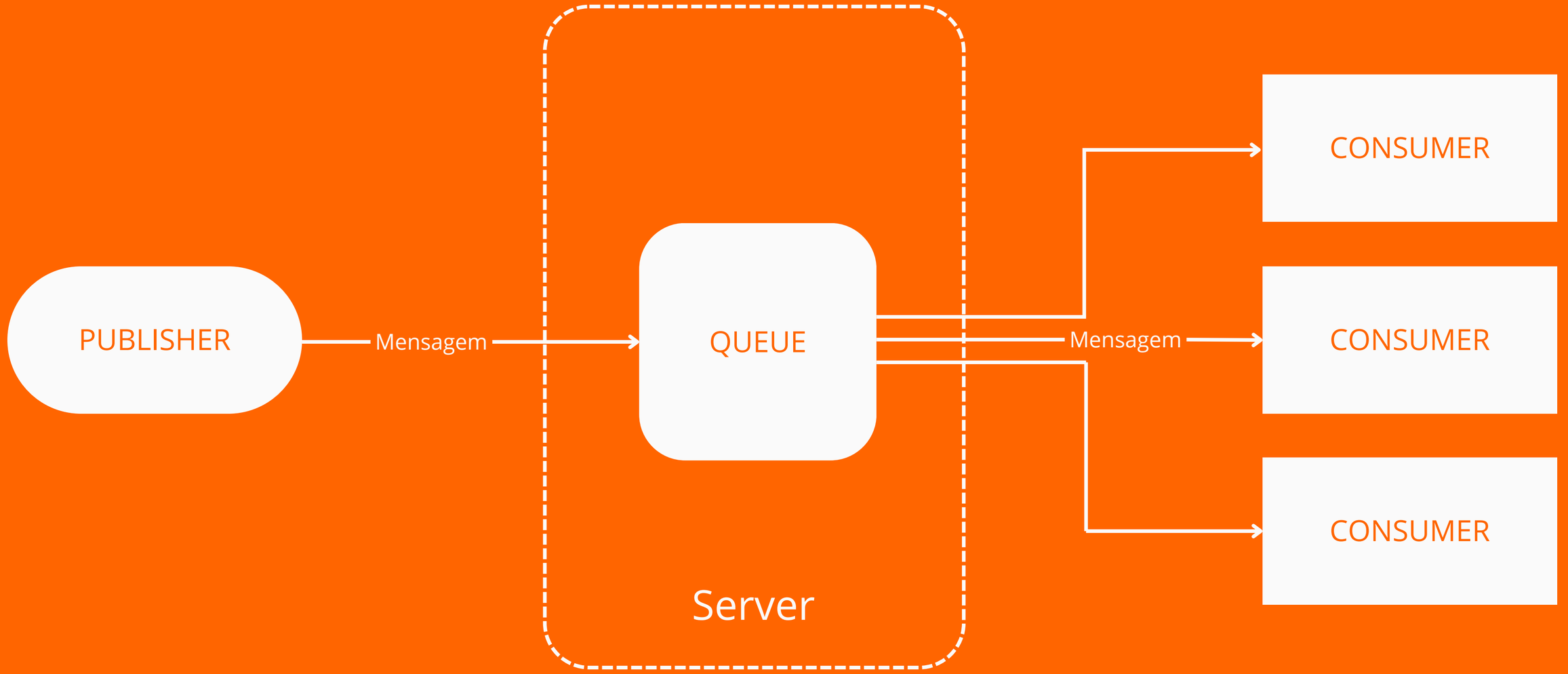
Queue
(O caminho)

Exchange
(A direção)

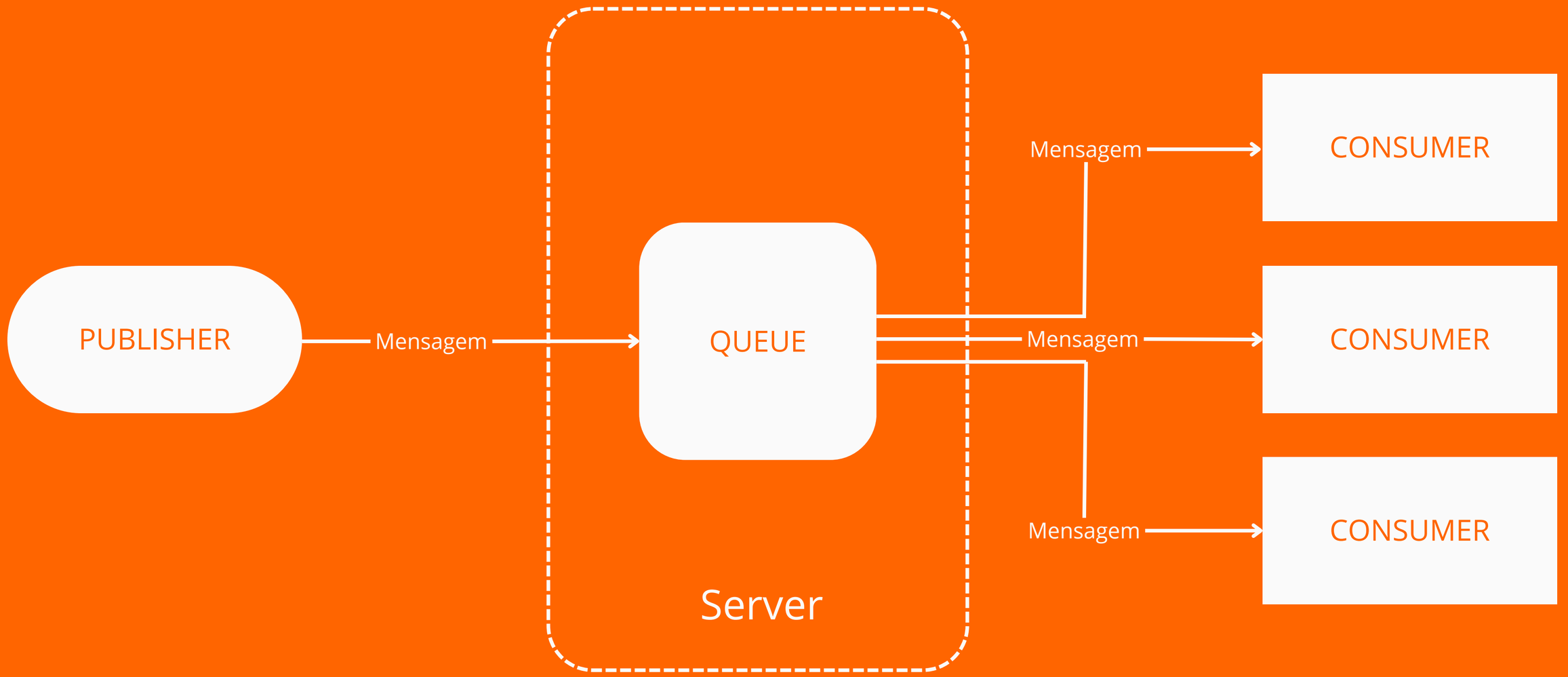
Message Broker
(O intermediador)

DLQ
(DEU RUIM!!!!!!)

FILA POINT-TO-POINT



FILA PUB/SUB


PUBLISHER
Mensagem
QUEUE
Server
Mensagem
Mensagem
Mensagem
CONSUMER
CONSUMER
CONSUMER


RabbitMQ

REQUISITOS MÍNIMOS

- Erlang 26.2.x
- RabbitMQ Server 3.13.x
- Java 17+
- Spring Boot 3.3.4
- Spring for RabbitMQ
- Lombok (Opcional)

ADVANCED MESSAGE QUEUING PROTOCOL (AMQP)

Padronização de como é realizada a comunicação entre os sistemas e dispositivos ligada à Internet das Coisas;

Protocolo de Rede;

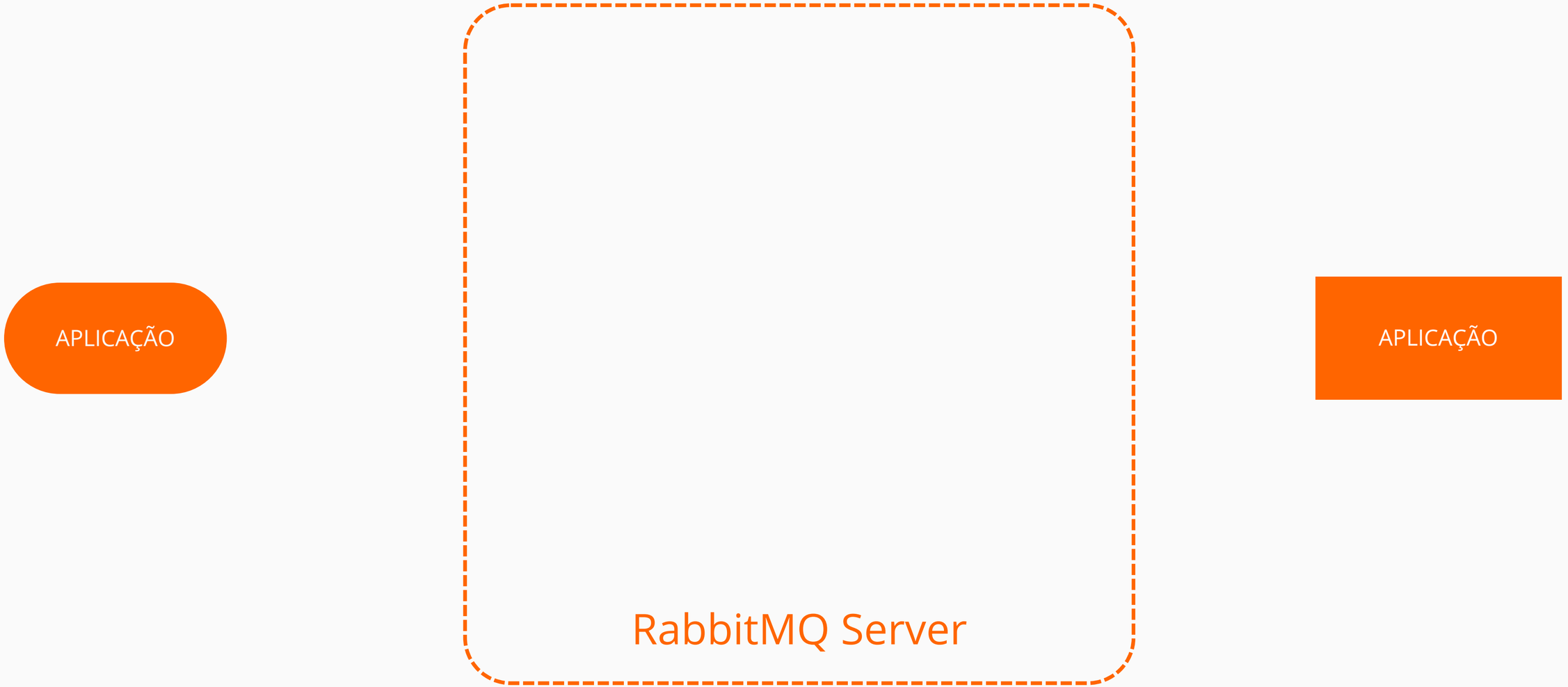
Topic Exchange;

Utilização de Bindings;

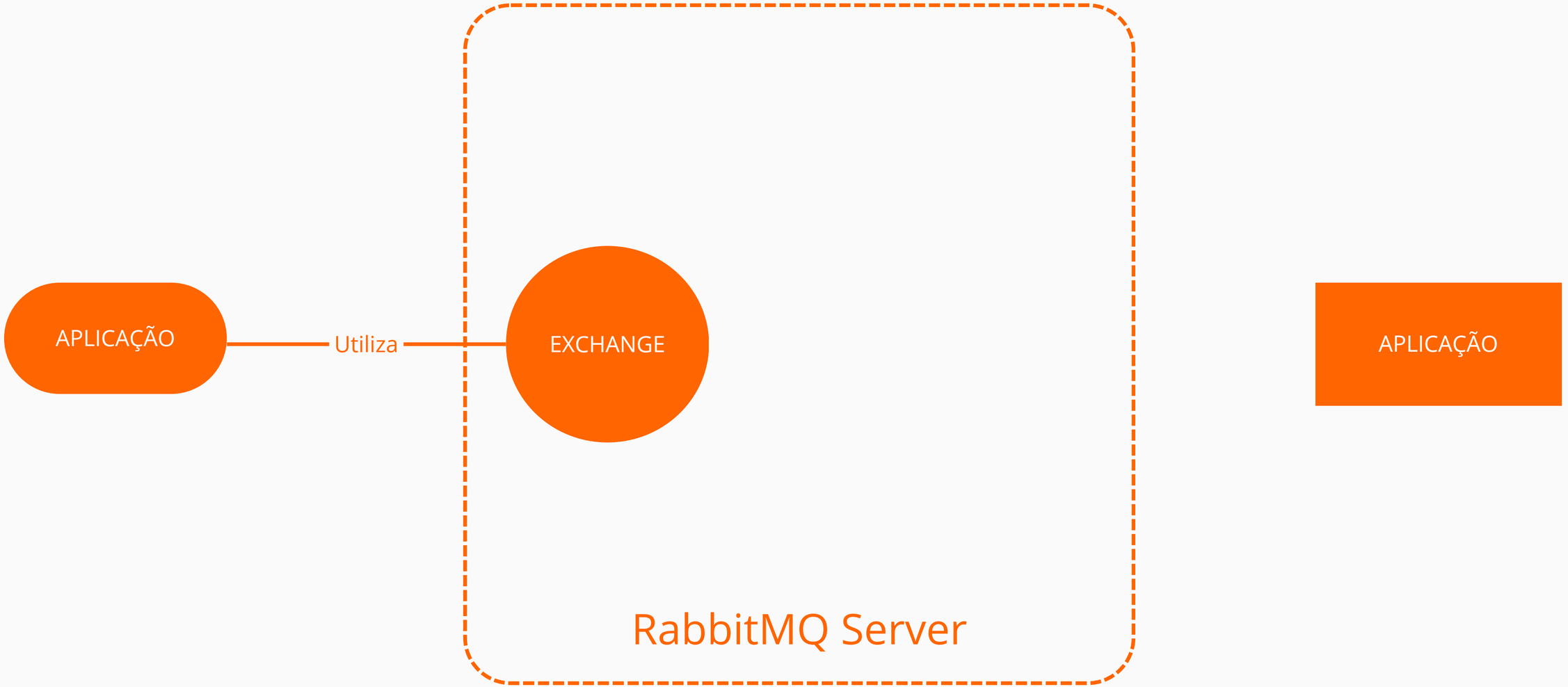
Parking Lot.

```
import org.springframework.amqp.core.Binding
mport org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.listener.SimpleMessageListenerContainer;
import org.springframework.amqp.rabbit.listener.adapter.MessageListenerAdapter;
```

Construindo um serviço com mensageria



Exchange

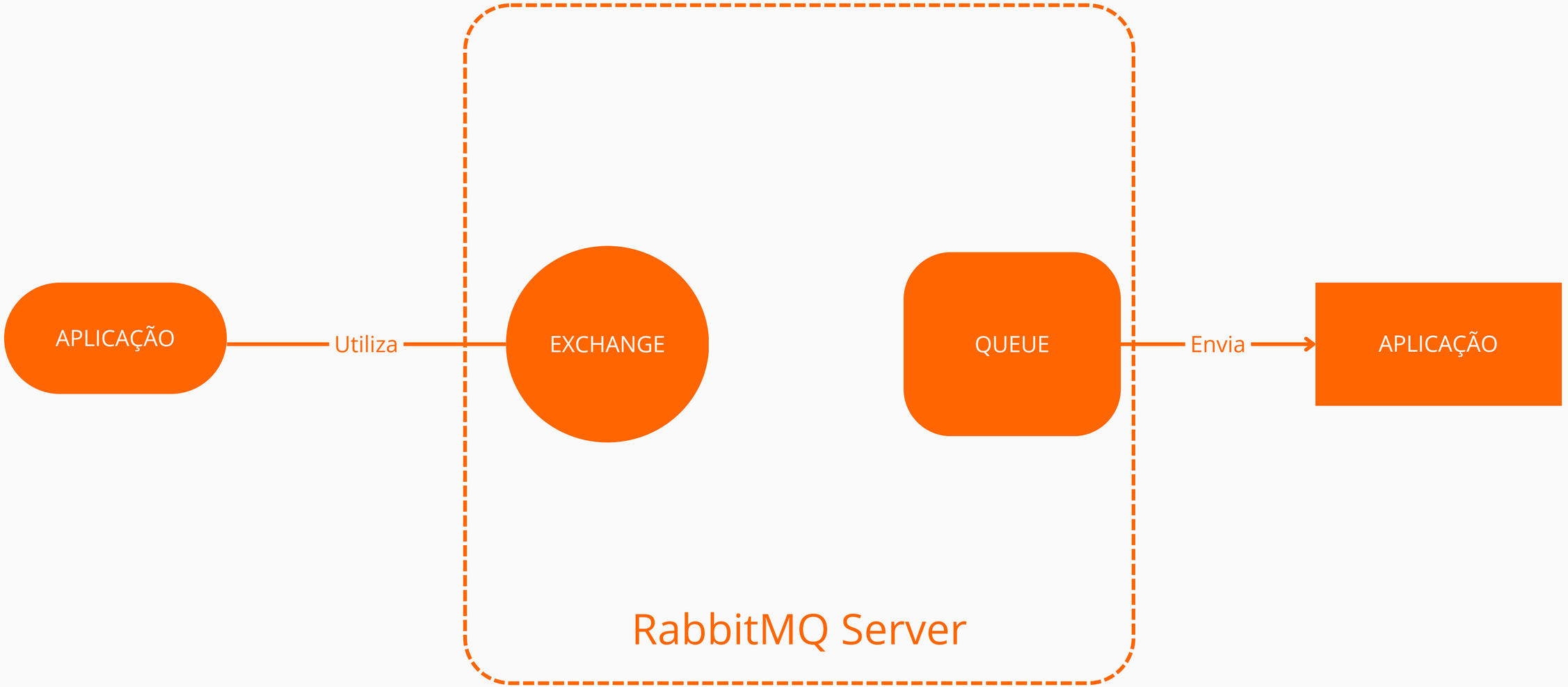


Criando um Topic Exchange

```
public static final String topicExchangeName = "spring-boot-exchange";

@Bean
TopicExchange exchange() {
   return new TopicExchange(topicExchangeName);
}
```

Queue

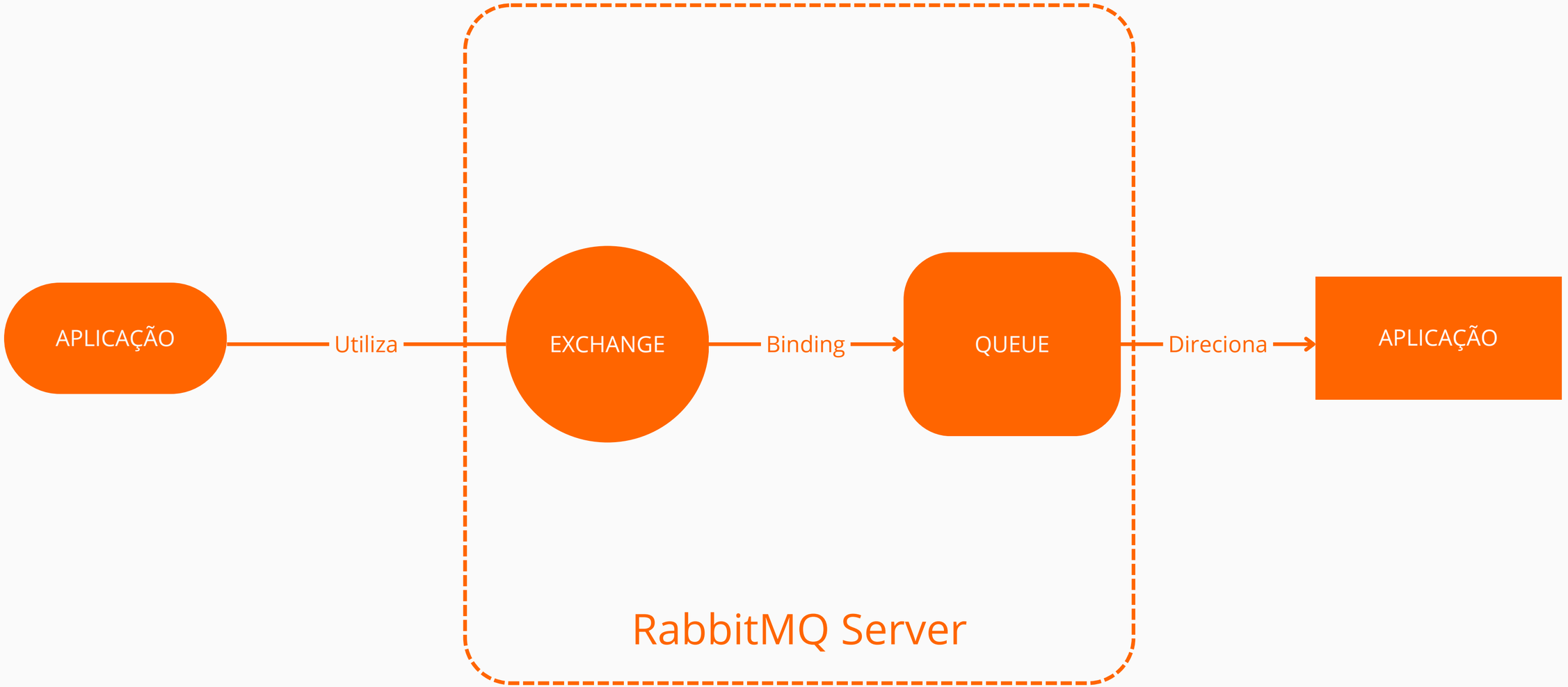

APLICAÇÃO
Utiliza
EXCHANGE
QUEUE
Envia
APLICAÇÃO
RabbitMQ Server

Criando minha fila

```
static final String queueName = "spring-boot";

@Bean
Queue queue() {
   return new Queue(queueName, true);
}
```

Binding

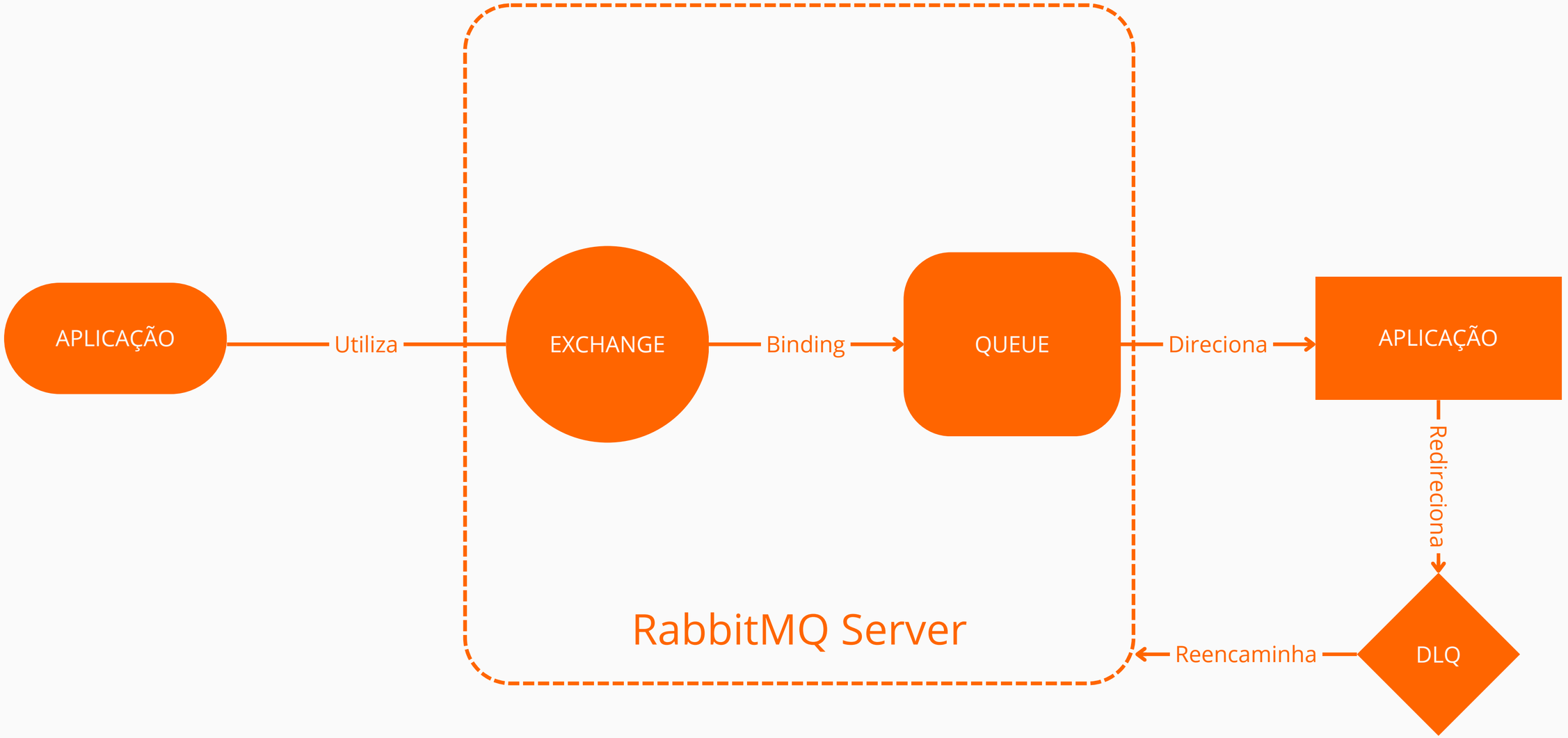

APLICAÇÃO
Utiliza
EXCHANGE
Binding
QUEUE
Direciona
APLICAÇÃO
RabbitMQ Server

Ligando os componentes

Utiliza-se de uma “routing key” para direcionar as mensagens;

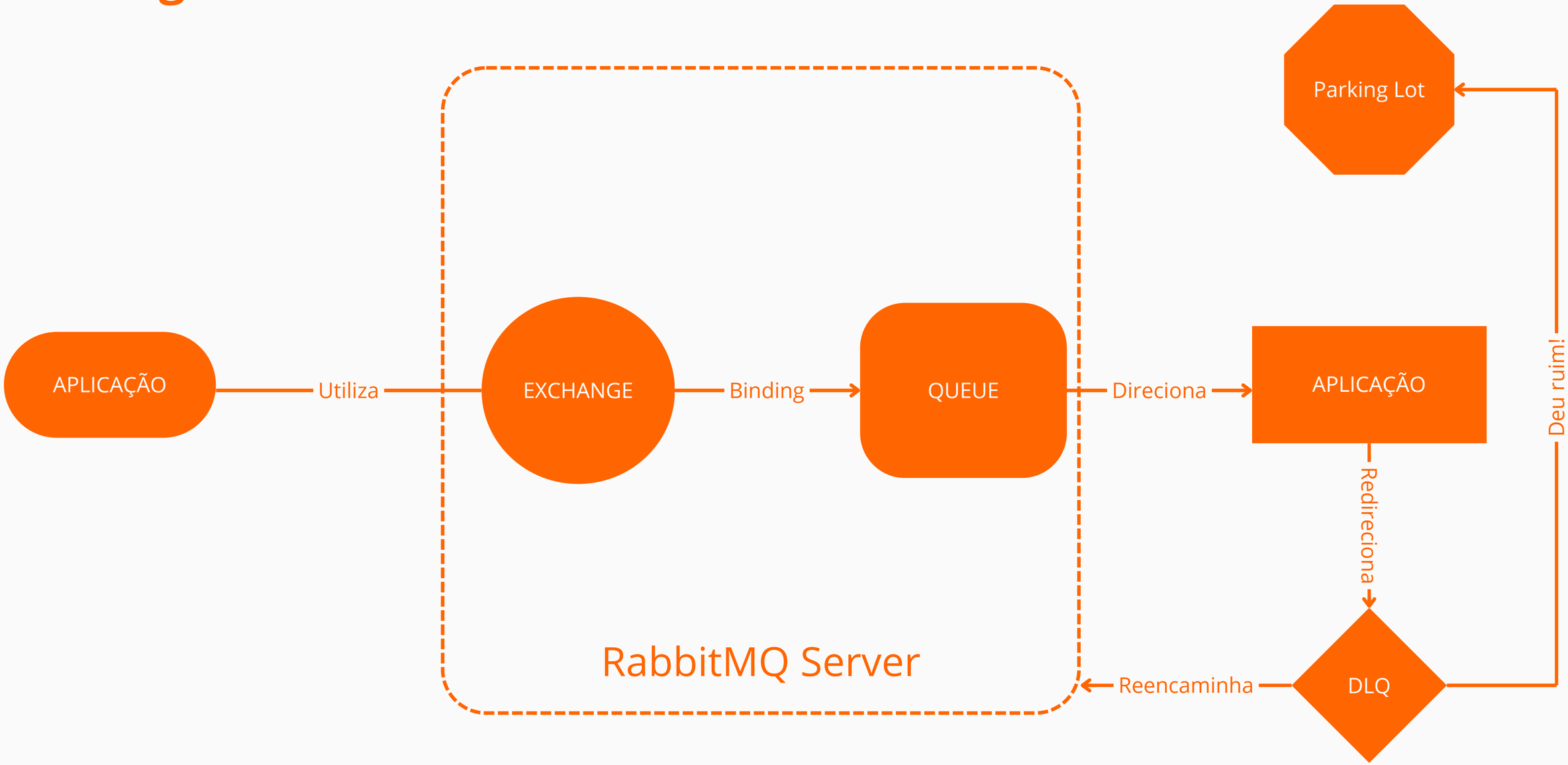
```
@Bean
Binding binding(Queue queue, TopicExchange exchange) {
   return BindingBuilder.bind(queue).to(exchange).with("test.queue.1");
}
```

DLQ


APLICAÇÃO
Utiliza
EXCHANGE
Binding
QUEUE
Direciona
APLICAÇÃO
Redireciona
RabbitMQ Server
Reencaminha
DLQ

Dead Letter Queue (DLQ)

Parking Lot


Parking Lot
APLICAÇÃO
Utiliza
EXCHANGE
Binding
QUEUE
Direciona
APLICAÇÃO
Deu ruim!
Redireciona
RabbitMQ Server
Reencaminha
DLQ

Montando o quebra cabeça

```
@Bean
MessageListenerAdapter listenerAdapter(MessageReceiver receiver) {
  return new MessageListenerAdapter(receiver, "receiveMessage");
}

@Bean
SimpleMessageListenerContainer container(ConnectionFactory connectionFactory,
MessageListenerAdapter listenerAdapter){
    SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
    container.setConnectionFactory(connectionFactory);
    container.setQueueNames(queueName);
    container.setMessageListener(listenerAdapter);
    return container;
}
```

ConnectionFactory - Busca a conexão do servidor do RabbitMQ.
MessageListenerAdapter - Utiliza uma classe criada para receber as mensagens.
container - Termina de montar o quebra cabeça.

Como isso tudo funciona junto?


APACHE
kafka®

INTRODUÇÃO

Possui uma arquitetura diferente dos outros sistemas de mensagerias convencionais

Alta disponibilidade, escalabilidade e processamento de dados em tempo real

Muito resiliente, alta confiabilidade e menos perda de dados

ARQUITETURA

- Cluster
- Broker
- Partitions
- Topico
- Producer
- Consumer
- Mensagem

CLUSTER

- Escalabilidade, distribuição de carga, resiliência
- zookeeper
- responsável por agrupar e gerenciar brokers e tópicos
- bootstrap server

BROKER

- servidor dentro do cluster, aonde é aplicado o funcionamento do sistema do kafka
- Responsável por receber, distribuir e armazenar mensagens
- gerencia as partitions dos tópcios
- possui um id e é acessível por meio de um endereço IP ou hostname
- Possui um broker líder dentre outros do mesmo topico

TOPICO

- Estrutura/canal onde as mensagens serão produzidas e consumidas e são replicadas em diferentes brokers
- É replicada em um ou mais brokers
- É subdividida em partitions
- Define fator de replica e numero de partitions ao ser criada

PARTITIONS

- Estrutura ordenada de mensagens dentro do tópico
- Rastreamento de offset e independentes entre si
- Permite paralelismo ao ser replicada em vários brokers
- Podem possuir uma chave especifica para receber a mensagem

PRODUCERS

- Envia a mensagem para o tópico especifico
- Possui sistema de confirmação de envio -> Acks
- Configuração de reenvio
- Se conecta ao bootstrap server que disponibiliza uma lista de brokers

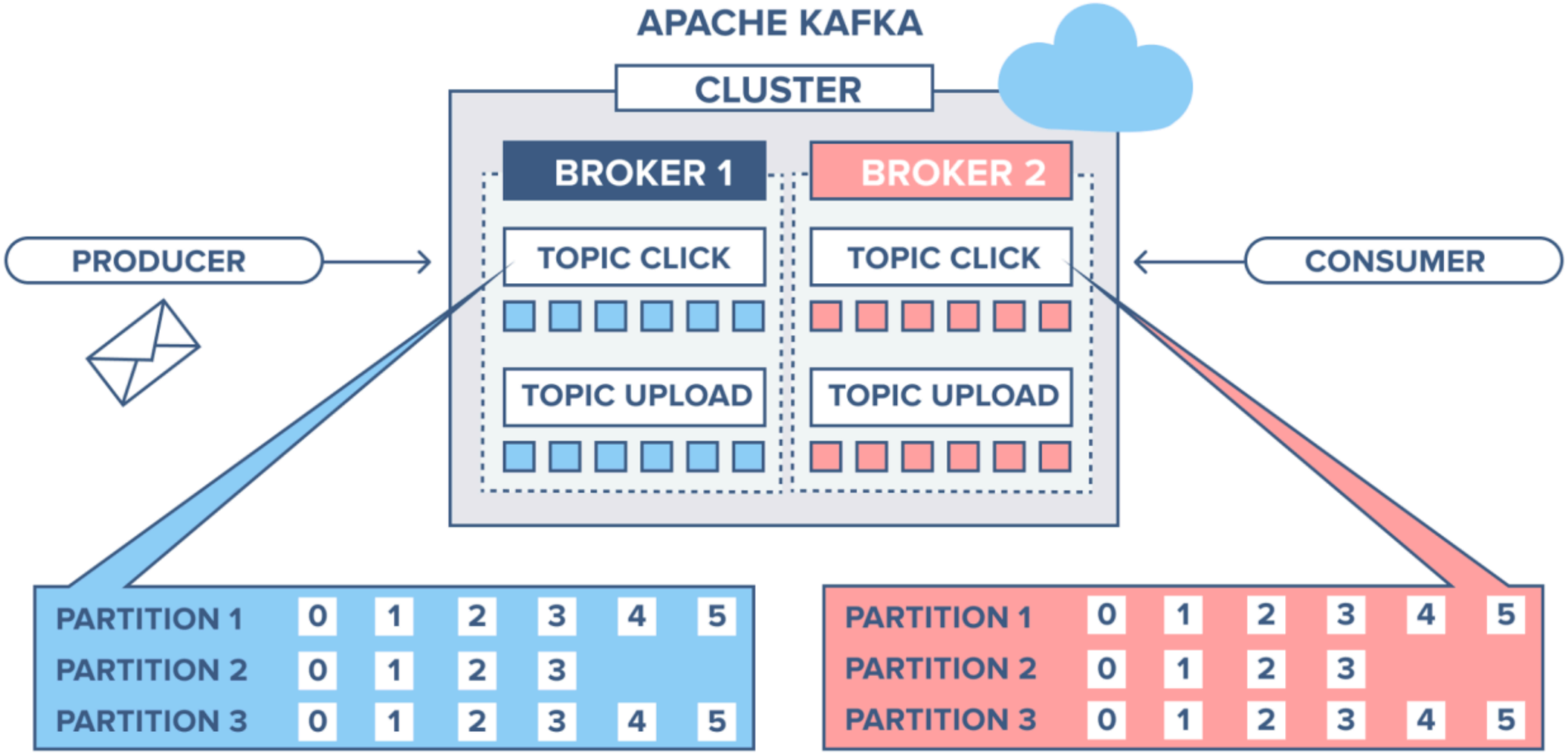
CONSUMERS

- Unidade de leitura de mensagem de um tópico
- Assim como producer, se conecta a um bootstrap server com uma lista de brokers
- Sinaliza o commit para as partitions para gerenciar o offset
- Consumer groups para distribuição de mensagens e melhorar desempenho

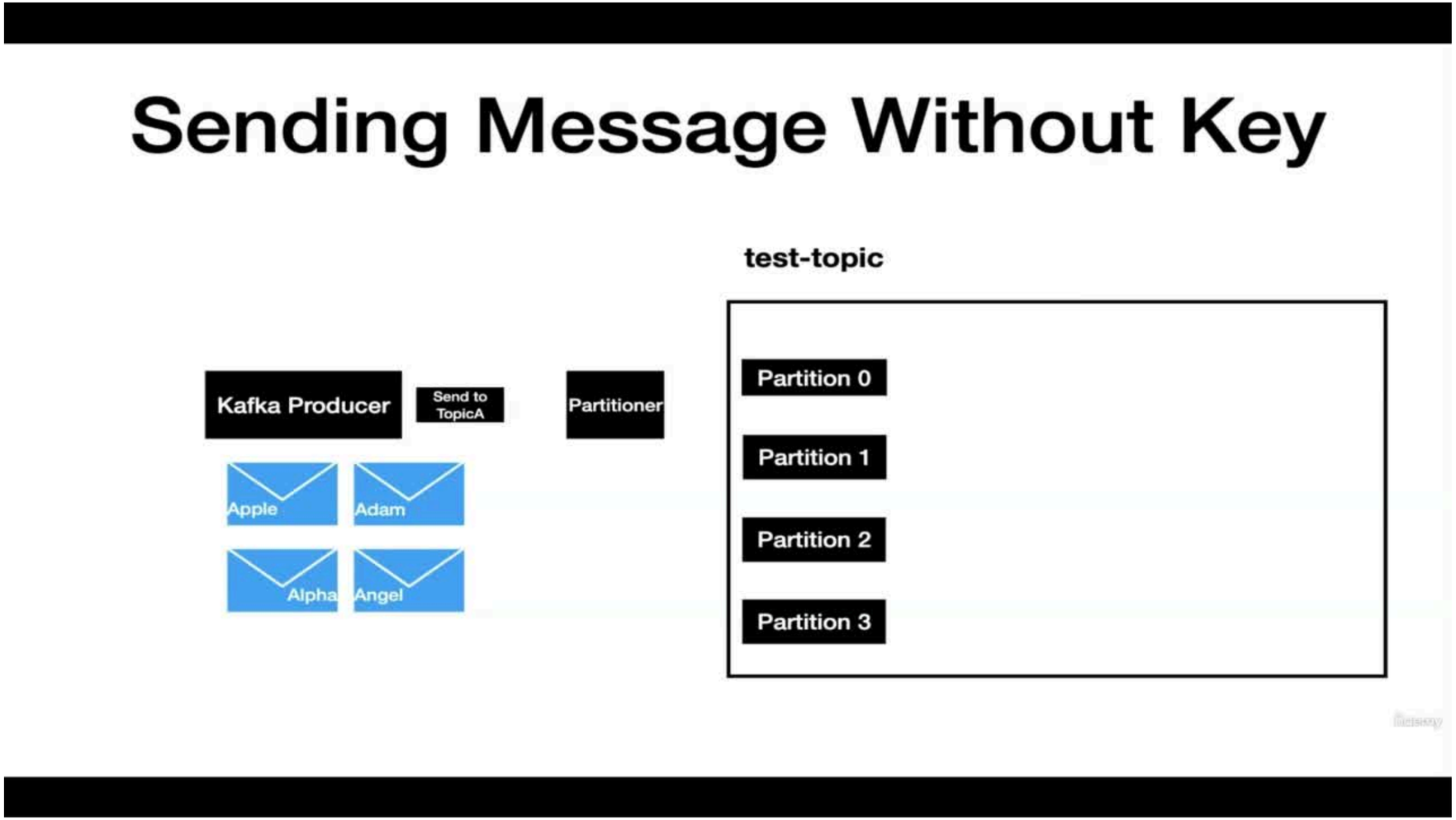
MENSAGEM

- pode possuir uma chave de identificação (opcional)
- Valores em formatos de texto comum, Json, Avro, XML, binário
- Outros valores como offset, timestamp e header

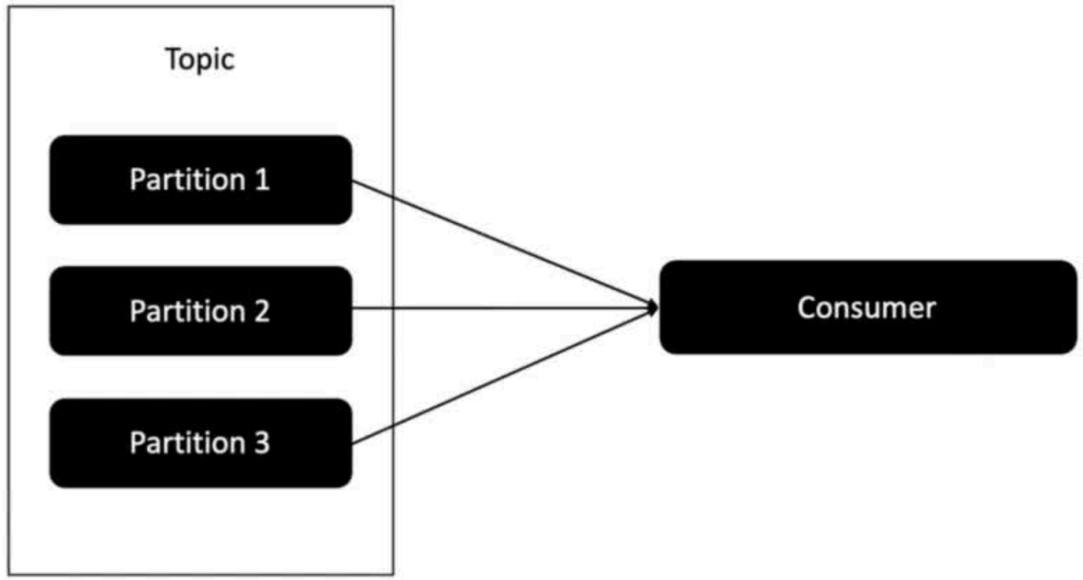
ESTRUTURA KAFKA



FUNCIONAMENTO DO PRODUCER



FUNCIONAMENTO DO CONSUMER



CONSTRUINDO UMA APLICAÇÃO KAFKA

- **Requisitos:**
  - Docker
    - Zookeeper
    - Kafka image
  - Spring boot 3.X.X
  - Java ou Kotlin na versão 17 ou 21

CONFIGURAÇÕES

- Docker:
  - image kafka
  - image zookeeper

```yaml
services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    networks:
      - kafka-net
  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_LISTENERS: INSIDE://kafka:9093,OUTSIDE://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INSIDE:PLAINTEXT,OUTSIDE:PLAINTEXT
      KAFKA_LISTENERS: INSIDE://0.0.0.0:9093,OUTSIDE://0.0.0.0:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: INSIDE
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_CREATE_TOPICS: "my_topic:3:1"
    networks:
      - kafka-net
    depends_on:
      - zookeeper
networks:
  kafka-net:
    driver: bridge
```

DEPENDENCIAS

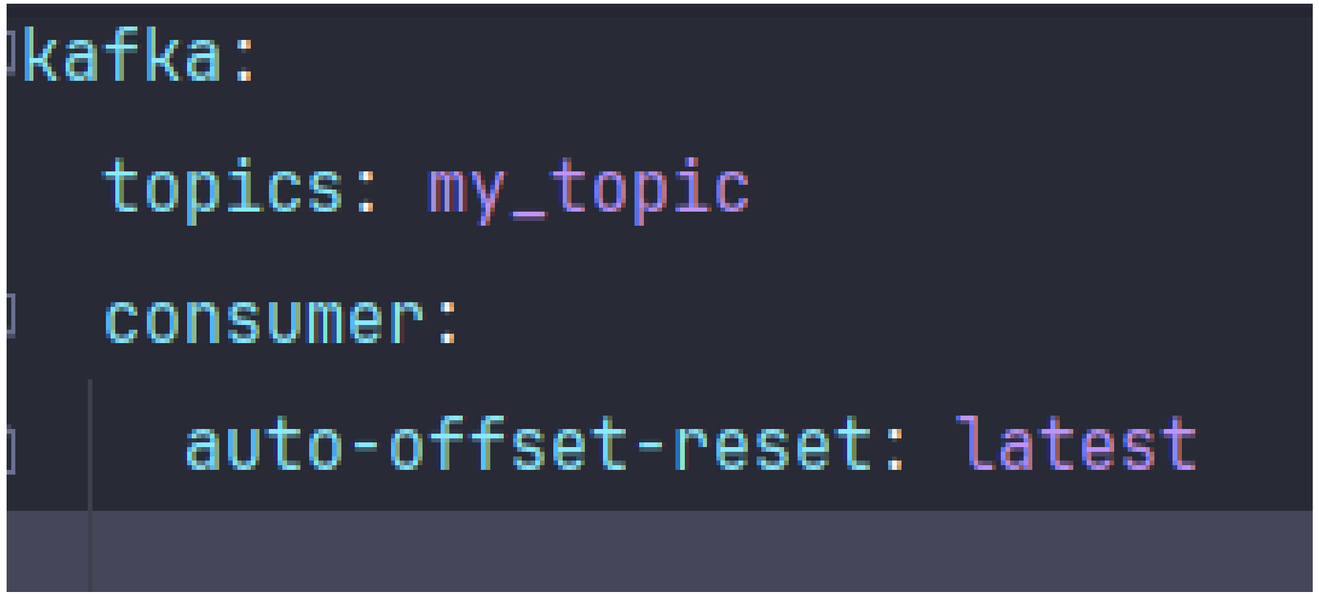
- Spring web
- Spring kafka

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'com.fasterxml.jackson.module:jackson-module-kotlin'
    implementation 'org.jetbrains.kotlin:kotlin-reflect'
    implementation 'org.springframework.kafka:spring-kafka'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.jetbrains.kotlin:kotlin-test-junit5'
    testImplementation 'org.springframework.kafka:spring-kafka-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
    implementation 'io.github.oshai:kotlin-logging-jvm:5.1.4'

}
```

CONFIGURAÇÕES

- application.yaml



CONFIGURAÇÕES

- consumer config

```kotlin
@Bean
fun consumerFactory(): ConsumerFactory<String, KafkaMessage> {
    val props = mutableMapOf<String, Any>()
    props[ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG] = "localhost:9092"
    props[ConsumerConfig.GROUP_ID_CONFIG] = "group-id"
    props[ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG] = "30000"
    props[ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG] = StringDeserializer::class.java
    props[ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG] = JsonDeserializer::class.java
    props[JsonDeserializer.TRUSTED_PACKAGES] = "*"
    return DefaultKafkaConsumerFactory(props, StringDeserializer(),
        JsonDeserializer(KafkaMessage::class.java, useHeadersIfPresent: false))
}

@Bean
fun kafkaListenerContainerFactory(): ConcurrentKafkaListenerContainerFactory<String, KafkaMessage> {
    val factory = ConcurrentKafkaListenerContainerFactory<String, KafkaMessage>()
    factory.consumerFactory = consumerFactory()
    factory.containerProperties.ackMode = ContainerProperties.AckMode.MANUAL
    return factory
}
```

CONFIGURAÇÕES

- producer config

```
@Bean
fun producerFactory(): ProducerFactory<String, KafkaMessage> {
    val configProps = mutableMapOf<String, Any>()
    configProps[ProducerConfig.BOOTSTRAP_SERVERS_CONFIG] = "localhost:9092"
    configProps[ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG] = StringSerializer::class.java
    configProps[ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG] = JsonSerializer::class.java
    return DefaultKafkaProducerFactory(configProps)
}

@Bean
fun kafkaTemplate(): KafkaTemplate<String, KafkaMessage> {
    return KafkaTemplate(producerFactory())
}
```

CONFIGURAÇÕES

- classe producer e consumer

```kotlin
@Component
class Producer
    (
    private val kafkaTemplate: KafkaTemplate<String, KafkaMessage>,
    private val logger: KLogger = KotlinLogging.logger {},
    @Value("\${kafka.topics:}")
    private val topic: String
) {
    fun send(message: KafkaMessage) {
        logger.info { "enviando mensagem para topico ${topic}..." }
        kafkaTemplate.send(message.topic, message)
        logger.info { "mensagem enviada para topico ${topic}!" }
    }
}
```

```kotlin
@Component
class Consumer {
    private val logger = KotlinLogging.logger {}

    @KafkaListener(topics = ["\${kafka.topics}"], groupId = "group_id")
    fun listener(message: ConsumerRecord<String, KafkaMessage>,
                 ack: Acknowledgment) {
        logger.info { "mensagem recebida: ${message}" }
        ack.acknowledge()
        logger.info { "acknowledge enviado!" }
    }
}
```

RODANDO A APLICAÇÃO

- acessar link do repositirio e baixar projeto https://github.com/IanZeta64/kafka-test
- rodar docker-compose
- rodar aplicação localmente
- acessar rota [POST] http://localhost:8080/kafka
  - parametro message
- conferir logs de envio e recebimento

```
2024-09-24T00:43:06.625-03:00  INFO 24664 --- [nio-8080-exec-3] c.example.kafka_test.producer.Producer   : enviando mensagem para topico my_topic...
2024-09-24T00:43:06.625-03:00  INFO 24664 --- [nio-8080-exec-3] c.example.kafka_test.producer.Producer   : mensagem enviada para topico my_topic!
2024-09-24T00:43:06.629-03:00  INFO 24664 --- [ntainer#0-0-C-1] c.example.kafka_test.consumer.Consumer   : mensagem recebida: ConsumerRecord(topic = my_topic, partition = 2, leaderEpoch = 0,
 offset = 17, CreateTime = 1727149386625, serialized key size = -1, serialized value size = 38, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = KafkaMessage
 (topic=my_topic, body=olamundo))
2024-09-24T00:43:06.629-03:00  INFO 24664 --- [ntainer#0-0-C-1] c.example.kafka_test.consumer.Consumer   : acknowledge enviado!
```

MUITO OBRIGADO!