Apache Kafka

*O que é, como funciona e por que é essencial para Dados em Tempo Real

Gabriel Rodrigues Barbosa
Faculdade de Computação (Facom)
Universidade Federal de Uberlândia (UFU)
Monte Carmelo, Brasil
gabriel.barbosa@ufu.br

***Abstract*—O Apache Kafka é uma plataforma distribuída de streaming orientada a eventos, amplamente utilizada para ingestão e processamento de grandes volumes de dados em tempo real. Este artigo tem como objetivo apresentar os fundamentos teóricos que sustentam o funcionamento do Kafka, detalhando sua arquitetura, mecanismos de retenção, ordenação e escalabilidade. Também é realizada uma comparação com outros sistemas de mensageria, como RabbitMQ, destacando as principais diferenças em termos de modelo, desempenho e protocolos utilizados. Os resultados da análise demonstram que o Kafka oferece vantagens significativas para aplicações que requerem alta disponibilidade, tolerância a falhas e throughput elevado. A abordagem apresentada contribui para o entendimento das aplicações práticas do Kafka em ambientes distribuídos e escaláveis.**

***Index Terms*—Middleware, Kafka, Apache Kafka**

I. Introdução

Desde o final da década de 1960, os sistemas distribuídos vêm se consolidando como pilares da computação moderna, impulsionando soluções robustas para desafios relacionados à escalabilidade, disponibilidade, transparência e consistência de dados. Inicialmente motivados pela necessidade de interconectar mainframes em ambientes corporativos, esses sistemas evoluíram a ponto de se tornarem componentes indispensáveis na arquitetura de aplicações contemporâneas. Andrew S. Tanenbaum, referência clássica no estudo da área, define sistemas distribuídos como um conjunto de computadores independentes que se apresentam ao usuário como um sistema único e coerente, cooperando entre si para alcançar objetivos comuns [1].

Com o avanço da tecnologia e a popularização da internet a partir dos anos 2000, observou-se uma verdadeira revolução na forma como as informações são geradas, compartilhadas e processadas. A democratização do acesso à rede mundial tornou possível o surgimento de plataformas digitais, como redes sociais e serviços de streaming, que passaram a demandar arquiteturas altamente distribuídas e resilientes. Nesse contexto, emergiram novos paradigmas como Big Data e Internet das Coisas (IoT), caracterizados por grandes volumes de dados e pela necessidade de comunicação em tempo real entre dispositivos, sistemas e usuários. A crescente demanda por decisões em tempo real em aplicações críticas, como sistemas financeiros, plataformas de recomendação e monitoramento de sensores, impõe o desafio de processar dados em movimento com baixa latência e alta confiabilidade.

A complexidade crescente das aplicações modernas trouxe à tona a importância dos middleware — camadas intermediárias de software que viabilizam a comunicação entre sistemas distribuídos de maneira transparente e eficiente. Middleware atua como um elo entre diferentes componentes da infraestrutura tecnológica, oferecendo suporte à troca de mensagens, controle de concorrência, gerenciamento de filas e abstração da localização dos recursos. Dentre os diversos middleware existentes, como RabbitMQ, ActiveMQ e ZeroMQ, o Apache Kafka se destaca por sua capacidade de lidar com grandes volumes de dados em tempo real de forma distribuída e tolerante a falhas.

A arquitetura orientada a eventos (Event-Driven Architecture – EDA) é um modelo computacional no qual componentes do sistema comunicam-se por meio da emissão e consumo de eventos, permitindo maior desacoplamento, escalabilidade e reatividade no processamento de informações. Essa abordagem tornou-se fundamental em sistemas modernos que exigem respostas rápidas a mudanças de estado, como em transações bancárias, sistemas de recomendação e monitoramento de logs.

Entre os diversos middleware desenvolvidos nas últimas décadas, destaca-se o Apache Kafka, criado em 2011 pela equipe do LinkedIn e posteriormente tornado projeto da Apache Software Foundation. Projetado especificamente para lidar com fluxos contínuos de dados, Kafka tornou-se referência em processamento de eventos em tempo real, com adoção por empresas como Netflix, Uber, Spotify e LinkedIn. Sua arquitetura distribuída garante alta performance, tolerância a falhas e escalabilidade horizontal, características fundamentais em cenários de Big Data e microsserviços. Nesse cenário, o Apache Kafka não apenas viabiliza o transporte eficiente de dados entre sistemas, como também atua como pilar central em arquiteturas modernas de dados, sendo utilizado como backbone para integração de microsserviços, pipelines de dados e sistemas analíticos em tempo real [2].

Este artigo tem como objetivo apresentar uma análise abrangente sobre o papel dos sistemas distribuídos no cenário atual, com foco na atuação do middleware Apache Kafka. Serão explorados os fundamentos teóricos que sustentam essa tecnologia, bem como os desafios contemporâneos relacionados ao processamento e distribuição de dados em tempo real.

Além disso, serão discutidos casos de uso relevantes e o impacto do Kafka na infraestrutura de grandes corporações.

II. Fundamentação Teórica

No contexto de sistemas distribuídos, o middleware é uma camada essencial de software que atua como intermediária entre o sistema operacional e os aplicativos distribuídos. Tanenbaum e Van Steen definem o middleware como "a camada de software que fica entre o sistema operacional e os aplicativos em cada lado de um sistema de computação distribuído em uma rede" [1]. Essa camada tem como principal objetivo abstrair a complexidade da comunicação entre componentes distribuídos, oferecendo transparência, coordenação e integração entre os diversos nós de um sistema.

Com o avanço das tecnologias e a intensificação do uso de arquiteturas baseadas em microsserviços, Internet das Coisas (IoT) e Big Data, os middlewares modernos passaram a desempenhar funções ainda mais estratégicas. Entre suas principais responsabilidades estão o gerenciamento da troca de mensagens, controle de concorrência, abstração de localização de recursos, garantia de confiabilidade e tolerância a falhas. Em especial, os sistemas de mensageria baseados em *message brokers* têm se destacado por sua capacidade de operar com altos volumes de dados e garantir a escalabilidade e a integridade na comunicação entre serviços distribuídos.

Nesse cenário, tecnologias de *streaming* de eventos emergem como soluções robustas para lidar com fluxos contínuos de dados em tempo real. Tais soluções são fundamentais para aplicações que exigem baixa latência, como monitoramento de sensores, sistemas financeiros e plataformas de recomendação.

A. Apache Kafka

O Apache Kafka é um middleware distribuído voltado ao processamento de fluxos de dados baseado em eventos. Desenvolvido originalmente pelo LinkedIn e posteriormente tornado projeto da Apache Software Foundation, o Kafka se consolidou como uma plataforma de *event streaming* altamente escalável, tolerante a falhas e eficiente [3]. Sua arquitetura orientada a eventos o torna ideal para ambientes que exigem comunicação assíncrona, integração de sistemas heterogêneos e alta disponibilidade.

A tecnologia é amplamente adotada por empresas como Netflix, Spotify, Uber e Twitter (atualmente X), que compartilham a necessidade de lidar com grandes volumes de dados em tempo real [6]. O Kafka opera predominantemente sobre o Protocolo de Controle de Transmissão (TCP), da camada de transporte do modelo OSI, garantindo a entrega ordenada e confiável das mensagens trocadas entre os componentes [7].

Conforme ilustrado na Figura 1, a arquitetura do Apache Kafka é composta por diversos elementos integrados. O *Kafka Broker* atua como o nó central do sistema, responsável por armazenar e encaminhar mensagens aos consumidores. A coordenação entre os brokers, que anteriormente dependia do Apache ZooKeeper, pode atualmente ser gerenciada por meio do *Kafka Raft Metadata Mode* (KRaft), que elimina a necessidade do ZooKeeper, simplificando a administração do cluster.

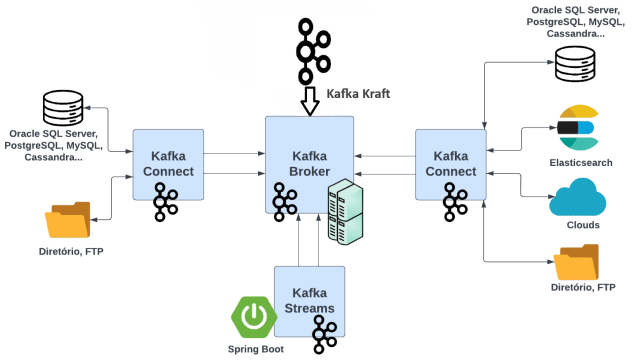

Fig. 1. Arquitetura do Apache Kafka e seus componentes.

Além disso, o módulo *Kafka Connect* permite a integração com sistemas externos, como bancos de dados (PostgreSQL, MySQL, Oracle), sistemas de arquivos (FTP) e plataformas em nuvem. O *Kafka Streams* é responsável pelo processamento de dados em tempo real, oferecendo uma API poderosa para transformar, agregar e filtrar fluxos de dados. Essa integração entre publicação, persistência, processamento e consumo torna o Kafka uma solução completa para arquiteturas orientadas a eventos.

1) Comunicação Assíncrona: A comunicação assíncrona é um paradigma no qual os componentes de um sistema interagem sem a necessidade de sincronização temporal, ou seja, sem que o emissor e o receptor estejam ativos simultaneamente. Nesse modelo, as mensagens são encaminhadas a um intermediário — geralmente uma fila ou um tópico — e processadas de forma independente pelo consumidor. Esse desacoplamento entre produtor e consumidor é fundamental para garantir resiliência, escalabilidade e flexibilidade em sistemas distribuídos [8].

O Apache Kafka adota essa abordagem ao permitir que produtores publiquem mensagens em tópicos, que ficam armazenadas de forma persistente até serem consumidas. Essa arquitetura baseada em logs imutáveis garante a durabilidade e a ordenação dos eventos, o que favorece o uso de padrões arquiteturais como *event sourcing* e *CQRS* (Command Query Responsibility Segregation) [9]. Tais padrões são amplamente utilizados no desenvolvimento de aplicações baseadas em microsserviços, onde a independência e a reatividade dos componentes são requisitos essenciais.

Ao adotar a comunicação assíncrona e eventos imutáveis, o Kafka se destaca como uma solução robusta para a construção de sistemas reativos, escaláveis e resilientes, alinhados às demandas das aplicações modernas e às boas práticas de engenharia de software distribuído.

III. Arquitetura e Funcionamento do Apache Kafka

O Apache Kafka é baseado em uma arquitetura distribuída orientada a logs, composta por três principais entidades: *producers* (produtores), *brokers* (servidores) e *consumers* (consumidores). Os dados são organizados em estruturas chamadas *tópicos* (topics), os quais são divididos em múltiplas *partições* (partitions). Essa divisão permite o paralelismo na leitura e gravação de dados, promovendo maior escalabilidade.

Cada partição funciona como um log imutável de mensagens, onde os eventos são adicionados de forma sequencial no final da fila. Os produtores são responsáveis por publicar eventos em tópicos específicos. Ao produzir uma nova mensagem, o Kafka a direciona para uma partição do tópico, com base em regras definidas, como round-robin, hash da chave ou escolha manual do produtor.

A Figura 2 ilustra esse processo. Nela, dois clientes produtores (*Producer client 1* e *Producer client 2*) enviam eventos que são distribuídos entre as partições de um tópico. O evento gerado é armazenado sequencialmente ao final da partição à qual foi atribuído, respeitando a ordem de entrada dentro da partição. Essa característica garante consistência sequencial, importante para muitos sistemas consumidores.

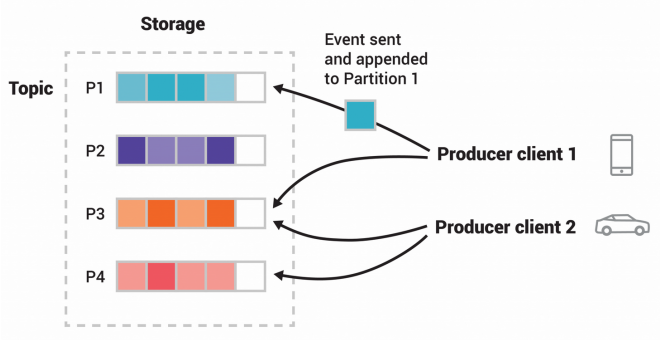

Fig. 2. Ilustração do particionamento de tópicos no Apache Kafka. Cada produtor envia eventos que são armazenados sequencialmente em partições específicas do tópico.

Além da estrutura de tópicos e partições, o Kafka mantém metadados no Zookeeper (ou, em versões mais recentes, internamente por meio do *KRaft*), que auxiliam na coordenação entre brokers e na atribuição de partições aos consumidores. O mecanismo de replicação garante alta disponibilidade e tolerância a falhas, permitindo que múltiplas cópias de cada partição sejam armazenadas em brokers distintos.

A separação entre produtores, tópicos e consumidores desacopla os componentes do sistema, promovendo robustez e flexibilidade. Esse modelo torna o Kafka ideal para aplicações que exigem alta taxa de ingestão de dados, como sistemas de monitoramento, pipelines de dados em tempo real, logs de aplicativos e integrações entre microsserviços.

A. Comparativo entre Kafka e RabbitMQ

Para compreender melhor as características do Apache Kafka, é relevante compará-lo a outros sistemas de mensageria amplamente utilizados. O RabbitMQ, por exemplo, é uma solução tradicional baseada em filas e amplamente adotada em arquiteturas de microsserviços. A Tabela I apresenta uma comparação entre o Kafka e o RabbitMQ, considerando critérios como modelo de operação, retenção, throughput, escalabilidade, ordenação de mensagens e protocolos suportados.

TABLE I
Comparativo entre Apache Kafka e RabbitMQ

Critério	**Kafka**	**RabbitMQ**
Modelo	Log distribuído (armazenamento)	Filas tradicionais (memória/disk)
Retenção	Persistente (horas/dias)	Efêmera (após consumo)
Throughput	Alto (milhões de msg/segundo)	Moderado (milhares de msg/segundo)
Escalabilidade	Horizontal nativa	Limitada (cluster manual)
Ordem de mensagens	Garantida por partição	Não garantida nativamente
Protocolo	TCP proprietário (`Kafka protocol`)	AMQP, MQTT, STOMP

O Apache Kafka destaca-se por sua arquitetura distribuída nativa e capacidade de escalar horizontalmente de forma eficiente, garantindo alta performance e confiabilidade mesmo em sistemas com grande volume de dados. A ordenação das mensagens é preservada dentro de uma partição, o que favorece o processamento sequencial em sistemas sensíveis à ordem de eventos. Já o RabbitMQ, embora flexível em protocolos e fácil de integrar, apresenta limitações quanto à escalabilidade nativa e ordenação.

Essas diferenças tornam o Kafka mais indicado para sistemas de análise em tempo real, processamento de eventos e integração com pipelines de dados, enquanto o RabbitMQ é amplamente adotado em sistemas que priorizam simplicidade de uso, múltiplos protocolos e interoperabilidade com legados.

Fonte: Adaptado de Kreps et al. (2011) e documentação oficial dos projetos Apache Kafka e RabbitMQ.

IV. Funcionamento do Apache Kafka: Passo a Passo

O Apache Kafka opera como uma plataforma de mensageria distribuída orientada a logs, projetada para alta taxa de transferência e baixa latência. A seguir, descreve-se o fluxo completo de funcionamento do sistema em um ambiente típico de publicação e consumo de mensagens:

A. 1. Criação de Tópicos

O primeiro passo é a criação de *tópicos*, que funcionam como canais lógicos para a publicação de dados. Cada tópico pode ser subdividido em múltiplas *partições*, permitindo o processamento paralelo das mensagens. Essa configuração é realizada no cluster Kafka por meio de comandos administrativos ou automaticamente via API.

B. 2. Produção de Mensagens

Aplicações chamadas *producers* são responsáveis por enviar dados ao Kafka. Cada mensagem contém uma chave (*key*), um valor (*value*) e metadados adicionais, como carimbo de tempo (*timestamp*). O Kafka utiliza a chave para determinar a partição de destino da mensagem, assegurando que mensagens com a mesma chave sejam direcionadas à mesma partição, mantendo a ordem.

C. 3. Armazenamento em Partições

Ao chegar ao broker Kafka, a mensagem é anexada ao final do log da partição correspondente, de forma imutável e sequencial. Cada partição é armazenada como um arquivo físico no disco do broker, o que permite alto desempenho em escrita sequencial.

D. 4. Replicação para Tolerância a Falhas

Para garantir disponibilidade e resiliência, cada partição pode ter múltiplas réplicas distribuídas entre brokers diferentes. Um dos brokers é designado como líder da partição, sendo responsável por coordenar todas as operações de leitura e escrita. Os demais atuam como seguidores, replicando os dados do líder de forma assíncrona.

E. 5. Consumo por Clientes

Os *consumers* são aplicações que leem dados dos tópicos. Eles se organizam em *grupos de consumidores* (consumer groups), onde cada partição é atribuída a apenas um consumidor do grupo, garantindo o paralelismo sem sobreposição de leitura. Os consumidores mantêm o controle do deslocamento (*offset*), que indica até qual ponto da partição os dados já foram lidos.

F. 6. Retenção e Reprocessamento

O Kafka armazena as mensagens por um tempo configurável, mesmo após o consumo. Isso permite o reprocessamento de dados, auditoria e recuperação em caso de falha. A retenção pode ser definida por tempo ou por tamanho de log, tornando o sistema altamente configurável para diferentes casos de uso.

G. 7. Monitoramento e Gerenciamento

Por fim, o funcionamento do cluster Kafka é gerenciado por ferramentas integradas como o *Kafka Manager*, Prometheus e Grafana. Métricas como latência, throughput, uso de disco e status das réplicas são monitoradas continuamente para garantir a operação eficiente do sistema.

Este fluxo de funcionamento permite que o Kafka seja utilizado em arquiteturas de dados modernas, como pipelines de processamento em tempo real, microsserviços e sistemas de análise distribuída.

V. Conclusão

Neste artigo, foi apresentado um estudo aprofundado sobre o Apache Kafka, abordando desde os conceitos teóricos fundamentais até a análise prática de sua arquitetura e funcionamento. O Kafka mostrou-se uma solução robusta para o processamento de fluxos de dados em tempo real, oferecendo alta escalabilidade, persistência de mensagens e garantia de ordenação por partição.

A comparação com o RabbitMQ evidenciou as diferenças de modelo e desempenho entre os sistemas, reforçando a adequação do Kafka para cenários de big data, analytics e integração de microserviços em larga escala. Sua estrutura baseada em logs distribuídos e a capacidade de retenção prolongada posicionam-no como uma tecnologia chave em arquiteturas orientadas a eventos.

Como trabalho futuro, recomenda-se a implementação prática de um ambiente com Apache Kafka, utilizando múltiplos produtores e consumidores em diferentes domínios de aplicação, como monitoramento, processamento em tempo real e integração com sistemas de aprendizado de máquina. Tais estudos complementariam a base teórica e possibilitariam uma análise de desempenho em cenários reais.

References

References

[1] TANENBAUM, Andrew S.; VAN STEEN, Maarten. Sistemas distribuídos: princípios e paradigmas. 2. ed. São Paulo: Pearson Education do Brasil, 2008.

[2] COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; et al. Sistemas distribuídos. 5. ed. Porto Alegre: Bookman, 2013. E-book. p.18. ISBN 9788582600542. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9788582600542/. Acesso em: 15 jul. 2025.

[3] KREPS, Jay et al. Kafka: a distributed messaging system for log processing. In: *Proceedings of the NetDB*, 2011.

[4] Apache Kafka. Apache Kafka. Published 2024. Acesso em: 13 jul. 2025. https://kafka.apache.org/

[5] CASTRO, Marcos Sérgio. *Sistemas distribuídos: fundamentos e aplicações*. São Paulo: Novatec, 2012.

[6] CONFLUENT. Apache Kafka Documentation. Disponível em: https://kafka.apache.org/documentation/. Acesso em: 18 jul. 2025.

[7] KUROSE, James F.; ROSS, Keith W. *Redes de computadores e a internet: uma abordagem top-down*. 6. ed. São Paulo: Pearson, 2018.

[8] TANENBAUM, Andrew S.; VAN STEEN, Maarten. *Sistemas distribuídos: princípios e paradigmas*. 2. ed. São Paulo: Pearson, 2008.

[9] FOWLER, Martin. Event Sourcing. 2005. Disponível em: https://martinfowler.com/eaaDev/EventSourcing.html. Acesso em: 15 jul. 2025.