

Apache Kafka com Java: Producers, Consumers e Integração Prática.

# **Parte II: Java com Apache Kafka**

**[[](https://www.linkedin.com/in/chmulato/)](https://www.linkedin.com/in/chmulato/)**

# **[Christian Mulato](https://www.linkedin.com/in/chmulato/)**

Desenvolvedor Java Sênior | Especialista em Back-end | Jakarta, Spring Boot, REST APIs, Docker | Engenheiro Químico

# 5 de julho de 2025

# **Visão Geral**

Esta parte mostra como integrar aplicações Java ao Apache Kafka, cobrindo desde a configuração do cliente até exemplos práticos de producers e consumers.

# **Estrutura de Pastas e Artefatos**

Os principais arquivos e diretórios desta parte estão em parte2-java/:

* docker-compose.yml: ambiente Kafka para testes locais
* pom.xml: dependências Maven do projeto Java
* src/main/java/com/mulato/: código-fonte dos Producers e Consumers
* src/test/java/com/mulato/: testes automatizados
* target/: arquivos compilados e JAR gerado após build

Consulte cada pasta para exemplos completos e adapte conforme seu ambiente.

# **Configuração do Ambiente Java**

# Java 11+ (recomendado Java 17+)

# Gerenciador de dependências: Maven ou Gradle

# Dependência principal: org.apache.kafka:kafka-clients

# **Exemplo de dependência Maven**

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka-clients</artifactId>

<version>3.7.0</version>

</dependency>

# **Producer em Java**

# Exemplo básico de envio de mensagens para um tópico Kafka:

# Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<String, String> producer = new KafkaProducer<>(props);

ProducerRecord<String, String> record = new ProducerRecord<>("meu-topico", "chave", "mensagem");

producer.send(record);

producer.close();

# **Consumer em Java**

# Exemplo básico de leitura de mensagens de um tópico:

# Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("group.id", "meu-grupo");

props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

consumer.subscribe(Collections.singletonList("meu-topico"));

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));

for (ConsumerRecord<String, String> record : records) {

System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());

}

}

# **Exemplo Prático: Producer e Consumer em Java**

A seguir, você encontra exemplos didáticos de Producer e Consumer em Java, ideais para quem está começando a integrar aplicações com o Apache Kafka. Os arquivos completos estão em:

parte2-java/src/main/java/com/mulato/PedidoProducer.java

e

parte2-java/src/main/java/com/mulato/PedidoConsumer.java

# **Como executar os exemplos**

1.**Garanta que o Kafka está rodando em** localhost:9092

Utilize odocker-compose.yml fornecido na pasta parte2-java/ para subir o ambiente local rapidamente:

docker-compose up -d

2.**Compile o projeto Java com Maven**

O projeto já possui um pom.xml pronto com todas as dependências necessárias. Basta rodar:

mvn clean compile

3.**Execute o Producer para enviar mensagens**

mvn exec:java -Dexec.mainClass="com.mulato.PedidoProducer"

O Producer simula o envio de pedidos para o tópico Kafka.

4.**Execute o Consumer para ler as mensagens**

mvn exec:java -Dexec.mainClass="com.mulato.PedidoConsumer"

O Consumer consome e imprime os pedidos recebidos.

# **Producer Java — Enviando pedidos**

O Producer é responsável por publicar mensagens (pedidos) em um tópico Kafka. Veja um exemplo básico:

# Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<String, String> producer = new KafkaProducer<>(props);

ProducerRecord<String, String> record = new ProducerRecord<>("meu-topico", "chave", "mensagem");

producer.send(record);

producer.close();

# **Consumer Java — Lendo pedidos do tópico**

O Consumer é responsável por ler as mensagens publicadas no tópico. Veja um exemplo básico:

# Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("group.id", "meu-grupo");

props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

consumer.subscribe(Collections.singletonList("meu-topico"));

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));

for (ConsumerRecord<String, String> record : records) {

System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());

}

}

**Dica:** Experimente rodar múltiplos consumers no mesmo grupo para ver como o Kafka distribui as mensagens entre eles.

Esses exemplos são apenas para fins didáticos e funcionam em ambientes locais com o Kafka rodando no padrão (localhost:9092).

# **Teste Integrado: Producer e Consumer na Prática**

Para garantir que sua aplicação Java está realmente se comunicando com o Kafka, é fundamental realizar testes de integração. O projeto já inclui um exemplo realista em

parte2-java/src/test/java/com/mulato/KafkaIntegrationTest.java

# Esse teste automatizado:

* Sobe o ambiente Kafka local (use docker-compose up -d na pasta parte2-java/).
* Envia uma mensagem para o tópico pedidos usando um Producer.
* Consome a mensagem usando um Consumer e valida se ela foi recebida corretamente.

# **Como executar o teste integrado**

1.**Suba o ambiente Kafka e Zookeeper**

# No terminal, dentro da pasta parte2-java/

docker-compose up -d

2.**Garanta que o tópico** pedidos existe

Se necessário, crie o tópico executando dentro do container Kafka:

docker exec -it <nome\_do\_container\_kafka> kafka-topics --bootstrap-server localhost:9092 --create --topic pedidos --partitions 1 --replication-factor 1

Use docker ps para descobrir o nome do container Kafka.

3.**Execute o teste com Maven**

mvn test

# O teste irá:

* Enviar uma mensagem para o tópico pedidos.
* Consumir a mensagem e validar se ela foi recebida corretamente.

4.**Finalize o ambiente**

# Após os testes, pare os containers:

docker-compose down

O teste é didático e pode ser adaptado para outros tópicos, mensagens ou cenários de integração.

# **Exercícios Práticos**

Para praticar e aprofundar os conceitos desta parte, consulte também o arquivo auxiliar:

* exercicios-parte2.md — Exercícios práticos de integração Java + Kafka, implementação de Producer/Consumer, testes e espaço para anotações.

# **Boas Práticas**

# Use consumer groups para escalabilidade

# Gerencie offsets de forma adequada (automático/manual)

# Implemente tratamento de exceções e retries

# Utilize serialização adequada (String, JSON, Avro)

# **Exercícios Sugeridos**

1. Crie um projeto Java com ***Maven*** ou ***Gradle***
2. Implemente um producer que envia mensagens simulando pedidos
3. Implemente um consumer que lê e imprime esses pedidos

# Experimente usar consumer groups e múltiplas partições

# **Recursos Recomendados**

# [**Kafka Java Client API**](https://kafka.apache.org/documentation/#producerapi)

# Exemplos oficiais: [**https://kafka.apache.org/quickstart**](https://kafka.apache.org/quickstart)

# **Código-Fonte e Exemplos**

Todo o conteúdo, exemplos práticos e arquivos de configuração desta parte estão disponíveis no repositório oficial do projeto no GitHub:

[**🔗**](https://github.com/chmulato/kafka-java-mastery) [**github.com/chmulato/kafka-java-mastery**](http://github.com/chmulato/kafka-java-mastery)

# Acesse, explore e contribua!