# Kafka

## O que veremos?

- Introdução
- Visão Geral/Arquitetura
- Produzindo Mensagens
- Consumindo Mensagens
- Schema Registry
- Kafka Connect

# O que é o Apache Kafka

- É um plataforma distribuída de streaming de eventos.
- Vai além de pub/sub tradicional.
- É open-source.
- Baixa latência e altíssimo throughput.
- Amplamente utilizado em aplicações orientadas a eventos, pipelines de dados de alta performance, streaming analytics, integração de dados e aplicação críticas.
- Ecossistema completo (Kafka Connect, Schema Registry, Streams API).

# História e Evolução do Kafka

- 2010: O LinkedIn começou a desenvolver o Kafka para lidar com a necessidade de um sistema de alto rendimento e baixa latência para dados de eventos em tempo real. Objetivos:
  - Desacoplar produtores e consumidores usando push-pull.
  - Persistência de mensagens no broker.
  - Alto throughput
  - Escalabilidade horizontal.

# História e Evolução do Kafka

- Final de 2010: publicação do Kafka no Github como um projeto Open Source.
- 2011: o projeto Kafka foi aceito pela Apache Software Foundation (ASF).
- 2012: tornou um projeto de nível superior da ASF.
- 2014: os desenvolvedores do Kafka fundaram a Confluent, que oferece suporte e treinamento às empresas que querem utilizar o Kafka.

# Kafka vs Brokers Tradicionais

| Kafka | Brokers Tradicionais |
| --- | --- |
| Log (append-only) distribuído | Baseados em filas |
| Mensagens persistidas | Mensagens são removidas após o consumo |
| Replay de mensagens possível | Replay de mensagens não é possível |
| Pull-based | Push-based |
| Escalabilidade horizontal | Escalabilidade vertical |
| Múltiplos consumer groups | Consumidores “competem” pelas mensagens |

# Kafka vs Brokers Tradicionais

| Kafka | Brokers Tradicionais |
| --- | --- |
| Roteamento simples | Roteamento complexo e mais flexível |
| Escrita sequencial em disco | Escrita randômica |
| Kafka Wire Protocol | AMQP/MQTT |
| Ideal para processamento de stream | Geralmente não suportam processamento de stream de dados |

# Replicação

- Tópicos são formados por partições e partições podem ter múltiplas réplicas.
- 2 Tipos diferentes de réplicas:
  - *Leader*: cada partição possui exatamente 1 líder. Produção e consumo (por padrão) acontece somente em partições líderes.
  - *Follower*: demais réplicas. Caso a *leader* “caia”, uma das *followers* é promovida a líder. Consomem mensagens da líder para se manter atualizadas. Podem estar *in-sync* ou *out of sync*.

## Replicação

- Uma réplica se torna *out of sync* quando se “atrasa” no consumo das mensagem.
- Somente réplicas *in-sync* são elegíveis a se tornarem líderes.

## Consumindo as mensagens

- O consumidor faz o “subscribe” nos tópicos.
- Partições são atribuídas ao consumidor e ele inicia o polling.
- Uma nova thread começar a enviar heartbeats periódicos ao Group Coordinator.
- Se o consumidor parar de enviar os heartbeats e/ou parar de consumir mensagens, o Group Coordinator entende que aquele consumidor está indisponível e inicia um rebalance.

# Schema Registry

- Produtores e consumidores precisam “concordar” no formato e schema das mensagens.

# Schema Registry

```
{
  "orderId": "ORD-123",
  "customerId": "CUST-999",
  "totalAmount": 150.75,
  "currency": "BRL",
  "createdAt": "2026-01-19T10:15:30Z"
}
```

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Order",
  "type": "object",
  "properties": {
    "orderId": { "type": "string" },
    "customerId": { "type": "string" },
    "totalAmount": { "type": "number" }
  },
  "required": ["orderId", "customerId", "totalAmount"],
  "additionalProperties": true
}
```

# Schema Registry

- O Kafka, por si mesmo, não impõe um formato; para ele, as mensagens nada mais são que um array binário.
- Nada impede que o produtor comece a enviar mensagens incompatíveis com o schema esperado pelos consumidores.
- O Schema Registry se propõe a resolver esse problema.

# Schema Registry

- É um repositório centralizado para gerenciar, versionar e validar *schemas* de dados.
- Permite definir “contratos” para produção e consumo de mensagens.
- É um componente externo aos brokers do Kafka.
- Formatos suportados:
  - JSON
  - Avro
  - Protobuf

# Avro

- O Apache Avro é um sistema de serialização de dados. O Avro fornece:
  - Estruturas de dados ricas.
  - Um formato de dados binário compacto e rápido.
  - Um arquivo de contêiner, para armazenar dados persistentes.
  - Chamada de procedimento remoto (RPC).
  - Integração simples com linguagens dinâmicas.
- Os schemas são definidos em arquivos JSON.

## Tipo de Compatibilidade

- Existem 3 tipos principais de compatibilidade no Schema Registry: *BACKWARD*, *FORWARD* e *FULL*.
- A verificação de compatibilidade é feita (por padrão), em relação à última versão.
- Existe também a versão *Transitive* de cada modo, onde cada versão nova é compatível com todas versões anteriores.
  - BACKWARD_TRANSITIVE
  - FORWARD_TRANSITIVE
  - FULL_TRANSITIVE

## Backward Compatibility

- Consumidores usando um novo schema podem ler mensagens produzidas usando o schema anterior.
- Consumidor V2 pode ler mensagem de um Produtor V1 (nunca o inverso)
- Consumidor V3 NÃO pode ler mensagens de um Produtor V1 (a menos que seja Transitive)
- Isso implica que ***todos os consumidores devem ser atualizados antes dos produtores.***

## Forward Compatibility

- Consumidores usando um schema antigo podem ler mensagens produzidas usando um novo schema.
- Consumidor V1 pode ler mensagem de um Produtor V2 (nunca o inverso)
- Consumidor V1 NÃO pode ler mensagens de um Produtor V3 (a menos que seja Transitive)
- Isso implica que ***todos os produtores devem ser atualizados antes dos consumidores.***

# Full Compatibility

- Consumidores podem ler mensagens produzida usando um schema anterior ou novo.
- Consumidor V2 pode ler mensagem de um Produtor V1, V2 ou V3
- Consumidor V3 NÃO pode ler mensagens de um Produtor V1 ou V5 (a menos que seja Transitive)
- A ordem de atualização é irrelevante.

# Kafka Connect

- Permite mover dados entre diferentes mecanismos de armazenamento ou sistemas de forma escalável e confiável.
- Principal componentes:
  - Conectores: plugins com a configuração de integração com os sistemas externos.
  - Tasks: unidades de trabalho responsáveis por mover os dados.
  - Workers: processos que executam as tasks.
  - Transforms: aplicam alterações/transformações nas mensagens.
- Os conectores são gerenciados através da API Rest do Kafka Connect.

# Kafka Connect

- Permite mover dados entre diferentes mecanismos de armazenamento ou sistemas de forma escalável e confiável.
- Principal componentes:
  - Conectores: plugins com a configuração de integração com os sistemas externos.
  - Tasks: unidades de trabalho responsáveis por mover os dados.
  - Workers: processos que executam as tasks.
  - Transforms: aplicam alterações/transformações nas mensagens.
- Os conectores são gerenciados através da API Rest do Kafka Connect.

## Kafka Connect - Conectores

- **Source connectors** são responsáveis por levar dados de sistemas externos para o Kafka.
  - JDBC
  - Debezium
  - FileStream
- **Sink connectors** fazem o caminho inverso: eles levam dados do Kafka para sistemas externos.
  - ElasticSearch
  - S3 / GCS / Azure Blob
  - MongoDB