Introdução

O que é event streaming?

Event Streaming é a prática para capturar dados em real-time de fontes de eventos como banco de dados, sensores, mobile devices, serviços de cloud, e aplicações em forma de streams de eventos, armazenando os eventos para depois serem recuperados, manipulados, processados e reagindo aos eventos em real-time ou retrospectivamente, permitindo realizar o roteamento dos eventos para diferentes destinos. Event Streaming portanto garante um fluxo contínuo e interpretação dos dados para que a informação esteja no local certo na hora certa.

Em que cenários pode-se utilizar event streaming?

Event streaming é utilizado para uma variedade de casos de uso, alguns exemplos podem ser citados:

- Processamento de pagamento e transações financeiras em real-time.
- Para continuamente capturar e analizar dados de sensores de IOT
- Para conectar, armazenar e tornar disponível dados produzidos por diferentes setores de uma empresa.
- Servir como a fundação para **plataforma de dados, arquitetura event-driven e microserviços**.

Breve historia do Apache Kafka

Apache Kafka é escrito em Scala e Java e foi criado pelos engenheiros de dados do Linkedin em 2011, sua tecnologia foi entregue para a comunidade como um projeto open-source como um alto escalável sistema de mensageria. Hoje em dia o Kafka é parte da empresa Confluent que fornece serviços de apache Kafka de maneira gerenciável e também oferece consultoria sobre o produto,

Apache Kafka é uma plataforma de event streaming, o que isto significa?

Kaka combina três capacidades chaves para implementação dos mais variados casos de uso, para streaming de eventos end-to-end.

1. **Publish** e **Subscribe** para escrever e ler streams de eventos a partir de outros sistemas.
2. **Armazenamento** de streams de eventos **durável e confiável** pelo tempo necessário.
3. **Processamento** de streams de eventos que ocorrem em tempo real ou retrospectivamente.

E todas essas funcionalidades é provido em uma **distribuída, alto escalável, elástico, tolerante a falhas e de maneira segura**.

Como kafka funciona?

Kafka é um sistema distribuído que consiste em **servidores e clientes** que se comunicam através de um protocolo de rede TCP de alta performance.

Servidores: Kafka pode ser executado como um cluster de um ou mais servidores que podem estar em múltiplos datacenters ou regiões de cloud. Alguns desses servidores fazem parte da **camada de armazenamento chamado de brokers**. Outros servidores podem rodar como **Kafka Connect** para continuamente **import** e **export** dados como streaming de eventos para integrar o Kafka com outros sistemas já existentes como bancos de dados relacionais ou outros clusters de Kafka. Para implementar casos de uso de missão crítica, um cluster de Kafka é alto escalável e tolerante a falhas: se um dos servidores falha, os outros servidores irão assumir o controle do trabalho para garantir a continuidade da operação sem perder dados.

Clientes: Eles permitem que sejam desenvolvidos aplicações distribuídas e microserviços para ler, escrever e processar streams de eventos em paralelo, que escala e de maneira tolerante a falhas mesmo em casos de problemas de rede ou falhas das máquinas. Kafka provê dezenas de clientes para as mais variadas linguagens de programação

Conceitos principais e terminologias

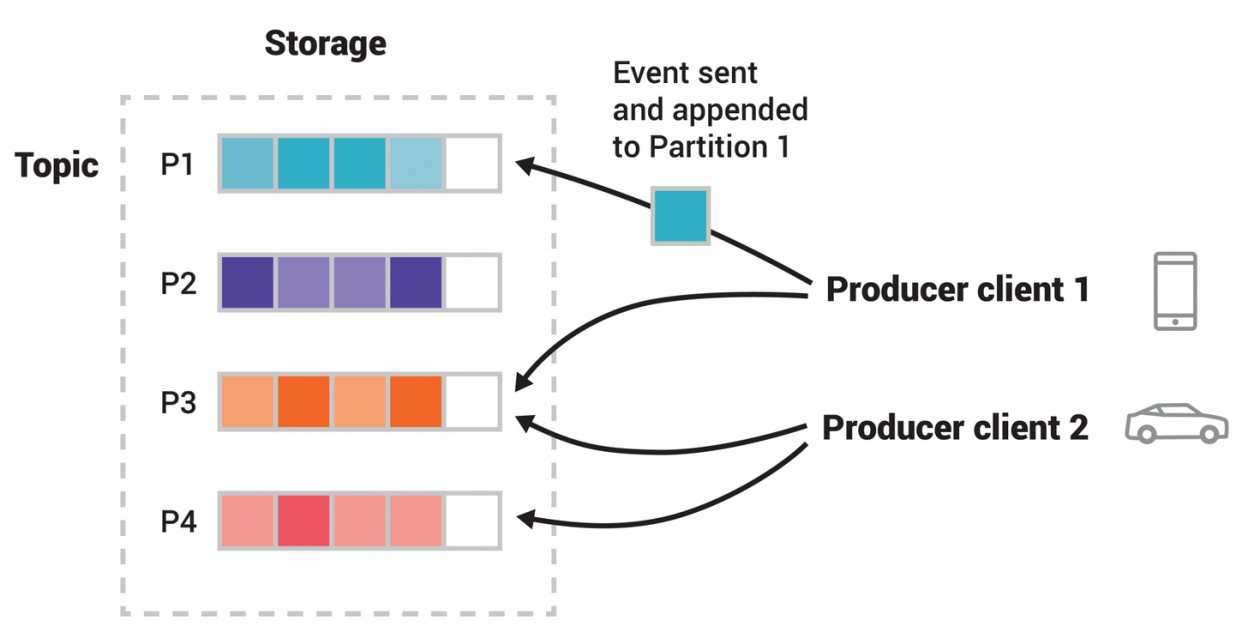
Quando realiza operações de **leitura ou escrita** no **kafka**, esse processo é realizado na forma de eventos, conceitualmente um evento tem uma **chave, valor, timestamp e headers de metadados opcionais**. Exemplo de um evento:

- **Chave do evento:** "ID do pedido 8391238198"
- **Valor do evento:** "Pedido no valor de R$1000,00"
- **Timestamp do evento:** "2020-09-07T12:00:00"

Producers: são aquelas aplicações que **publicam** eventos no Kafka e **consumers** são aqueles que se inscrevem para **ler e processar** os eventos. No Kafka produtores e consumidores são totalmente desacoplados e agnósticos um do outro sendo um conceito chave para atingir a alta escalabilidade que o Kafka é conhecido por fornecer. Por exemplo um produtor nunca deve esperar por consumidores, pois o Kafka possui mecanismos que garantem o processamento de eventos apenas uma única vez.

Eventos são organizados e armazenados em **Topics**, de maneira bem simplista um tópico é semelhante a uma pasta no sistema de arquivos e os eventos são os arquivos presentes dentro da pasta. Tópicos no Kafka são sempre **multi-producers e multi-subscribers**: um tópico pode ter zero, um ou vários produtores que escrevem os eventos e vários consumidores que consomem esses eventos. Eventos em um tópico podem ser lidos sempre que é necessário, **diferente** de sistemas tradicionais de mensageria, os eventos não são apagados depois de serem consumidos, ao invés disso é permitido realizar a configuração **por tópico** de por quanto tempo o Kafkfa irá armazenar os eventos e depois que o tempo configurado seja atingido os eventos antigos serão descartados.

Tópicos são **particionados**, o que significa que o tópico está espalhado por vários **buckets** localizados em diferentes Kafka brokers (**intermediário**). Essa distribuição dos dados é muito importante para a escalabilidade porque permite as aplicações clientes **ler e escrever** os dados a partir de vários **brokers** ao mesmo tempo. Quando um novo evento é publicado em um tópico, esse evento é adicionado em uma **partição do tópico**. Eventos com a mesma **chave** são escritos na mesma partição e o Kafka garante que qualquer consumidor de uma partição do tópico irá sempre ler os eventos das partições exatamente na mesma ordem que eles foram escritos.

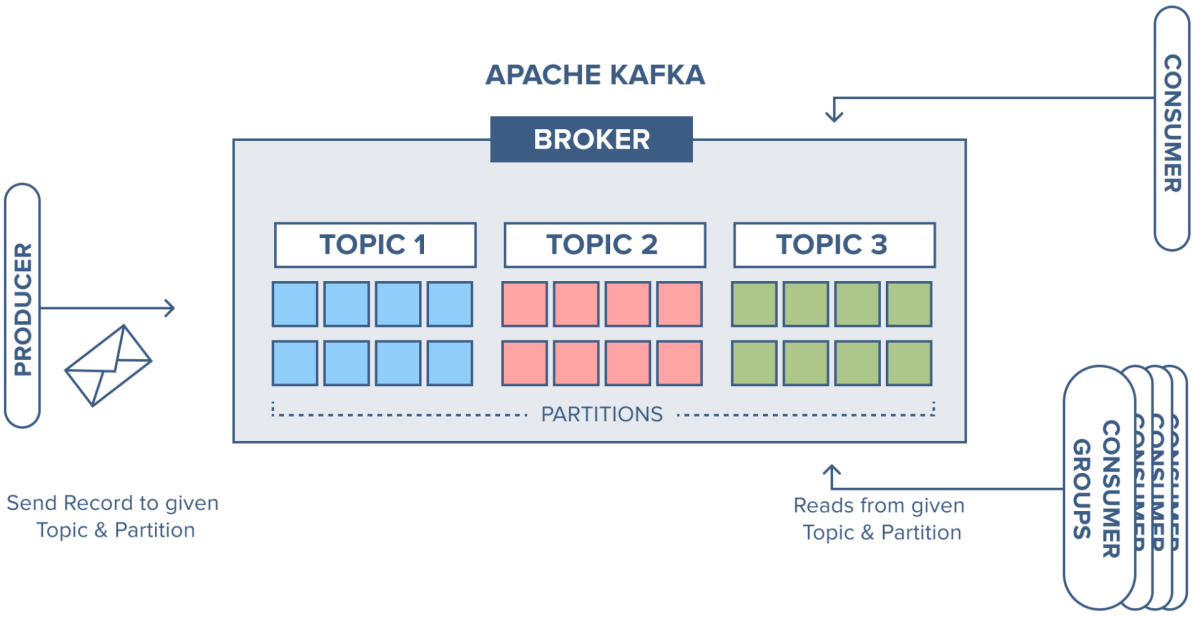


Para ter os dados tolerante a falhas e com alta disponibilidade, cada tópico pode ser **replicado**, mesmo entre regiões ou diferentes datacenters, para que sempre haja **múltiplos brokers** que tenham uma cópia dos dados em caso aconteça alguma falha, necessite realizar alguma manutenção nos brokers e etc.

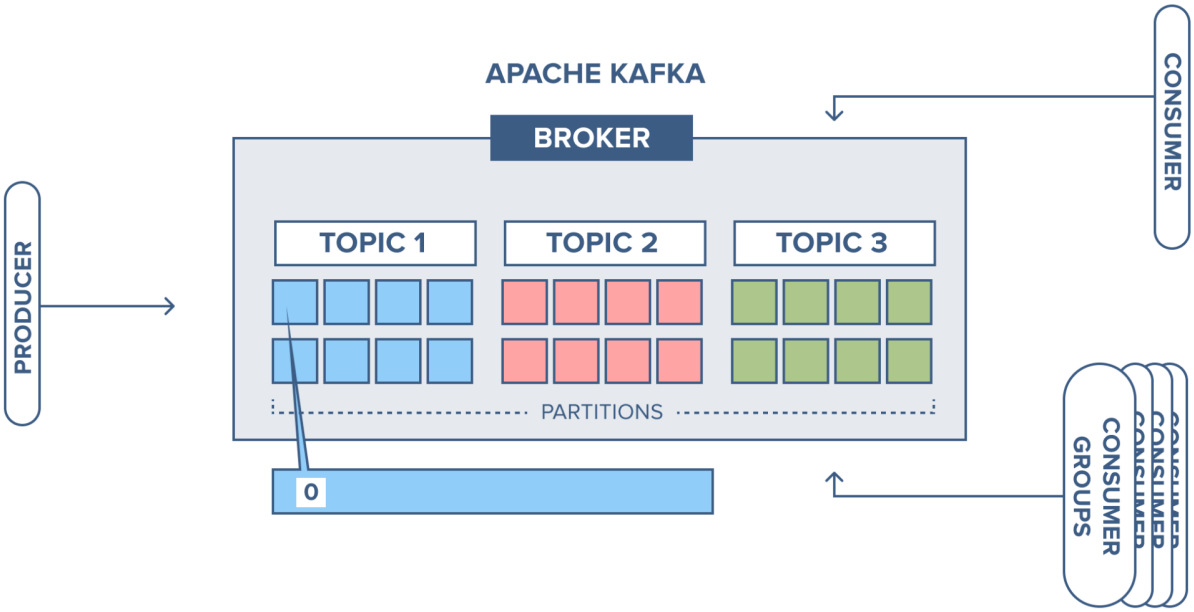
Uma configuração comum para ambientes de produção é utilizar o **fator de replicação de 3**, onde irá sempre ter 3 cópias do dado. Essa replicação é realizada no nível das partições do tópico.

Fluxo de mensagens no Kafka

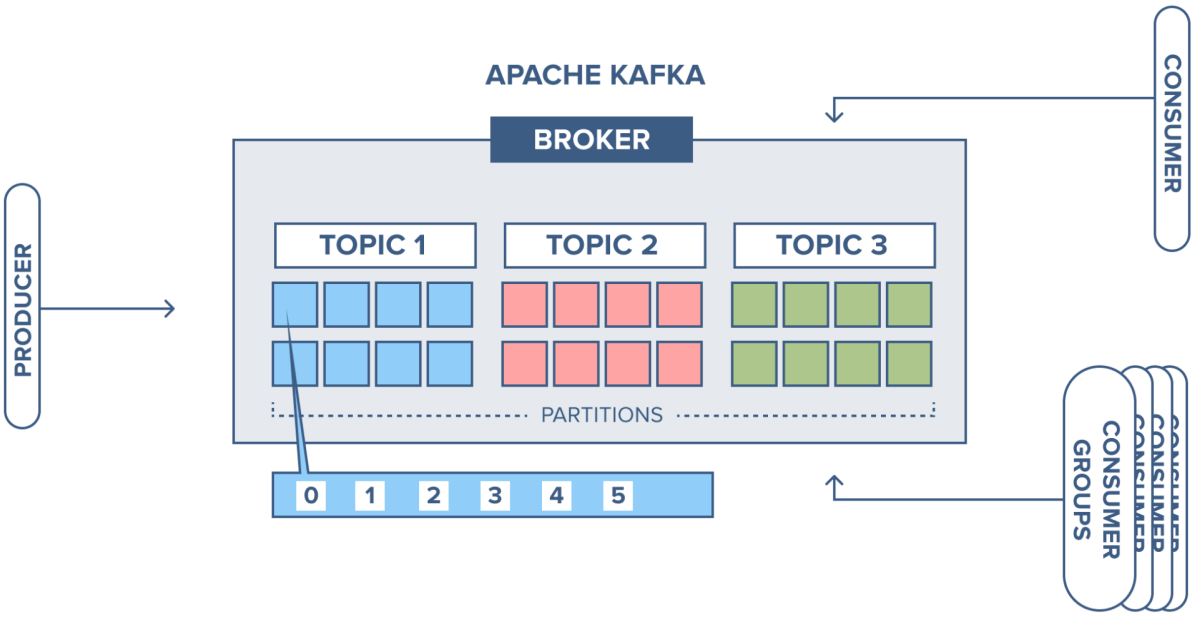
- Exemplo de um broker de kafka com 3 tópicos.



- Produtor envia uma mensagem para a partição 1 no tópico 1, e no exemplo a partição esta vazia e com isso a mensagem é gravada no offset 0, indicando que esta na primeira posição da partição.



- Novas mensagens são enviadas para a partição 1 e com isso o offset está sendo incrementado de 1 em 1.



- Esse exemplo demonstra o **commit log** termo utilizado no Kafka onde cada mensagem enviada ao tópico é adicionada ao **commit log** e não existe uma maneira de alterar os registros já existentes no log. Este também será o mesmo Offset no qual os consumers irão utilizar para iniciar o processo de leitura das mensagens presentes em cada partição.