# Guia Teórico e Prático de Apache Kafka com Java

**Autor:** Manus AI

## 1. Introdução ao Apache Kafka

O Apache Kafka é uma plataforma de *streaming* de eventos distribuída, de código aberto, projetada para lidar com *feeds* de dados em tempo real de alto rendimento. Ele é amplamente utilizado para construir *pipelines* de dados em tempo real e aplicações de *streaming* que podem processar dados à medida que eles chegam. Originalmente desenvolvido no LinkedIn, o Kafka se tornou um padrão da indústria para sistemas que exigem durabilidade, escalabilidade e baixa latência no processamento de eventos [1].

### Por que Kafka?

Em um mundo cada vez mais orientado a dados, a capacidade de processar e reagir a eventos em tempo real é crucial. O Kafka preenche essa lacuna, permitindo que as empresas capturem, armazenem e processem fluxos de dados de forma eficiente. Ele serve como um "sistema nervoso central" para microsserviços, integrando diferentes partes de uma arquitetura distribuída através de um fluxo contínuo de informações.

## 2. Conceitos Fundamentais do Kafka

Para entender o Kafka, é essencial dominar alguns conceitos chave que formam sua arquitetura e funcionamento.

### 2.1. Tópicos (Topics)

No Kafka, os **tópicos** são as categorias ou nomes de *feeds* aos quais os dados são publicados. Pense em um tópico como uma pasta em um sistema de arquivos, onde mensagens (eventos) de um tipo específico são armazenadas. Os produtores escrevem dados em tópicos, e os consumidores leem dados de tópicos [2].

**Analogia:** Imagine um serviço de correio. Um tópico seria como uma caixa de correio específica para um tipo de correspondência, por exemplo, "Contas de Luz" ou "Notificações de Pedidos". Todas as contas de luz vão para a caixa de "Contas de Luz", e todas as notificações de pedidos vão para a caixa de "Notificações de Pedidos".

### 2.2. Partições (Partitions)

Cada tópico é dividido em uma ou mais **partições**. As partições permitem que um tópico seja distribuído em vários *brokers* (servidores Kafka), o que possibilita a escalabilidade e a tolerância a falhas. Cada partição é uma sequência ordenada e imutável de mensagens, à qual novas mensagens são anexadas. Uma mensagem dentro de uma partição recebe um *offset* sequencial, que é um identificador único dentro dessa partição [3].

**Analogia:** Voltando ao serviço de correio, se a caixa de "Contas de Luz" ficar muito grande, ela pode ser dividida em várias caixas menores, como "Contas de Luz - Zona Norte" e "Contas de Luz - Zona Sul". Cada uma dessas caixas menores é uma partição, e as cartas dentro delas ainda estão em ordem.

## 2.3. Streaming de Dados (Data Streaming)

O Kafka é fundamentalmente uma plataforma de *streaming* de dados. Isso significa que ele lida com dados em movimento, em contraste com sistemas de banco de dados tradicionais que lidam com dados em repouso. O *streaming* de dados permite o processamento contínuo e em tempo real, o que é vital para aplicações que exigem respostas imediatas a eventos, como detecção de fraudes, monitoramento de sistemas ou personalização de experiências de usuário [4].

## 2.4. O Log: O Coração do Kafka

No cerne do Kafka está o conceito de um **log de *commit*** distribuído e particionado. Cada partição é um log, uma sequência de registros imutável e somente anexável. Uma vez que uma mensagem é escrita no log, ela não pode ser alterada. Isso garante a durabilidade e a consistência dos dados. Os consumidores mantêm seu próprio *offset* dentro de cada partição, permitindo que leiam as mensagens de forma independente e em seu próprio ritmo [5].

**Analogia:** O log é como um livro-razão contábil. Cada transação (mensagem) é adicionada ao final do livro e nunca é apagada ou alterada. Você sempre pode voltar e ver o histórico completo das transações.

## 2.5. Cache e Durabilidade

Embora o termo "cache" possa sugerir dados temporários, no contexto do Kafka, ele se refere mais à forma como os dados são gerenciados para desempenho e durabilidade. O Kafka persiste todas as mensagens em disco, mesmo que os consumidores as tenham lido. Ele usa o sistema de arquivos do sistema operacional para armazenar os dados, aproveitando o **page cache** do kernel para otimizar o desempenho de leitura e escrita. Isso significa que os dados lidos recentemente ou frequentemente acessados permanecem na memória RAM, proporcionando acesso rápido, enquanto a persistência em disco garante que nenhum dado seja perdido [6].

## 2.6. Enfileiramento (Queuing) e Publicação/Assinatura (Pub/Sub)

O Kafka combina características de sistemas de enfileiramento tradicionais e sistemas de publicação/assinatura. Em um sistema de enfileiramento, as mensagens são consumidas por um único consumidor. Em um sistema Pub/Sub, as mensagens são transmitidas para todos os assinantes. O Kafka atinge um meio-termo: dentro de um **grupo de consumidores**, cada partição é consumida por apenas um consumidor, agindo como uma fila. No entanto, múltiplos grupos de consumidores podem consumir o mesmo tópico de forma independente, agindo como um modelo Pub/Sub [7].

**Tabela 1: Comparativo entre Fila, Pub/Sub e Kafka**

| Característica | Fila Tradicional | Pub/Sub Tradicional | Apache Kafka |
|---|---|---|---|
| Consumo de Mensagem | Um por mensagem | Todos os assinantes | Um consumidor por partição dentro de um grupo; múltiplos grupos podem consumir o mesmo tópico |
| Persistência | Geralmente temporária | Geralmente temporária | Persistente em disco, configurável por tempo ou tamanho |
| Ordem das Mensagens | Garantida | Não garantida | Garantida por partição |
| Escalabilidade | Limitada | Limitada | Altamente escalável via partições e grupos de consumidores |

# 3. Implementação Prática com Java

Para interagir com o Kafka em Java, utilizamos a API de cliente oficial. Vamos explorar como criar um produtor e um consumidor simples.

## 3.1. Configuração do Projeto (Maven)

Adicione as seguintes dependências ao seu `pom.xml`:

```
XML
```

```
<dependencies>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>3.6.1</version> <!-- Verifique a versão mais recente -->
    </dependency>
</dependencies>
```

## 3.2. Produtor Kafka (Kafka Producer)

Um produtor é responsável por enviar mensagens para um tópico Kafka. As mensagens são compostas por uma chave (opcional) e um valor.

Java

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;

import java.util.Properties;

public class SimpleProducer {

    public static void main(String[] args) {
        String bootstrapServers = "127.0.0.1:9092"; // Endereço do seu
broker Kafka
        String topic = "meu_primeiro_topico";

        // 1. Criar propriedades do produtor
        Properties properties = new Properties();
        properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);
        properties.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
        properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());

        // 2. Criar o produtor
        KafkaProducer<String, String> producer = new KafkaProducer<>
(properties);

        // 3. Criar um registro (mensagem)
        ProducerRecord<String, String> record = new ProducerRecord<>(topic,
"minha_chave", "Olá, Kafka do Java!");
```

```
        // 4. Enviar dados
        producer.send(record);

        // 5. Limpar e fechar o produtor
        producer.flush();
        producer.close();

        System.out.println("Mensagem enviada para o tópico: " + topic);
    }
}
```

## 3.3. Consumidor Kafka (Kafka Consumer)

Um consumidor é responsável por ler mensagens de um ou mais tópicos Kafka. Os consumidores operam dentro de **grupos de consumidores**.

Java

```
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class SimpleConsumer {

    public static void main(String[] args) {
        String bootstrapServers = "127.0.0.1:9092"; // Endereço do seu
broker Kafka
        String groupId = "meu_grupo_consumidor";
        String topic = "meu_primeiro_topico";

        // 1. Criar propriedades do consumidor
        Properties properties = new Properties();
        properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);
        properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());

properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        properties.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
```

```
"earliest"); // Ler desde o início se não houver offset salvo

        // 2. Criar o consumidor
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>
(properties);

        // 3. Assinar o tópico
        consumer.subscribe(Collections.singletonList(topic));

        // 4. Fazer polling por novas mensagens
        System.out.println("Consumidor iniciado. Aguardando mensagens no
tópico: " + topic);
        while (true) {
            ConsumerRecords<String, String> records =
consumer.poll(Duration.ofMillis(100));

            for (ConsumerRecord<String, String> record : records) {
                System.out.println("Chave: " + record.key() + ", Valor: " +
record.value() +
                                   ", Partição: " + record.partition() + ",
Offset: " + record.offset());
            }
        }
        // O consumidor não é fechado neste exemplo para continuar ouvindo.
        // Em uma aplicação real, você gerenciaria o ciclo de vida do
consumidor.
    }
}
```

## 3.4. Normalização e Transformação de Mensagens

O processo de **normalização** e **transformação** de mensagens é comum em arquiteturas de *streaming*. Isso pode envolver:

- **Enriquecimento de Dados**: Adicionar informações de outras fontes a uma mensagem.
- **Filtragem**: Remover mensagens irrelevantes.
- **Agregação**: Combinar várias mensagens em uma única.
- **Alteração de Formato**: Converter de JSON para Avro, por exemplo.

Essas operações podem ser realizadas de diversas maneiras em Java:

1. **Consumidor-Produtor (Consumer-Producer Loop)**: Um consumidor lê de um tópico, processa/normaliza a mensagem e, em seguida, um produtor envia a mensagem transformada para outro tópico. Este é o método mais simples para transformações básicas.

2. **Kafka Streams**: Uma biblioteca cliente para construir aplicações e microsserviços onde o processamento de dados é armazenado no Kafka. Ele permite processar dados em tempo real, realizar agregações, junções e transformações complexas de forma declarativa e escalável `8`.

### 3.5. Encaminhamento entre Tópicos

O encaminhamento de mensagens entre tópicos é uma funcionalidade intrínseca ao Kafka e pode ser implementado usando os métodos acima. Por exemplo, uma aplicação pode consumir de `topico_entrada`, aplicar alguma lógica de negócio (como normalização) e produzir o resultado para `topico_saida`. Isso é a base para construir *pipelines* de dados complexos e arquiteturas de microsserviços orientadas a eventos.

## 4. Mapa do Tesouro: Construindo sua Primeira Aplicação Kafka em Java

Este "mapa do tesouro" é um guia passo a passo, sem código, para você construir uma aplicação básica com Kafka e Java, aplicando os conceitos aprendidos.

1. **Configurar Ambiente Kafka**: Instale e configure um *cluster* Kafka (pode ser um único *broker* para desenvolvimento local) e Zookeeper (ou use o modo KRaft do Kafka). Verifique se os *brokers* estão rodando e acessíveis.
2. **Criar Tópicos**: Use a ferramenta de linha de comando do Kafka ( `kafka-topics.sh` ) para criar um ou mais tópicos que sua aplicação usará. Defina o número de partições e fatores de replicação adequados.
3. **Configurar Projeto Java**: Crie um novo projeto Maven ou Gradle. Adicione a dependência `kafka-clients` ao seu `pom.xml` ou `build.gradle`.
4. **Desenvolver Produtor**: Crie uma classe Java para o seu produtor. Configure as propriedades necessárias (endereço do *broker*, serializadores). Crie `ProducerRecord`s e use `producer.send()` para enviar mensagens para o tópico.
5. **Desenvolver Consumidor**: Crie uma classe Java para o seu consumidor. Configure as propriedades (endereço do *broker*, deserializadores, `group.id`, `auto.offset.reset`). Assine o tópico ( `consumer.subscribe()` ) e use um loop `while(true)` com `consumer.poll()` para buscar e processar mensagens.
6. **Testar Comunicação**: Inicie o consumidor primeiro e, em seguida, o produtor. Verifique se as mensagens enviadas pelo produtor estão sendo recebidas e impressas pelo consumidor.
7. **Implementar Lógica de Negócio (Opcional)**: Adicione lógica de processamento às mensagens recebidas pelo consumidor. Isso pode incluir validação, transformação ou

armazenamento em um banco de dados.

8. **Gerenciar Offsets**: Entenda como o Kafka gerencia os *offsets* dos consumidores. Em produção, você pode querer comitar os *offsets* manualmente para garantir o processamento *exactly-once* ou *at-least-once*.
9. **Explorar Kafka Streams (Próximo Passo)**: Para processamento de *streaming* mais complexo, como junções, agregações e janelas de tempo, comece a explorar a API Kafka Streams. Crie uma aplicação Kafka Streams que consuma de um tópico, transforme os dados e produza para outro tópico.

# 5. Referências


[1] Apache Kafka. "What is Apache Kafka?". Disponível em:
[2] Confluent. "Kafka Topics". Disponível em:
[3] Confluent. "Kafka Partitions". Disponível em:
[4] Apache Kafka. "Use Cases". Disponível em:
[5] Apache Kafka. "Design". Disponível em:
[6] Confluent. "Kafka Storage". Disponível em:
[7] Apache Kafka. "Consumer Groups". Disponível em:
[8] Apache Kafka. "Kafka Streams". Disponível em: