Análise e processamento de dados usando Apache Kafka, Spark e Pinot

Ingrid Iplinsky de Oliveira

Universidade Federal de Uberlândia
Faculdade de Ciência da Computação
Bacharelado em Sistemas de Informação

Uberlândia
2024

Ingrid Iplinsky de Oliveira

Análise e processamento de dados usando Apache Kafka, Spark e Pinot

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.
Área de concentração: Sistema de Informação

Orientador: Daniele Carvalho Oliveira

Uberlândia
2024

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

A474m Sobrenome, Nome do aluno, 1979-

2014 Título do Trabalho / Nome e Sobrenome do aluno. - 2014.
81 f. : il.

Orientador: Nome do Orientador.

Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1.Computação - Teses. 2. Simulação (Computadores) - Teses. I. Sobrenome, Nome do orientador. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

Dedico este trabalho com imenso carinho e gratidão aos meus queridos pais, Aline e Eli, que sempre estiveram ao meu lado, incentivando e apoiando cada passo desta jornada acadêmica. Obrigado por serem minha fonte constante de inspiração e por acreditarem em mim incondicionalmente.

Ao meu amado marido, Renato, expresso meu profundo agradecimento por seu amor, compreensão e paciência ao longo desses anos de estudo. Sua presença e apoio foram fundamentais para minha perseverança e sucesso. Ao meu querido filho, João Lucas, que trouxe luz e alegria aos meus dias, obrigado por compreender os momentos em que precisei me ausentar para estudar e por ser minha motivação extra para buscar sempre o melhor.

Cada um de vocês foi parte essencial desta conquista. Este trabalho não apenas representa meu esforço pessoal, mas também é uma celebração da nossa jornada juntos. Que este seja apenas o começo de muitas vitórias que compartilharemos. Com amor e gratidão, Ingrid.

Agradecimentos

Gostaria de expressar minha profunda gratidão à minha orientadora, Professora Daniele Carvalho Oliveira, pelo seu apoio inestimável, orientação e dedicação durante todo o período de elaboração deste trabalho. Sua expertise, paciência e encorajamento foram fundamentais para o meu crescimento acadêmico e pessoal. Obrigada por acreditar no meu potencial e por compartilhar seu conhecimento de maneira tão generosa.

Também desejo estender meus sinceros agradecimentos a todos os docentes que cruzaram meu caminho ao longo da faculdade. Cada um de vocês contribuiu de maneira significativa para minha formação, desafiando-me a crescer, inspirando-me com sua paixão pelo ensino e guiando-me na busca pelo conhecimento.

Resumo

O volume de dados gerados pelos usuários tem crescido de forma exponencial. No entanto, muitas empresas enfrentam desafios na captura e análise dessas informações, o que impacta diretamente na geração de valor para seus negócios. Com isso o presente trabalho de conclusão de curso tem o objetivo de apresentar uma forma de realizar análise de dados no âmbito de *Big Data*, utilizando os softwares Apache Kafka para consumir os dados e distribuir dentro do ecossistema, o Apache Spark para realizar análises e filtragem dos dados e o Apache Pinot para fazer o armazenamento e possibilitar futuras consultas. E para a análise utilizou-se dados da *Application Programming Interface* (API) do Twitter disponibilizado na Kaggle, visando a variedade de dados e permitindo analisar tendências. Os resultados obtidos demonstram a viabilidade e eficácia da metodologia proposta. O sistema desenvolvido foi capaz de lidar com a ingestão e processamento de dados, demonstrando a escalabilidade e desempenho das ferramentas Apache Kafka, Apache Spark e Apache Pinot. Além disso, as análises realizadas forneceram insights valiosos sobre os dados do Twitter, destacando a capacidade do sistema em extrair informações relevantes.

Palavras-chave: Big Data. Apache Kafka. Apache Spark. Apache Pinot.

Abstract

The volume of data generated by users has grown exponentially. However, many companies face challenges in capturing and analyzing this information, directly impacting the generation of value for their business. Therefore, this course completion work aims to present a way of performing data analysis within the scope of Big Data, using Apache Kafka software to consume the data and distribute it within the ecosystem, Apache Spark to perform data analysis and filtering, and Apache Pinot to store and enable future queries. For the analysis, the Twitter API was used, aiming at the variety of data and allowing trends to be analyzed. The results obtained demonstrate the feasibility and effectiveness of the proposed methodology. The developed system could handle data ingestion and processing, demonstrating the scalability and performance of Apache Kafka, Apache Spark, and Apache Pinot tools. Furthermore, the analyses provided valuable insights into Twitter data, highlighting the system's ability to extract relevant information.

Keywords: Big Data. Apache Kafka. Apache Spark. Apache Pinot.

Lista de ilustrações

Lista de tabelas

Lista de Siglas

API *Application Programming Interface*

CPU *Central Process Unit*

CSV Valores Separados por Vírgula

DAG *Directed Acyclic Graph*

HDFS *Hadoop Distributed File System*

IDC *International Data Corporation*

IoT *Internet of things*

ISR *Synchronous Replica Set*

JSON *JavaScript Object Notation*

JVM *Java Virtual Machine*

KRaft *Apache Raft*

ML *Machine Learning*

OLAP *Online Analytical Processing*

RPC *Remote Procedure Call*

SQL *Structured Query Language*

XML *Extensible Markup Language*

Sumário

Capítulo 1

Introdução

No contexto atual, a Internet está conectando milhares de dispositivos que produzem uma grande variedade de informações. Esses dados abrangem uma variedade de tipos, como localização do usuário, pesquisas de produtos, interesses esportivos, visitas a páginas da web, buscas por entretenimento como filmes e músicas, entre outros. Essas informações desempenham um papel crucial nas estratégias de tomada de decisão das organizações, influenciando a inovação de produtos, a oferta de serviços e, especialmente, as estratégias de marketing digital. Durante a pandemia, o marketing digital experimentou um avanço significativo, impulsionando as empresas a investirem na transformação digital de seus negócios para se adaptarem ao novo cenário.

A produção de informações tem tido um crescimento exponencial em todo o mundo, impulsionado por uma variedade de fontes de dados, como dispositivos de *Internet of things (IoT)* e redes sociais. Segundo um estudo publicado pela *International Data Corporation* (IDC) na "*Data Age 2025*", prevê-se que até 2025, o volume global de dados crescerá para 163 *zettabytes*, representando um aumento de 10 vezes em relação aos dados gerados em 2016. Esses dados consistem em informações únicas, provenientes de experiências de usuários e da coleta de dados de sensores. As organizações que detêm essas informações terão acesso a um vasto conjunto de oportunidades de negócios em um novo cenário. (REINSEL JOHN GANTZ, 2017).

Estratégia de armazenamento, processamento e análise de dados dos usuários é de extrema importância para que as organizações consigam antecipar e se adaptar às mudanças do mercado. Thomas Varas, vice-presidente da IDC, enfatiza que aplicações analíticas, software de inteligência artificial, ferramentas de integração e integridade de dados permanecem essenciais para as estratégias de busca por novas oportunidades e mitigação de riscos para as organizações (REINSEL JOHN GANTZ, 2017).

Gerenciar um extenso conjunto de dados apresenta desafios significativos para as organizações. Isso envolve uma série de questões, incluindo a organização de dados não estruturados, a garantia da consistência dos dados, a disponibilidade e acessibilidade dos mesmos, bem como a segurança das informações (PEHCEVSKI, 2018). Segundo Peh-

cevski (2018), os dados não estruturados podem compreender de 70% a 80% do total de dados nas empresas, sendo que a maioria desses dados é proveniente das redes sociais, que representam cerca de 80% do volume de dados em escala global (PEHCEVSKI, 2018).

Antes da chegada dos sistemas de s*treaming* de eventos, o processamento de dados era realizado em lotes periódicos. Isso implica que os dados primeiro devem ser armazenados e, em seguida, processados em intervalos definidos. Essa abordagem exige muito tempo para concluir a análise dos dados, o que dificulta realizar análises em tempo hábil. Por exemplo, uma empresa que deveria aguardar até o final do dia, semana ou mês para analisar os milhões de registros para depois gerar insights (Google Cloud, 2023).

À medida que entra-se na era do *Big Data*, *Data Science* e *Data Analytics*, as organizações estão cada vez mais exigindo análises de dados em curto prazo. Os dados desempenham um papel crucial nas decisões estratégicas e comerciais das organizações. Diante dessa necessidade, as empresas buscam desenvolver sistemas que permitam desenvolver análises das informações conforme a necessidade, o que tem levado ao surgimento de plataformas baseadas em *streaming* de eventos. Essas plataformas processam os dados de entrada enquanto estão em execução no servidor, cálculos ultrarrápidos e contínuos dos dados de *streaming* de alta velocidade. Além disso, empregam um mecanismo de consulta contínua que gera alertas e ações simultaneamente, bem como visualizações ao vivo definidas pelo usuário (Google Cloud, 2023).

A Tabela 1 apresenta algumas estimativas sobre a geração de dados pelas principais plataformas globais. A velocidade de criação dos dados está diretamente relacionada ao volume, já que quanto mais rápida for a geração dos dados, maior será o volume de informações a serem processadas. (PEHCEVSKI, 2018).

Tabela 1 – Mídias Sociais e dados produzidos. Fonte Pehcevski (2018).

YouTube	Os usuários carregam 100 horas de novos vídeos por minuto.A cada mês, mais de 1 bilhão de usuários únicos acessam o YouTube. Mais de 6 bilhões de horas de vídeo são assistidas a cada mês, o que corresponde a quase uma hora para cada pessoa na Terra. Esse valor é 50% superior ao gerado no ano anterior.
Facebook	A cada minuto, 34.722 curtidas são registradas. 100 terabytes de dados são carregados diariamente. Atualmente, o site possui 1,4 bilhão de usuários. O site foi traduzido para 70 idiomas.
Twitter/X	O site tem mais de 645 milhões de usuários. O site gera 175 milhões de *tweets* por dia.
Foursquare	O site é usado por 45 milhões de pessoas em todo o mundo. Recebe mais de 5 bilhões de check-in por dia. A cada minuto, 571 novos sites são lançados.
Google	O site recebe mais de 2 milhões de consultas de pesquisa por minuto. Todos os dias, 25 *petabytes* são processados.
Apple	Aproximadamente 47.000 aplicativos são baixados por minuto.

Este trabalho visa apresentar uma solução para coleta de dados, processamento das informações, análise e armazenamento dos dados, de forma a auxiliar as organizações na tomada de decisões estratégicas.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é apresentar uma abordagem para processamento de dados. Para alcançar esse propósito, será desenvolvido e implementado um *pipeline* de dados abrangendo a ingestão, processamento e armazenamento.

1.1.2 Objetivos Específicos

1. Coletar dados da API do Twitter a partir da base de dados disponível no Kaggle.
2. Desenvolver um esquema Avro para manter a formatação dos dados e a serialização.
3. Implementar os Producer que importe os dados do Twitter, realiza as transformações necessárias para publicar no tópico do Kafka.
4. Criar uma aplicação Spark para consumir os dados do Kafka, realizar manipulações e publicar os resultados em um novo tópico do Kafka
5. Construir uma aplicação no Pinot que se integre ao Kafka, leia o tópico e publique os dados em uma tabela específica, permitindo acesso e consulta aos dados.

1.2 Organização da Monografia

O presente trabalho encontra-se organizado da seguinte forma: Capítulo 2 aborda o referencial teórico necessário para a compreensão de termos e das ferramentas utilizadas no trabalho. O Capítulo 3 apresenta a metodologia para o desenvolvimento do projeto, o Capítulo 4 aborda-se os experimentos realizados e os resultados obtidos e o Capítulo 5 traz a conclusão do trabalho com os objetivos alcançados e as sugestões de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo, são apresentadas as bases teóricas essenciais para a compreensão do presente trabalho. São discutidos os conceitos e requisitos fundamentais para a implementação de soluções de *Big Data*, incluindo definições de Sistemas Distribuídos baseados em Eventos, bem como uma análise das arquiteturas do Apache Kafka, Apache Spark, Apache Pinot, Apache Avro e o Docker.

2.1 Big Data

Big Data refere-se a conjuntos de dados extremamente grandes e complexos que desafiam as capacidades de processamento tradicionais de software (ORACLE, 2024b)

Pehcevski destaca três características essenciais de *Big Data*: o alto volume de dados, a natureza não estruturada desses dados e a velocidade com que são produzidos, capturados e processados rapidamente (PEHCEVSKI, 2018). Por sua vez, Taurion descreve o *Big Data* como a união de cinco elementos chave, conhecidos como os 5"V"do *Big Data*: volume, velocidade, variedade, veracidade e valor agregado (TAURION, 2015).

Há três classificações de dados: dados estruturados, não-estruturados e semiestruturados (REDES, 2024):

Dados estruturados são dados que seguem um formato predefinido e esperado. Embora possam ser provenientes de diversas fontes, o ponto em comum é que os campos têm uma configuração fixa, e são armazenados de forma organizada. Esse modelo de dados predefinido facilita a entrada, consulta e análise dos dados (ORACLE, 2024a).

Os dados não estruturados não seguem uma organização pré-determinada. Em vez de campos definidos em um formato específico, eles podem apresentar uma variedade de formatos e tamanhos. Embora geralmente incluam texto (como em um campo de texto aberto em um formulário), os dados não estruturados podem se manifestar em diversas formas, como imagens, áudio, vídeo, arquivos de documento e outros formatos de arquivo. A característica comum entre todos os dados não estruturados é a ausência

de uma estrutura definida. Os dados não estruturados geralmente são gerados de maneira não padronizada e podem ser desafiadores de analisar (ORACLE, 2024a).

Os dados semiestruturados apresentam uma combinação de elementos estruturados e não estruturados, caracterizando-se por um esquema flexível que não é tão rígido quanto o dos dados estruturados. Com o auxílio de metadados, que permitem aos usuários definir alguma estrutura ou hierarquia parcial, esses dados podem ser organizados até certo ponto, evitando a desorganização típica dos dados não estruturados. Os metadados incluem tags e outros marcadores, bem como formatos como *JavaScript Object Notation* (JSON), *Extensible Markup Language* (XML) ou Valores Separados por Vírgula (CSV), que delimitam os elementos e impõem uma hierarquia. No entanto, o tamanho dos elementos pode variar, e a ordem nem sempre é crucial (REDES, 2024).

2.1.1 Ciclo de vida dos dados

O ciclo de vida dos dados dentro no *Big Data* consiste nas seguintes etapas: coletas, filtragem e classificação, análise, armazenamento, compartilhamento e publicação, segurança e recuperação de dados (PEHCEVSKI, 2018).

Na etapa de coleta os dados as informações podem ser recebidas de diferentes fontes e de diferentes formatos como vídeo, imagem, textos, *logs*, *Structured Query Language* (SQL), sendo de forma estruturada ou não. O sistema de *Big Data* deve ser capaz de se conectar com sistemas externos e receber essas informações (PEHCEVSKI, 2018).

Durante a etapa de filtragem e classificação, é comum realizar um pré-processamento das informações para auxiliar na análise.

Na fase de análise dos dados, os objetivos principais são identificar relações entre as informações e desenvolver técnicas eficientes para a mineração de dados, visando previsões futuras. Alguns métodos comuns nesta etapa incluem algoritmos de mineração de dados, análise de *clusters*, análise estatística, análise de correlações e análise de regressão. Esta etapa apresenta desafios significativos, como a seleção da melhor técnica de análise, a complexidade dos dados, a escalabilidade dos algoritmos e a heterogeneidade dos dados, todos agravados pela alta velocidade de geração dos dados, o que pode comprometer a precisão dos algoritmos (PEHCEVSKI, 2018).

No estágio de armazenamento, é essencial avaliar a melhor estrutura para manter as informações de forma confiável, acessível e disponível. A estrutura de armazenamento deve oferecer espaço confiável e uma interface de acesso adequada (PEHCEVSKI, 2018).

Na fase de compartilhamento e publicação, os dados são distribuídos para as aplicações consumidoras (PEHCEVSKI, 2018).

A segurança dos dados é crucial em todas as etapas do processo. Todos os dados gerados devem ser mantidos de forma segura para garantir a privacidade, confidencialidade, integridade e governança das informações (PEHCEVSKI, 2018).

Por fim, na fase de recuperação dos dados, é possível utilizar as informações armazenadas para agregar valor e gerar novos dados mais completos (PEHCEVSKI, 2018).

2.1.2 Requisitos Arquitetônicos e de Infraestrutura

Conforme apresentado na sessão anterior relacionado ao ciclo ciclo de vida dos dados no *Big Data* proposto por (PEHCEVSKI, 2018), o ambiente para conseguir suportar *Big Data* deve conter os seguintes requisitos:

- O sistema deve ser capaz de lidar com dados sem um modelo fixo, permitindo o processamento de dados complexos, independentemente de suas características.
- Deve possuir escalabilidade e alto desempenho para coletar e processar dados em tempo real ou em lotes de forma eficiente.
- É necessário suportar o particionamento de dados devido ao grande volume de informações.
- Os dados devem ser replicados e compartilhados entre os nós do sistema para garantir a tolerância a falhas, permitir o processamento em várias etapas e o particionamento multiplataforma.

Considerando essas exigências, este trabalho propõe uma arquitetura que atenda aos requisitos de um sistema *Big Data* e gere informações úteis para as organizações. Os próximos capítulos, apresentará o ecossistema do Apache Kafka, Apache Spark e Apache Pinot. No entanto, antes de abordar o Apache Kafka, é necessário compreender o conceito de sistemas distribuídos baseados em eventos.

2.2 Sistemas Distribuídos Baseados em Eventos

Um evento é qualquer mudança de estado do software ou do hardware do sistema (RedHat, 2019). Esta mudança pode ser gerada internamente ou externamente do sistema. Por exemplo, em um sistema de compra *on-line* há varias etapas para que uma compra seja concluída, se observar apenas a etapa de inserir um item no carrinho, tem a ação do cliente selecionar um item, o sistema internamente precisa separar por determinado prazo o item selecionado, para garantir a compra do cliente (RedHat, 2019).

Uma mensagem representa a notificação de um evento, realizada para alertar uma outra parte sobre a ocorrência desse evento (RedHat, 2019). Um sistema distribuído baseado em eventos é uma forma de comunicação indireta, também conhecida como sistema de publicação e subscrição, onde dois grupos distintos interagem: um grupo público de eventos e outro grupo assinado para receber notificações sobre eventos específicos (COULOURIS et al., 2013). Retornando ao exemplo do de compra *on-line*, quando o evento de adição

de um item ao carrinho é acionado, a parte responsável pelo controle de estoque precisa ser notificado desse evento para que possa separar o item até a conclusão da compra ou por um determinado período de tempo.

Os sistemas baseados em eventos apresentam duas características principais: a heterogeneidade do ecossistema e a notificação assíncrona dos eventos (COULOURIS et al., 2013). A heterogeneidade refere-se ao fato de que a fonte que publica o evento é diferente do sistema que consome os dados do evento. Já na notificação assíncrona, os publicadores e assinantes estão desacoplados, o que significa que um evento pode ser publicado em um momento em que o assinante não está ativo, e pode haver vários assinantes para o mesmo evento. O sistema baseado em eventos precisa ser capaz de integrar diferentes aplicações e garantir a entrega das mensagens de novos eventos aos assinantes (COULOURIS et al., 2013).

2.3 Apache Kafka

O Apache Kafka é uma plataforma de *streaming* distribuída de código aberto amplamente empregada para o processamento de fluxos de dados, a criação de pipelines de dados em tempo real e a integração de dados em grande escala.(FOUNDATION, 2023a). O Kafka permitiu a criação de uma nova geração de aplicativos distribuídos com habilidade de ampliar para processar bilhões de eventos por minuto (FOUNDATION, 2023a).

Foi desenvolvido inicialmente pela empresa LinkedIn em 2008 e posteriormente, em 2011, foi doado para a Apache Software Foundation, tornando-se um projeto de código aberto. O Kafka foi concebido originalmente utilizando a linguagem Scala para lidar com a crescente demanda de fluxo de dados na plataforma do LinkedIn. Em 2014, membros da equipe original do LinkedIn se envolveram no desenvolvimento do Kafka fundaram a empresa Confluent, especializada em soluções e serviços relacionados ao Apache Kafka (FOUNDATION, 2023a).

Inicialmente concebido como um sistema de fila de mensagens, o Kafka evoluiu rapidamente para se tornar uma plataforma completa de *streaming* de eventos. Atualmente, é capaz de lidar com mais de 1 milhão de mensagens por segundo, ou trilhões de mensagens por dia (DEVELOPER, 2023).

Atualmente, o Kafka é adotado por mais de 80% das empresas envolvidas na Fortune 100 (DEVELOPER, 2023). Além disso, diversas outras empresas renomadas, como Netflix, Itaú, Via Varejo, PagSeguro, Uber, Twitter, Airbnb, LinkedIn, CloudFlare, Spotify, Yahoo! , Pinterest, Tumblr, entre outros, também fazem uso do Kafka (FOUNDATION, 2023a). Essa ampla adoção abrange diferentes setores e uma variedade de casos de uso, tanto em escala grande quanto pequena (FOUNDATION, 2023a). O Kafka emergiu como a tecnologia mais utilizada por desenvolvedores e arquitetos na criação de aplicativos escaláveis e em tempo real para *streaming* de dados (FOUNDATION, 2023a). Embora existam

outras tecnologias disponíveis no mercado que oferecem funcionalidades semelhantes, o Kafka se destaca devido a diversas questões, como (DEVELOPER, 2023):

1. Alta Escalabilidade

   O Kafka opera em *clusters*, oferecendo uma oferta que permite expandir ou contrair o sistema conforme a demanda de armazenamento e processamento de dados.

2. Baixa Latência

   O tempo de entrega de grandes volumes de dados é especificamente baixo, atingindo uma latência de apenas 2 milissegundos.

3. Armazenamento Permanente

   O Kafka viabiliza o armazenamento de fluxos de dados em *clusters* distribuídos, garantindo durabilidade, confiabilidade e tolerância a falhas.

4. Alta Disponibilidade

   C*lusters* são empregados para conferir ao Kafka uma alta disponibilidade e tolerância a falhas, garantindo a segurança dos dados sem risco de perdas.

2.3.1 Arquitetura

O Apache Kafka possui os seguintes componentes em sua arquitetura: Kafka Connect, Kafka Broker, Kafka Streams, Kafka Raft (PAULA, 2022). A figura 1 apresenta o ecossistema do Apache Kafka.

Antes de definir cada componente do Apache Kafka precisasse entender alguns termos importantes do ecossistema (GARG, 2013).

1. Produtores – são os responsáveis por gerar as mensagens para o ecossistema do Kafka.

2. Consumidores – são os elementos que conectam no Kafka e extraem as mensagens.

3. Tópicos – são as mensagens que o Kafka recebe, ele agrupa conforme sua seleção e as organiza em pares valor-chave sequencialmente.

4. Partição – dentro de um tópico pode haver varias camadas de partição de uma mensagem. A partição permite a elasticidade, tolerância à falha e a escabilidade do Apache Kafka.

5. Replicas – é semelhante à partição, porém a replica é de todo o *cluster* garantindo a redundância e a disponibilidade dos dados caso algum componente do *cluster* falhe.

6. *Log file* – cada tópico possui um registro no formato de log, de forma estruturada e em sequencia.

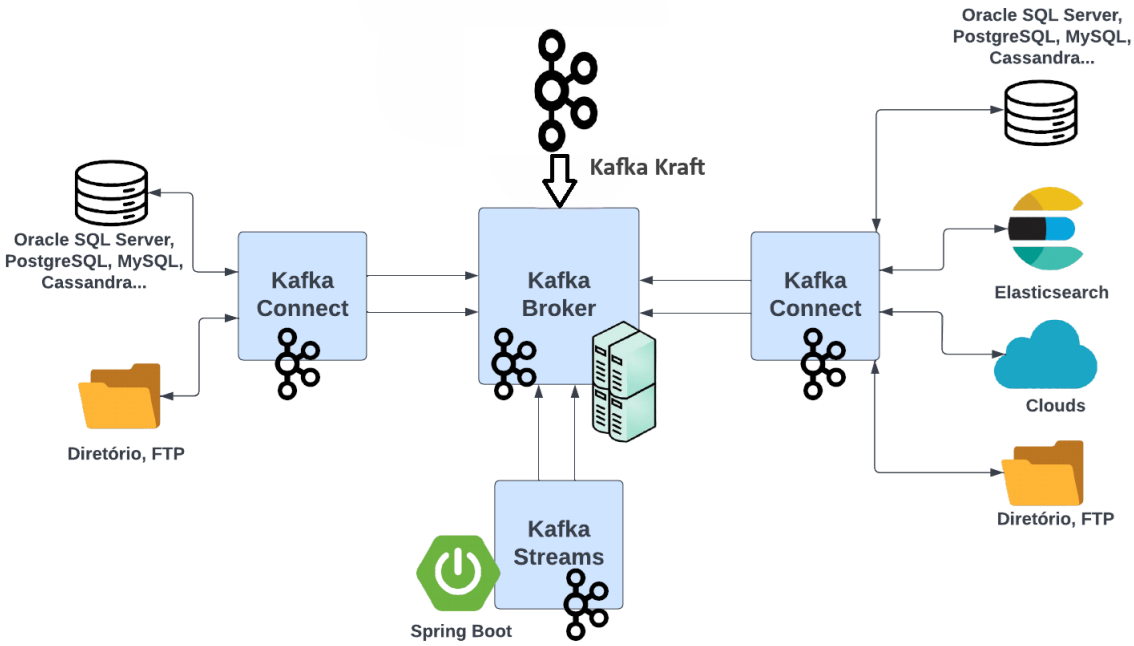

Figura 1 – Ecossistema do Apache Kafka

Fonte: (FOUNDATION, 2023a)

7. Segmentos - estão localizados dentro das partições e organizam as informações contidas nos arquivos de log dessa partição. Cada tópico possui suas próprias partições e segmentações. A segmentação é usada para gerenciar a ordenação dos dados no arquivo de log, bem como o tempo de armazenamento desses dados.

2.3.1.1 *Kafka Connect*

O *Kafka Connect* desempenha o papel de receber mensagens enviadas pelos produtores de eventos e de conectar sistemas entre si (DEVELOPER, 2023). É utilizado pelos consumidores de eventos para retirar mensagens do Kafka, por exemplo, para sistemas que monitoram logs e realizam análises, permitindo o processamento de várias informações de forma paralela (SHAPIRA et al., 2021). Na figura 25 é apresentado os componentes do *Kafka Connect* e como eles interagem.

O foco principal do *Kafka Connect* é receber e enviar fluxos de dados, simplificando a comunicação entre sistemas. Através do *Confluent Cloud,* são disponibilizados conectores Kafka pré-construídos e completamente gerenciados, o que torna a conexão instantânea com fontes e coletores de dados muito mais fácil. (DEVELOPER, 2023)(SHAPIRA et al., 2021).

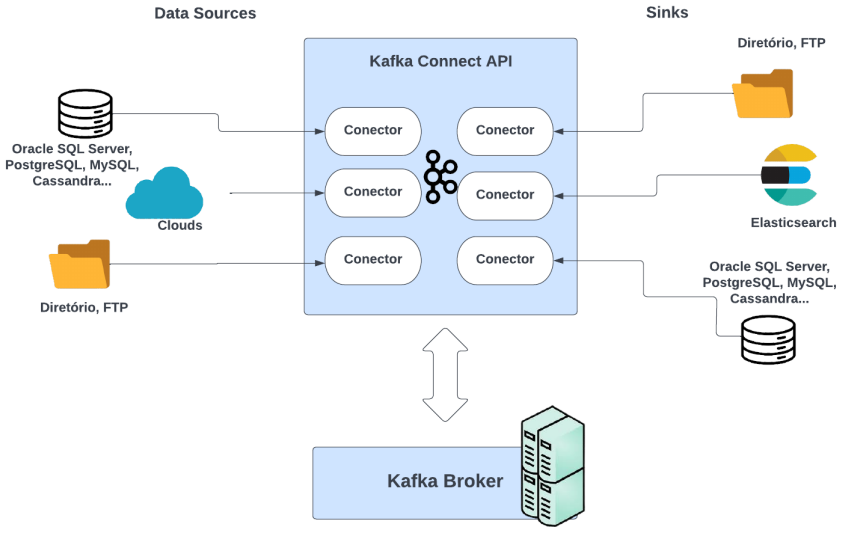

Figura 2 – Arquitetura do Kafka Connect

Fonte: (FOUNDATION, 2023a)

2.3.1.2 *Kafka Broker*

O *Kafka Broker* desempenha o papel de armazenar informações e controlar o acesso aos tópicos no ecossistema do Kafka. Ele recebe os tópicos enviados pelo *Kafka Connect* e gerencia o processamento para garantir que os consumidores desses tópicos recebam as mensagens corretamente (GARG, 2013).

O *Kafka Broker* é organizado em *cluster* para possibilitar que o sistema execute várias tarefas de forma paralela e assíncrona. Essa arquitetura fortalece a tolerância a falhas, graças à redundância e replicação de dados, o que assegura operações contínuas e sem perda de dados. Na figura 3 é apresentado a arquitetura do *Kafka Broker* e como eles comunicam.

2.3.1.3 *Kafka Streams*

O *Kafka Streams* é o componente que facilita o processamento de dados em tempo real. Ele possibilita a recepção de grandes volumes de dados, aplicação de transformações e regras de negócio, e o subsequente fluxo da aplicação. A *Kafka Streams* API é uma biblioteca (Java, Scala) que permite a construção de aplicações de fluxo de dados em cima do Kafka. Com o *Kafka Streams*, é possível desenvolver regras de negócio utilizando a linguagem Java juntamente com o *framework Spring Boot.*(SHAPIRA et al., 2021).

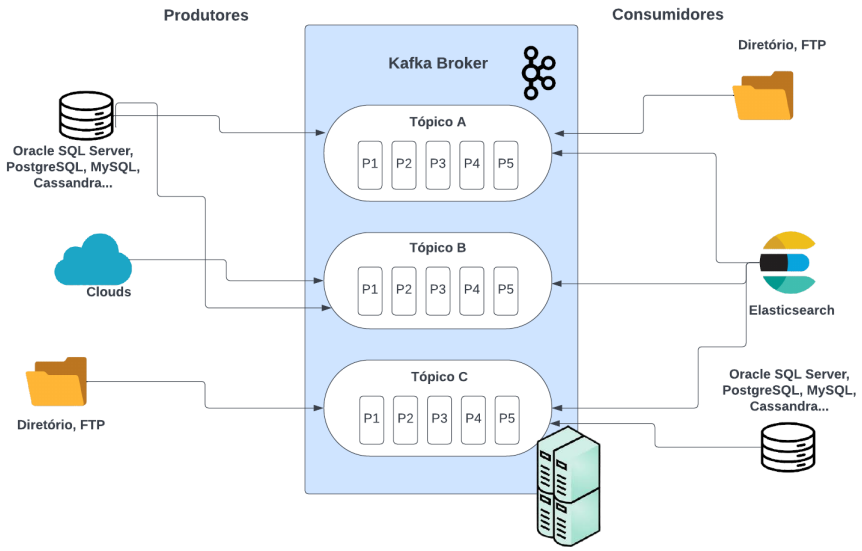

Figura 3 – Arquitetura do Kafka Broker

Fonte: (FOUNDATION, 2023a)

2.3.1.4 *Kafka Raft*

O *Apache Raft* (KRaft) é um protocolo de consenso introduzido no *KIP-500* com o objetivo de eliminar a dependência do *Apache ZooKeeper* para o gerenciamento de metadados. Essa mudança simplifica significativamente a arquitetura do Kafka, consolidando a responsabilidade pelos metadados dentro do próprio Kafka, em vez de dividi-la entre dois sistemas distintos: *ZooKeeper* e Kafka (DEVELOPER, 2024a).

O modo KRaft utiliza um novo serviço de controlador de quórum no Kafka, substituindo o controlador do *Zookeeper*, e faz uso de uma variante baseada em eventos do protocolo de consenso Raft. Essa abordagem proporciona uma solução mais integrada e simplificada para o gerenciamento de metadados no Kafka, melhorando sua escalabilidade e confiabilidade (DEVELOPER, 2024a).

Os nós do controlador KRaft formam um quórum Raft responsável pelo gerenciamento do log de metadados do Kafka. Este log registra todas as alterações nos metadados do *cluster*, incluindo informações como tópicos, partições, *Synchronous Replica Set* (ISR) e configurações (DEVELOPER, 2024b).

Por meio do protocolo de consenso Raft, os nós controladores asseguram a consistência e a eleição de líderes sem depender de sistemas externos. O líder do log de metadados, denominado controlador ativo, é encarregado de processar todas as requisições *Remote Procedure Call* (RPC) feitas pelos corretores. Enquanto isso, os controladores seguidores replicam os dados gravados pelo controlador ativo e permanecem como reserva ativa em caso de falha do controlador ativo (DEVELOPER, 2024b).

Através do conceito de *log* de metadados, os *brokers* utilizam *offsets* para rastrear os

metadados mais recentes armazenados nos controladores KRaft, o que resulta em uma disseminação mais eficiente de metadados e uma recuperação mais rápida em caso de falhas do controlador(DEVELOPER, 2024b).

2.4 Apache Spark

O Apache Spark é uma plataforma de análise de dados que oferece suporte ao processamento de grandes volumes de dados. Possui módulos integrados para SQL, *streaming*, aprendizado de máquina e processamento de gráficos. Uma das características mais marcantes do Spark é sua capacidade de computação em *cluster*, o que significa que pode distribuir tarefas de processamento entre vários nós de um *cluster* (DATABRICKS, 2024a).

Além disso, é tolerante a falhas, o que garante a continuidade das operações mesmo em caso de falhas de *hardware* ou *software*. O Spark também se destaca por sua capacidade de computação paralela, permitindo que múltiplas tarefas sejam executadas simultaneamente. Utiliza processamento em memória, o que aumenta significativamente a velocidade de processamento das aplicações em comparação com sistemas que dependem principalmente de acesso a disco (SANTOS, 2021).

O Spark estende o modelo MapReduce para usar com eficiência mais tipos de cálculos, incluindo consultas interativas e processamento de fluxo (DATABRICKS, 2024a).

MapReduce é uma estrutura distribuída, escrita em Java, integrante do ecossistema Apache Hadoop. Sua função é simplificar a complexidade da programação distribuída, oferecendo aos desenvolvedores duas etapas principais para implementação: o Mapear e o Reduzir (DATABRICKS, 2024b). Na fase de mapeamento, os dados são divididos em tarefas de processamento paralelas, permitindo a aplicação de transformações específicas a cada bloco de dados. Após a conclusão do mapeamento, entra em cena a etapa de redução, onde os dados previamente divididos são agregados. Normalmente, o *MapReduce* utiliza o *Hadoop Distributed File System* (HDFS) para entrada e saída de dados (DATABRICKS, 2024b).

O Apache Spark possibilita realizar o processamento de dados em lote ou em *streaming*, e há flexibilidade de escolha entre várias linguagens de programação, como Python, SQL, Scala, Java ou R (DATABRICKS, 2024a). Essa variedade de opções permite executar análises SQL ágeis e distribuídos para produção de painéis e relatórios. Além disso, o Spark facilita o desenvolvimento de projetos de ciência de dados em grande escala, uma vez que seu ecossistema permite a execução em *clusters*, o que possibilita análises de dados em *petabytes* sem comprometer a resolução. Por fim, o Spark suporta o treinamento de algoritmos de aprendizado de máquina para operar em *clusters*, ampliando ainda mais suas capacidades analíticas (DATABRICKS, 2024a).

2.4.1 Ecossistema do Apache Spark

O ecossistema do Spark possui os seguites componentes: *Spark SQL, Spark Streaming, MLib, GraphX e o Apache Spark Core*, conforme representado na figura 4.

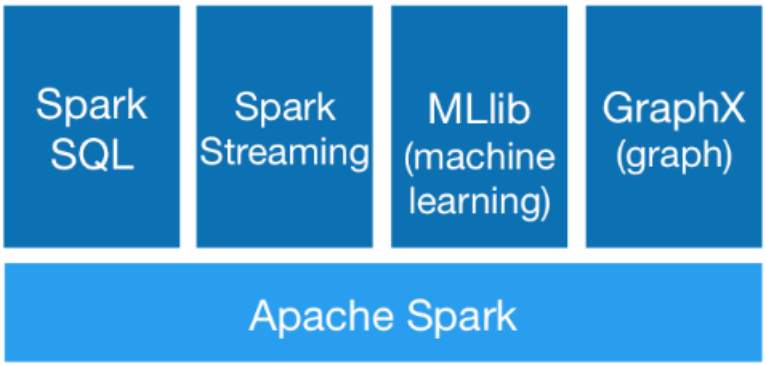

Figura 4 – Ecossistema do Apache Spark.

Fonte: Foundation (2018)

2.4.1.1 *Apache Spark Core*

É a base fundamental de execução subjacente da plataforma Spark, sobre como todas as outras funcionalidades são construídas. Esse mecanismo oferece as operações essenciais para o processamento, incluindo funções como mapear, reduzir, filtrar e coletar (SALLOUM et al., 2016). Além disso, possibilita a computação em memória e o acesso a conjuntos de dados armazenados em sistemas externos.

2.4.1.2 *Spark SQL*

Trata-se do componente que viabiliza a manipulação de dados estruturados, possibilitando consultas SQL ou o uso da *API DataFrame* através de Java, Scala, Python ou R. Este módulo, situado sobre o Spark Core, introduz uma nova abstração de dados conhecida como Esquema RDD , que oferece suporte para dados estruturados e semiestruturados (SALLOUM et al., 2016).

O *Spark SQL* oferece a capacidade de acessar várias fontes de dados distintas e realizar a tarefa entre elas. Incorpora um otimizador baseado em custos, armazenamento colunar e geração de código para acelerar as consultas. Além disso, é escalável para milhares de nós e consulta de longa duração, utilizando o mecanismo Spark, que garante total tolerância a falhas durante a execução das consultas (SALLOUM et al., 2016).

2.4.1.3 Spark Streaming

Trata-se do módulo responsável por permitir a aquisição de dados em lote ou em tempo real. O Spark Streaming disponibiliza as mesmas APIs estruturadas (*DataFrames*

e Datasets) do Spark, eliminando a necessidade de desenvolver ou manter duas tecnologias distintas para processamento em lote e *streaming*. Além disso, suas APIs unificadas simplificam a migração de trabalhos em lotes existentes no Spark para trabalhos de *streaming* (SALLOUM et al., 2016).

2.4.1.4 MLlib

Este é o módulo escalável de aprendizado de máquina do Spark, equipado com ferramentas que tornam o *Machine Learning* (ML) prático, escalável e de fácil utilização. Apresenta algoritmos de alto desempenho, que são até 100 vezes mais rápidos que os do *MapReduce* (SALLOUM et al., 2016). A MLlib inclui algoritmos de alta qualidade que se beneficiam da iteração e podem produzir resultados superiores às abordagens geradas pelo *MapReduce* (SALLOUM et al., 2016).

MLlib abrange uma ampla gama de algoritmos de aprendizado, como classificação, correção, recomendação e agrupamento. Além disso, oferece fluxos de trabalho e uma variedade de detalhes, incluindo transformações de recursos, construção de *pipelines* de ML, avaliação de modelos, álgebra linear distribuída e estatística (SANTOS, 2021).

2.4.1.5 *GraphX*

GraphX é uma API do Apache Spark dedicada à manipulação e computação paralela de gráficos. Consolidando etapas como remoção, transformação, carregamento, análise exploratória e computação iterativa em um único sistema, o GraphX oferece uma API altamente flexível, acompanhada de uma gama de algoritmos gráficos. Além disso, sua eficiência de desempenho rivaliza com os sistemas gráficos mais rápidos, preservando a flexibilidade, tolerância a falhas e facilidade de uso características do Spark (SALLOUM et al., 2016).

2.4.2 Arquitetura de uma aplicação Spark

A figura 5 ilustra a arquitetura de uma aplicação Spark (FOUNDATION, 2018)(SALLOUM et al., 2016).

2.4.2.1 *Driver Program*

O *Driver Program* é a aplicação principal responsável por coordenar a criação e execução do processamento definido pelo programador. Ele abriga o *SparkContext*, que coordena as operações no *cluster* e pode se conectar a vários *Cluster Manager*. O *SparkContext* serve como o ponto de entrada primário para as funcionalidades do Spark, através do qual o *Driver Program* interage com o Spark (SALLOUM et al., 2016).

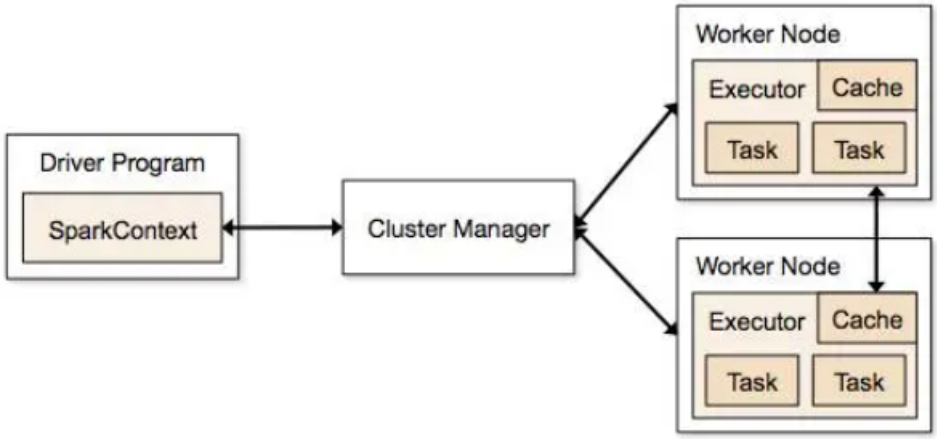

Figura 5 – Arquitetura do Apache Spark.

Fonte:Foundation (2018)

O *Driver Program* engloba vários outros componentes, como o *DAG Scheduler, o Task Scheduler, o Backend Scheduler e o Block Manager*. Todos esses são encarregados de traduzir o código desenvolvido pelo usuário em tarefas realizadas no *cluster*.

2.4.2.2 *Cluster Manager*

O *Cluster Manager* trata-se de um componente facultativo, necessário apenas quando o Spark é executado de forma distribuída. Sua função é gerenciar as máquinas que serão utilizadas no processo como workers (SALLOUM et al., 2016).

2.4.2.3 *Workers*

As máquinas que realizam as tarefas enviadas pelo *SparkContext* são conhecidas como *Worker Node*. Cada *Worker Node* disponibiliza recursos de *Central Process Unit* (CPU), memória e armazenamento para uma aplicação Spark. Um Executor, por sua vez, é um processo *Java Virtual Machine* (JVM) criado pelo Spark em cada *Worker* para executar essa aplicação.

Um trabalho no contexto do Spark é um algoritmo de processamento de dados realizado em um *cluster* para fornecer resultados ao *Driver Program*. Uma aplicação Spark pode iniciar vários trabalhos, sendo cada um deles dividido em um *Directed Acyclic Graph* (DAG) de projetos, onde cada estágio representa uma coleção de tarefas. Cada tarefa constitui a menor unidade de trabalho enviada pelo Spark para um executor. Caso o Spark seja executado de forma local, a máquina desempenha simultaneamente os papéis de *Driver Program* e *Worker* (SALLOUM et al., 2016).

2.5 Apache Pinot

O Apache Pinot é um sistema distribuído de armazenamento de dados para *Online Analytical Processing* (OLAP) em tempo real. Ele foi projetado para permitir a ingestão e consulta instantânea de dados de fontes de streaming ou lotes (FOUNDATION, 2023b).

O Pinot oferece análise com baixa latência, mesmo em ambientes de alta taxa de transferência. Ele armazena os dados em formato de coluna, implementando diversas técnicas inteligentes de indexação e pré-agregação. Além disso, o Pinot é escalável, adaptando-se às necessidades do sistema, garantindo um desempenho consistente em relação ao tamanho do *cluster* e um limite de consultas por segundo predefinido (FOUNDATION, 2023b).

Uma das características proeminentes do Pinot é sua capacidade de fornecer análises em tempo real diretamente aos usuários finais da aplicação. Portanto, o sistema deve manter-se constantemente atualizado para garantir a entrega de dados consistentes e em tempo real aos usuários (FOUNDATION, 2023b).

Desenvolvido inicialmente no LinkedIn, o Pinot foi concebido para suportar aplicativos de análise interativa em tempo real, como monitoramento de visualizações de perfil, análise de empresas, insights sobre talentos, entre outros. Posteriormente, em 2018, foi doado à *Apache Software Foundation* (FOUNDATION, 2023b).

2.5.1 Arquitetura

O Apache Pinot é dividido em quatro componentes principais, conforme apresentado na figura 6 (FOUNDATION, 2023b).

2.5.1.1 *Controller*

Como supervisor do estado global e da integridade do *cluster*, o *Zookeeper* é essencial para a harmonia do sistema. Dada sua função tanto como participante quanto observador no *Helix*, ele guia o estado dos demais componentes. É comum que o *Zookeeper* seja o primeiro componente iniciado após o início do *Helix*. (FOUNDATION, 2023b).

O Apache Helix é uma estrutura de gerenciamento de *cluster* genérica projetada para gerenciar partições e réplicas em sistemas distribuídos, incluindo a gestão de todos os servidores e agentes do Pinot. Utilizando o *Zookeeper* para manter o estado do *cluster*, cada componente em um *cluster* do Pinot é inicializado com um endereço do *Zookeeper* como parâmetro. Os diversos componentes distribuídos em um *cluster* do Pinot monitoraram as notificações do *Zookeeper* e enviaram atualizações por meio de seu agente integrado pelo Helix (FOUNDATION, 2023b).

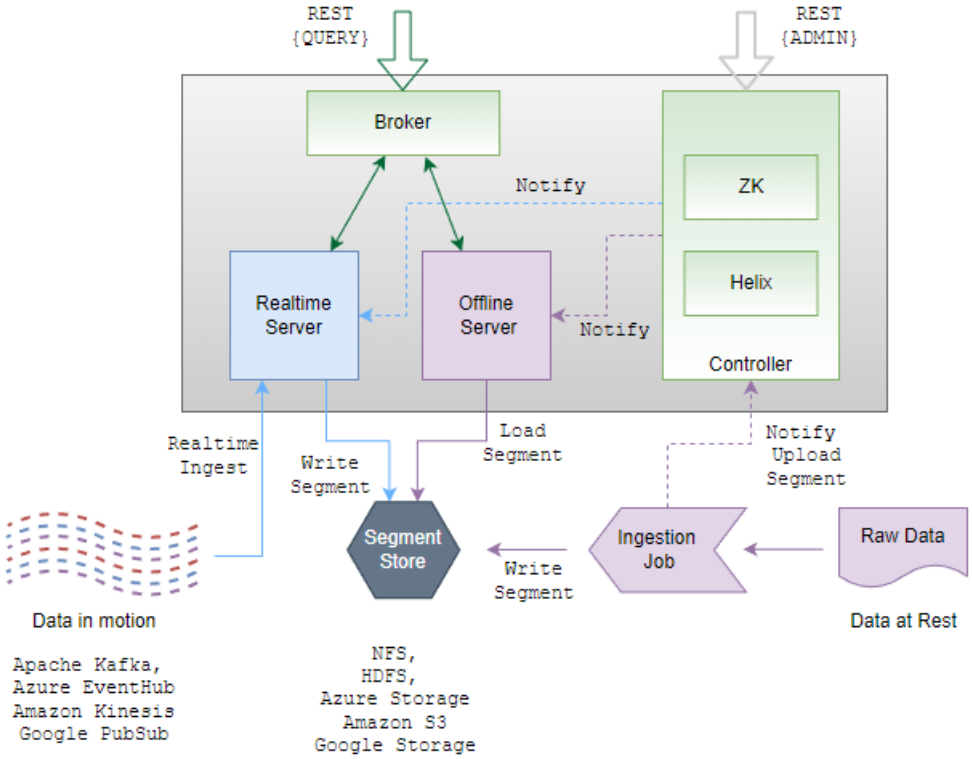

Figura 6 – Arquitetura do Apache Pinot.

Fonte:Foundation (2018)

2.5.1.2 Apache Zookeeper

É um serviço centralizado responsável por manter informações de configuração, considerar nomes, oferecer distribuição e fornecer serviços de agrupamento. (FOUNDATION, 2023b).

2.5.1.3 *Broker*

Os *brokers* são responsáveis pelo processamento das consultas no Pinot. Eles recebem as consultas dos clientes e as direcionam para os servidores protegidos. Posteriormente, eles coletaram os resultados desses servidores e os consolidaram em uma única resposta, que é então enviada de volta ao cliente. (FOUNDATION, 2023b).

2.5.1.4 Server

Os servidores abrigam os segmentos de dados e respondem às consultas relacionadas aos dados que armazenam. Existem dois tipos diferentes de servidores.: (FOUNDATION, 2023b):

1. *Offline*: Os servidores *offline* têm a responsabilidade de baixar segmentos do armazenamento de segmentos, hospedá-los e responder a consultas. Quando um novo

segmento é carregado no driver, ele decide quais servidores e quantas réplicas irão hospedar o novo segmento e a notificação para baixar o segmento de armazenamento. Ao receber essa notificação, os servidores fazem o *download* do arquivo de segmento e carregam no servidor para atender às consultas pertinentes.

2. *Realtime*: Os servidores em tempo real realizam a ingestão direta de um fluxo em tempo real (como Kafka, *EventHubs*). De forma periódica, eles geram segmentos dos dados ingeridos na memória, obedecendo a certos limites definidos. Esses segmentos são posteriormente persistentes no armazenamento de segmentos.

2.5.1.5 Minion

Este componente é um ator passivo que utiliza o *Helix Task Framework* para assumir tarefas computacionalmente intensivas de outros componentes. Ele pode ser integrado a um *cluster* Pinot existente e, então, carregar-se de tarefas designadas pelo controlador. Tarefas personalizadas podem ser adicionadas ao sistema por meio de anotações no *cluster*. (FOUNDATION, 2023b).

2.6 Apache Avro

O Apache Avro é um sistema de serialização de dados desenvolvido pelo projeto Apache. Ele foi projetado para ser compacto, rápido e eficiente em termos de processamento, tornando-o ideal para comunicação entre sistemas distribuídos e armazenamento de dados. Neste projeto usou-se o formato avro como padrão para troca de mensagens nos tópicos do *Producer* e do *Consumer* no Kafka (FOUNDATION, 2024)

Algumas vantagens em utilizar o formato Avro são:

1. Estrutura de Dados: O Avro define uma estrutura de dados em formato JSON chamada Esquema Avro. Este esquema descreve os tipos de dados permitidos, como registros, *enums, arrays* e mapas, bem como os campos ou elementos que compõem esses tipos. Esta estrutura de dados fornece uma maneira de garantir a compatibilidade entre diferentes sistemas que trocam dados no formato Avro.

2. Compactação e Eficiência: O Avro é projetado para ser compacto em tamanho e eficiente no âmbito de processamento. Ele usa técnicas de compactação, como codificação binária e compactação de dados, para reduzir o tamanho dos dados serializados. Isso não apenas economiza largura de banda durante a transmissão de dados pela rede, mas também reduz os requisitos de armazenamento quando os dados são gravados em disco.

3. Resolução de Esquema Dinâmico: Uma das características únicas do Avro é sua capacidade de suportar a resolução de esquemas dinâmicos. Isso significa que os

esquemas Avro podem evoluir ao longo do tempo sem quebrar a compatibilidade com versões anteriores. Isso é útil em ambientes distribuídos, nos quais diferentes partes do sistema podem ter versões diferentes do esquema.

4. Suporte a Múltiplas Linguagens de Programação: O Avro possui bibliotecas e APIs disponíveis para várias linguagens de programação, incluindo Java, Python, C#, Ruby e muitas outras. Isso permite que o Avro seja utilizado em uma variedade de ambientes e integre facilmente com diferentes tecnologias.

5. Integração com Ecossistema Hadoop: O Avro é amplamente utilizado no ecossistema Hadoop para armazenar dados em HDFS e processá-los usando *frameworks* como MapReduce, Hive e Spark. Sua eficiência e compatibilidade com várias linguagens de programação tornam-no uma escolha popular para trabalhar com grandes conjuntos de dados em Hadoop.

2.7 Docker

Docker é uma plataforma de código aberto que simplifica o desenvolvimento, implantação e execução de aplicativos dentro de contêineres. Ele oferece uma abordagem eficiente para empacotar, distribuir e executar software, garantindo consistência e portabilidade em diferentes ambientes (INC, 2024).

Os contêineres são uma tecnologia de virtualização na camada do aplicativo que encapsula tanto o código quanto as dependências de um aplicativo. Vários contêineres podem ser executados simultaneamente na mesma máquina e compartilhar o *kernel* do sistema operacional com outros contêineres. Cada contêiner é executado como um processo isolado no espaço do usuário. Uma das principais vantagens dos contêineres é sua eficiência em termos de recursos: ocupam menos espaço do que as máquinas virtuais, podem suportar mais aplicativos e excluir menos recursos de máquina virtual e sistema operacional para operar (INC, 2024).

Capítulo 3

Metodologia

O presente capítulo detalhará a arquitetura do sistema desenvolvido. Explicando a integração e comunicação entre os componentes propostos.

3.1 Desenvolvimento

3.1.1 Base de Dados

A escolha da base de dados do Twitter para este trabalho se deve à sua vasta variedade e diversidade de dados, os quais são gerados de forma contínua. Como mencionado na introdução, o Twitter conta com mais de 645 milhões de usuários, e a plataforma gera mais de 175 milhões de tweets diariamente (PEHCEVSKI, 2018).

Utilizou-se duas base de dados disponibilizada no Kaggle "*Twitter-Dataset*" e a "*Tweets*", os conjuntos de dados estão em arquivo no formato CSV (KAGGLE, 2024b) e (KAGGLE, 2024a). O Kaggle é uma plataforma *online* que oferece uma variedade de recursos relacionados à ciência de dados e aprendizado de máquina.

A base "*Twitter-Dataset*" possui as seguintes informações *Tweet_ID, Username, Text, Retweets, Likes e Timestamp*. O *Tweet_ID* é a identificação única do *tweet* e está no formato numérico, o Username é o nome de usuário de quem enviou o *tweet* e está no formato de texto, o *Text* é a mensagem enviada está no formato de texto, o *Retweets* é a quantidade de retuítes ou compartilhamento de mensagens e está no formato numérico, o Likes é a quantidade de *likes* que a mensagem recebeu de outros usuários do Twitter está no formato númerico e o *Timestamp* é a data e hora que a mensagem foi enviada e está no formato de data e hora.

A base "*Tweets*" possui as seguintes informações *author, content, country, date_time, id, language, latitude, longitude, number_of_likes e number_of_shares*. O *id* é a identificação única do *tweet* e está no formato texto, o *author* é o nome de usuário de quem enviou o *tweet* e está no formato de texto, o *content* é a mensagem enviada está no formato de texto, o *number_of_share*s é a quantidade de retuítes e está no formato numérico, o

number_of_likes é a quantidade de *likes* que a mensagem recebeu de outros usuários do Twitter está no formato númerico, o *date_time* é a data e hora que a mensagem foi enviada e está no formato de data e hora, *language* identifica o idioma da mensagem, o *country, latitude e longitude* estão vazios.

3.1.2 Arquitetura

Os dados serão produzidos em tópicos no Apache Kafka, onde qualquer atualização ou geração de novos dados resultará na inserção das informações no final do tópico. O Apache Spark será responsável por coletar todos os dados e realizar o processo de limpeza, removendo dados irrelevantes para o projeto. Após essas alterações, os dados serão novamente produzidos em outro tópico no Kafka, pronto para serem consumidos e armazenados pelo Apache Pinot.

Uma vez inseridos no Apache Pinot, os dados estarão prontamente disponíveis para manipulação através da interface gráfica do Pinot. A figura 7 apresenta os componentes utilizados e o fluxo dos dados.

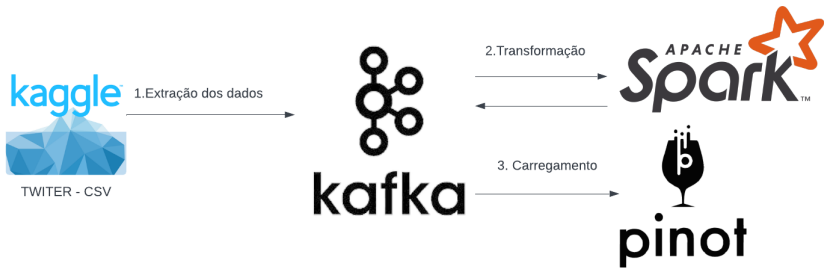

Figura 7 – Arquitetura do Projeto

Fonte: Autoria Própria

3.1.2.1 Kafka

Para o desenvolvimento do sistema Kafka, utilizou-se a paltaforma do Docker e foi configurado um arquivo *docker-compose* com as es*pecificações do ecossistema do Kafka, empregando a imagem* apache/kafka:3.7.0.

Para garantir organização, padronização e serialização dos arquivos de entrada, utilizou-se o esquema da Apache Avro para tabular os dados. Foi desenvolvido dois esquemas no formato JSON, um para cada produtor de acordo com os dados de entrada.

Desenvolveu-se dois Kafka Producers utilizando a linguagem de programação Python em conjunto com a biblioteca *confluent_kafka*. Os produtores são configurados para ler o esquema declarado e, em seguida, iniciar a leitura do arquivo CSV utilizando a biblioteca

Pandas. Os dados são serializados em Avro e enviados para um tópico do Kafka. O fluxo dos dados é apresentado na Figura 8.

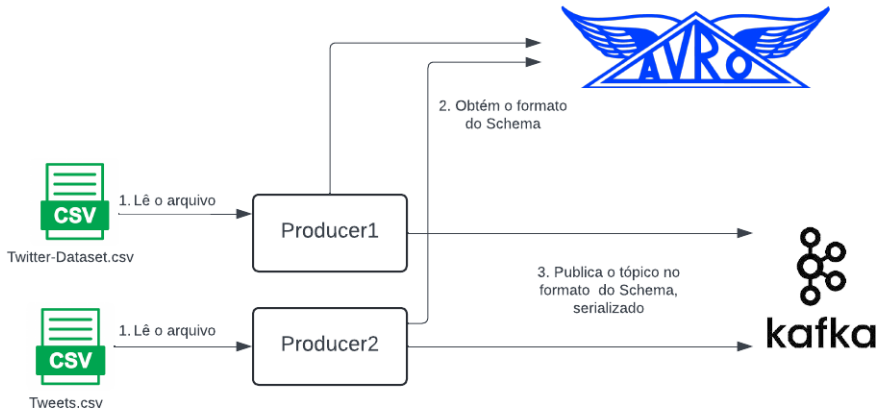

Figura 8 – Fluxo entre o Apache Kafka e Apache Avro

Fonte: Autoria Própria

3.1.2.2 Spark

Para iniciar o Spark, foi instalado a versão 3.5.1 e o Java JDK versão 21. Utilizando a biblioteca pySpark, criou-se uma sessão Spark para ler os dados provenientes dos dois tópicos do Kafka. Optou-se por realizar a leitura dos tópicos em lotes, criando assim dois DataFrames e renomeando suas colunas.

No processo, separou-se o atributo "*timestamp*"em quatro novas colunas: *dateDay, dateMonth, dateYear e hours*, visando facilitar consultas futuras. Também padronizou o formato da data e hora para manter consistência nos dados.

Figura 9 – Fluxo entre o Apache Kafka, Apache Avro e Apache Spark

Fonte: Autoria Própria

3.1.2.3 Pinot

Para configurar o Apache Pinot, também utilizou-se o Docker para iniciar o ecossistema usando a imagem *apachepinot/pinot:1.0.0*. Cria-se um arquivo *docker-compose* contendo todas as configurações necessárias. Na figura 10 é apresentado o fluxo de comunicação entre o Kafka e Pinot.

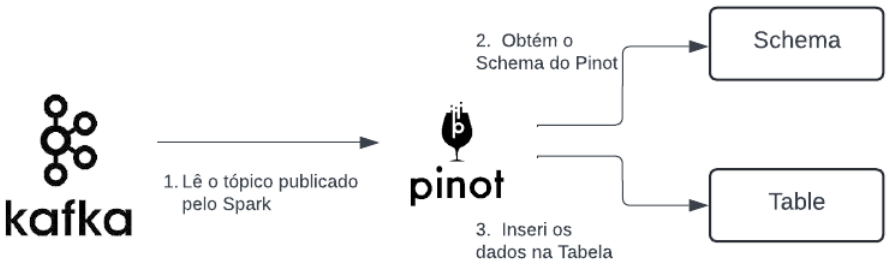

Figura 10 – Fluxo entre o Apache Kafka e Apache Pinot

Fonte: Autoria Própria

No Pinot foi essencial definir o esquema e a tabela; utilizou-se a interface gráfica do Pinot para realizar as configurações. Após a definição da tabela*realtime*, o Pinot é capaz de acessar o tópico do Kafka e, sempre que novas mensagens são recebidas, ele as insere automaticamente na tabela. Dessa forma, os dados ficam prontamente disponíveis para consultas sempre que necessário, com atualizações realizadas de forma automática.

O Pinot oferece uma interface que permite executar consultas *SQL* conforme necessário, a figura 11 apresenta a interface gráfica do Pinot.

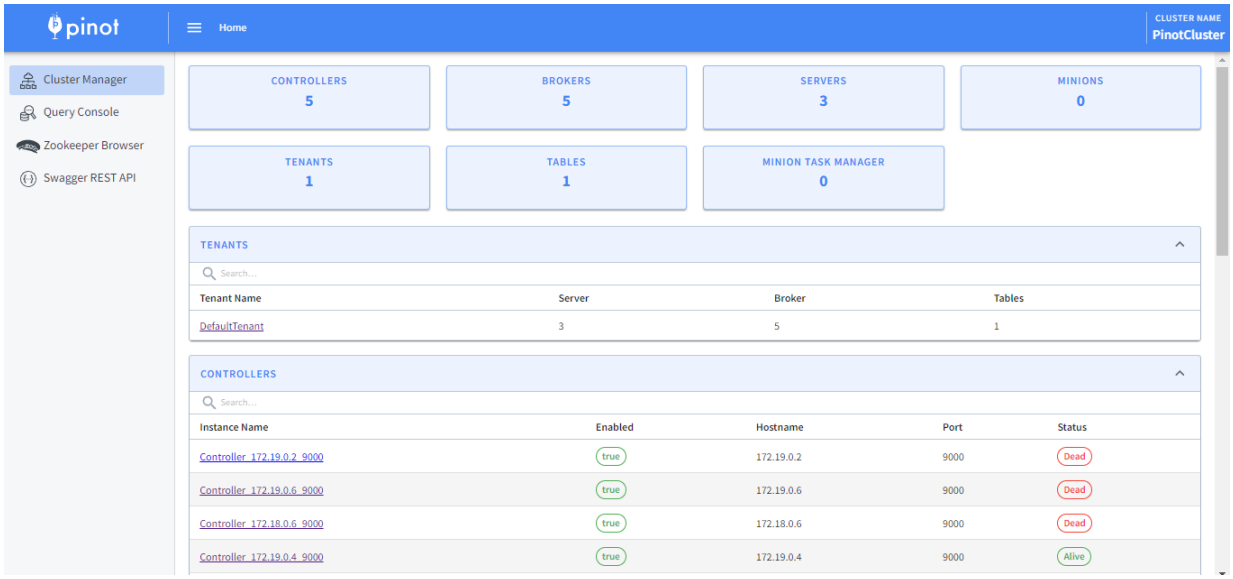

Figura 11 – Interface gráfica do Pinot

Fonte: Autoria Própria

Capítulo 4

Experimentos e Análise dos Resultados

Foram montados dois *cluster*s do Kafka, um contendo apenas 1 *broker* e outro com 3 *brokers*. Realizou-se quatro testes em cada *cluster* e, em seguida, comparou-se os tempos de execução de cada teste.

O projeto foi configurado localmente em uma máquina com as seguintes especificações: sistema operacional Windows 10 de 64 bits, processador *AMD Ryzen* 5 3500U com *Radeon Vega Mobile Gfx* (2.10 GHz) e 12,0GB de memória RAM.

O Kafka *Broker* foi configurado com os seguintes parâmetros: acks: all, linger.ms: 100, compression.type: gzip, enable.idempotence: true, max.in.flight.requests.per.connection: 5, request.required.acks: -1, retries: 5.

A base de dados "*twitter_dataset*"contém 10.000 registros, ocupando 2,56MB, enquanto "*tweets*"possui 52.543 registros, ocupando 7,51MB.

Para a integração entre o Kafka e o Spark foi utilizado o pacote spark-sql-kafka-0-10_2.12 versão 3.5.1 e para a integração entre o Apache Avro spark-avro_2.12 versão 3.5.1.

No Teste 1, mediu o tempo que o *Kafka Producer* 1 levou para ler o arquivo "*twitter_dataset*", serializá-lo no formato Avro e produzir no primeiro tópico.

No Teste 2, mediu o tempo que o *Kafka Producer* 2 levou para ler o arquivo "*tweets*", serializá-lo no formato Avro e produzir no primeiro tópico.

No Teste 3, foi medido o tempo que o Spark levou para deserializar e ler os tópicos produzidos pelos *Kafka Producers*, realizar as manipulações nos dados, serializá-los e produzir no tópico do Kafka que será consumido pelo Pinot.

No Teste 4, mediu o tempo em que os Kafka Producers 1 e 2 executaram simultaneamente.

4.1 Resultados

De acordo com o que foi descrito na Seção 3.1, apresenta-se a seguir os resultados obtidos utilizando o primeiro *cluster* do Kafka com um único *broker*.

O Teste 1 levou em média 1 minutos e 16 segundos para ser executado.

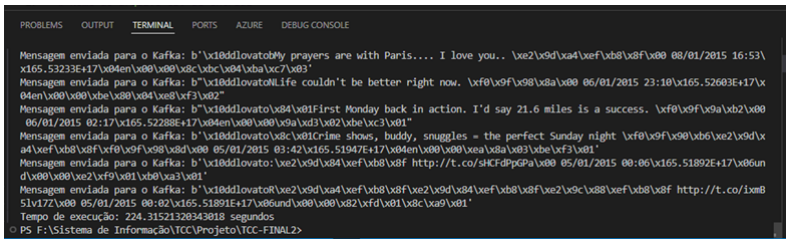

Figura 12 – 1º Testes realizado no primeiro *cluster*

O Teste 2 levou em média 4 minutos e 08 segundos.

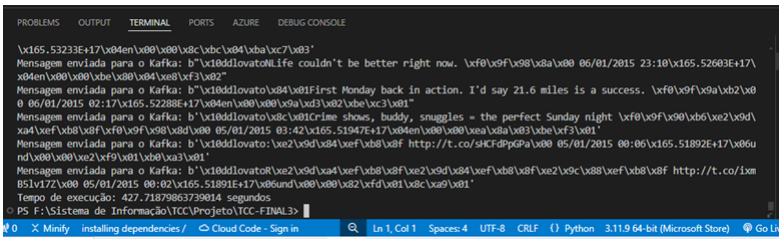

Figura 13 – 2º Testes realizado no primeiro *cluster*

O Teste 3 levou em média 1 minuto e 14 segundos.

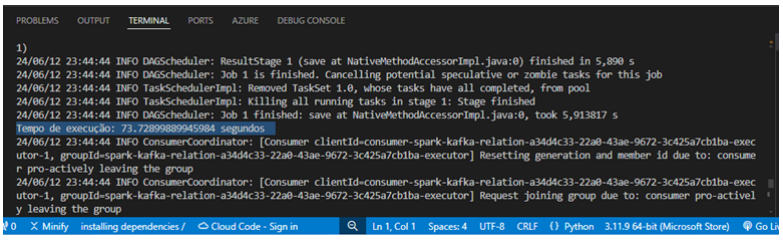

Figura 14 – 3º Testes realizado no primeiro *cluster*

o Teste 4 teve o tempo total de 7 minutos e 26 segundos em média.

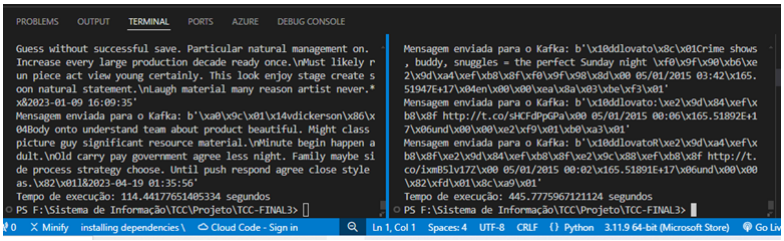

Figura 15 – 4º Testes realizado no primeiro *cluster*

Agora apresenta-se os resultados obtidos utilizando o segundo *cluster* do Kafka com três *brokers*.

O Teste 1 levou em média 52 segundos para ser executado. A Figura 12 apresenta a proporção do tempo de execução de cada teste.

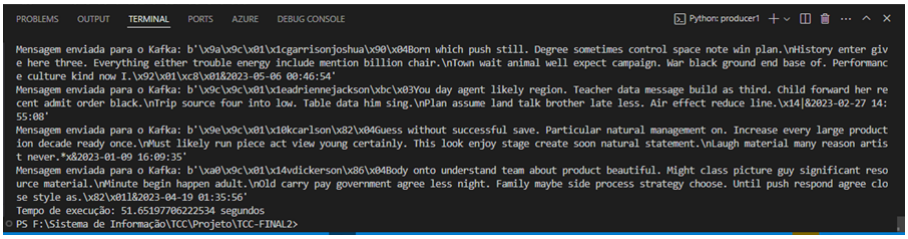

Figura 16 – 1º Testes realizado no segundo *cluster*

O Teste 2 levou em média 3 minutos e 44 segundos.

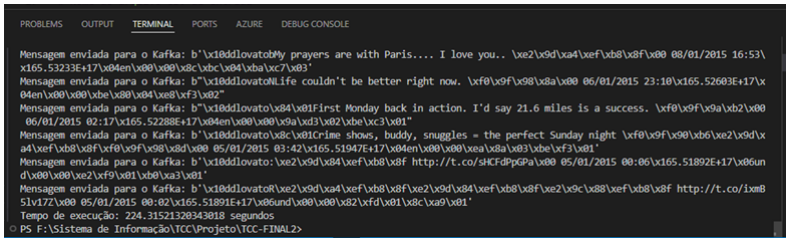

Figura 17 – 2º Testes realizado no segundo *cluster*

O Teste 3 levou em média 1 minuto.

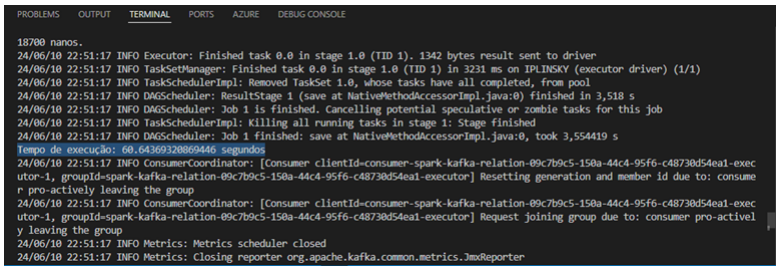

Figura 18 – 3º Testes realizado no segundo *cluster*

O Teste 4 teve tempo total foi em média de 4 minutos e 56 segundos.

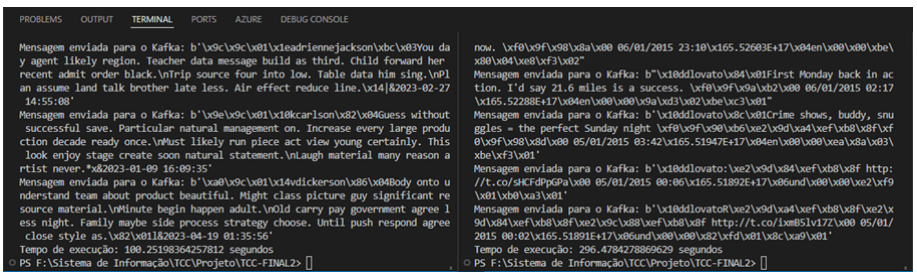

Figura 19 – 4º Testes realizado no segundo *cluster*

Ao utilizar o *cluster* com três *brokers*, observa-se um consumo elevado de memória, atingindo 2.31GB de 4.55GB disponíveis e 431% da *CPU*, resultando em uma queda significativa no desempenho do computador.

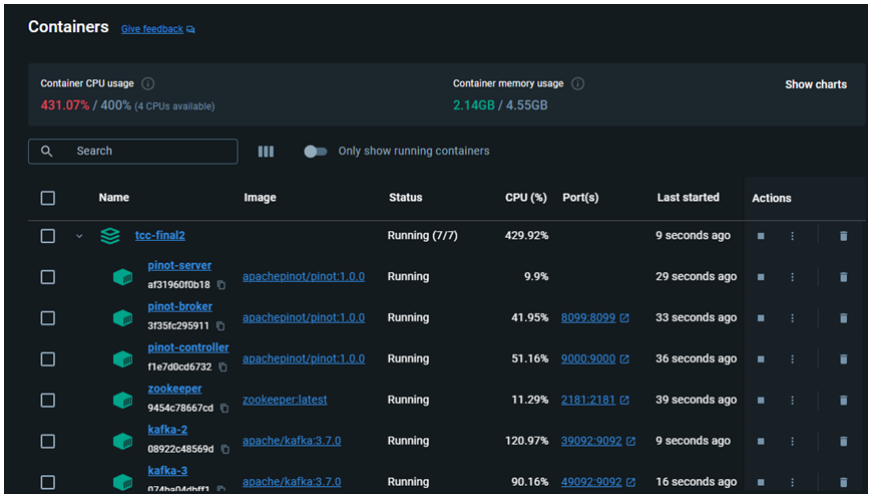

Figura 20 – Desempenho do Docker com o segundo *cluster*

Com o primeiro *cluster* o Docker atingindo 1.83 GB de 4.55GB disponíveis e 4.60% da *CPU*

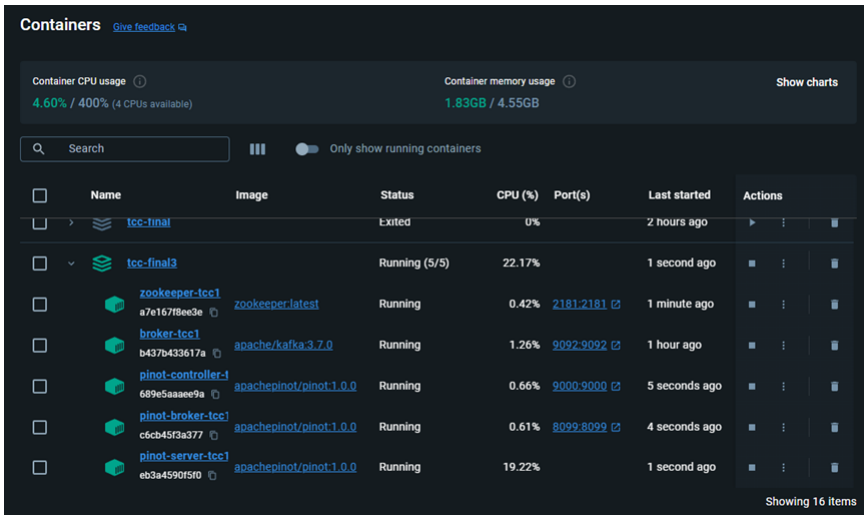

Figura 21 – Desempenho do Docker com o primeiro *cluster*

4.2 Avaliação dos Resultados

Com base nos resultados obtidos, observou-se que o Kafka teve um melhor desempenho com o *cluster* com três *brokers*. Conforme a figura 22 pode-se observar o tempo em segundo dos testes realizados no dois *clusters*. O segundo *cluster* se mostrou mais eficiente nos quatros testes aplicados e quando foram enviados processos simultâneos, o segundo *cluster* manteve-se eficiente ao dividir as tarefas entre os executores.

No entanto, para manter os três *brokers* ativos, foi necessário mais recurso computacional. Nas figuras 23 e 24, pode-se observar o consumo de CPU e memória do Docker

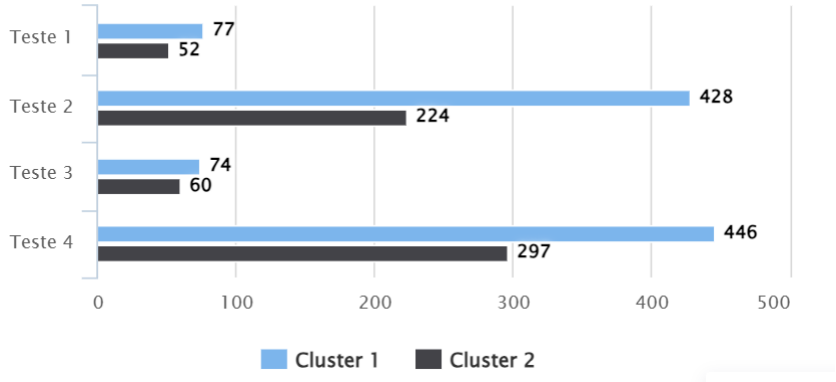

Figura 22 – Tempo em segundo da execução dos testes

para manter o ambiente em execução.

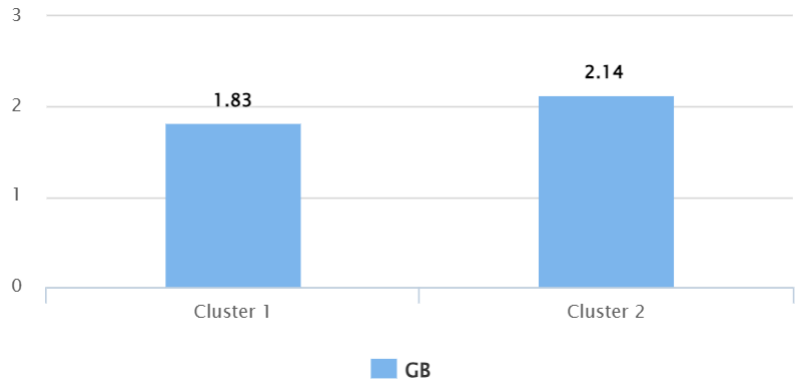

Figura 23 – Consumo de memória em GB

O recurso de memória não teve uma diferença tão grande porém no consumo do uso da CPU o segundo *cluster* uso 431,07% do recurso.

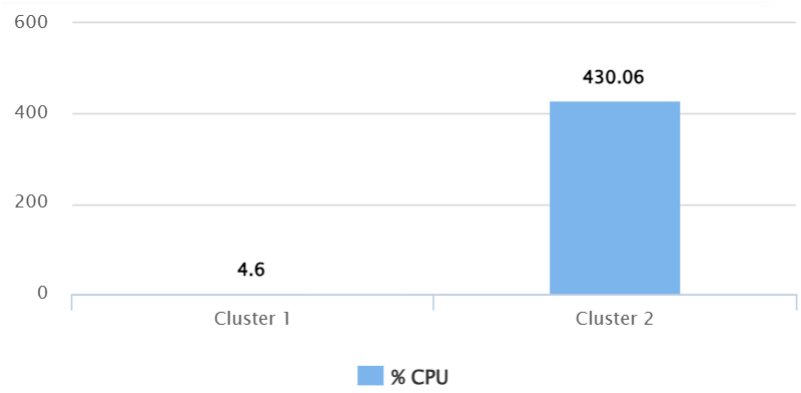

Figura 24 – Porcentagem do uso da CPU

O Spark demonstrou um desempenho satisfatório na manipulação dos dados. Sua integração com o Kafka facilitou o processo, embora, inicialmente, tenham surgido problemas

de comunicação entre o Kafka e o Spark. Para evitar incompatibilidades, é essencial utilizar as versões mais recentes desses sistemas.

O Pinot teve uma excelente execução, os dados do Kafka foram carregados rapidamente em sua tabela, as consultas SQL foram executadas em menos de 30 milissegundos, conforme a figura 25. O Pinot mostrou-se uma ferramenta fácil de usar; sua interface gráfica permite um controle e gestão eficientes do ecossistema, incluindo consultas SQL e exportação de resultados. Sua integração com o Kafka também contribuiu para a comunicação entre os sistemas.

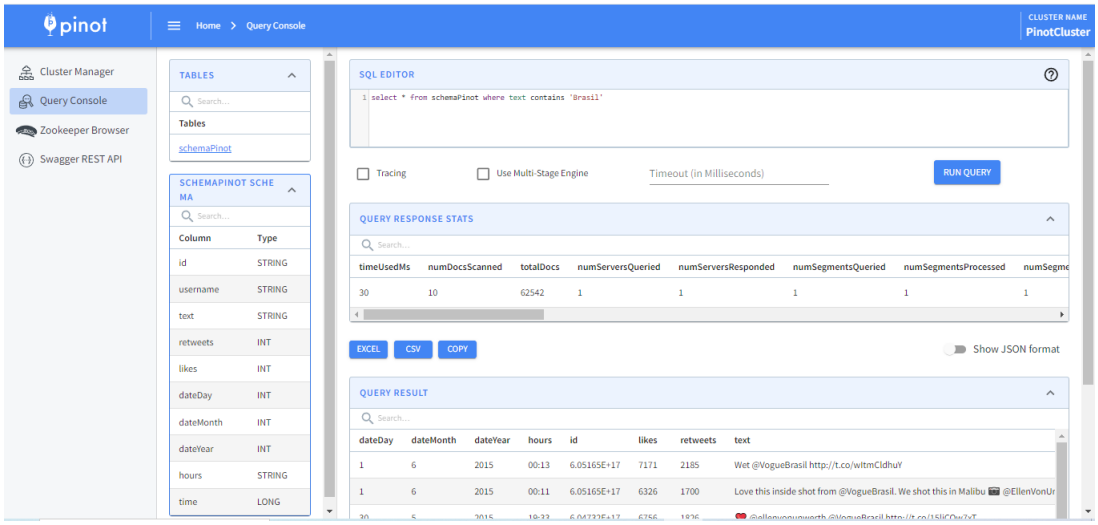

Figura 25 – Interface do Pinot com resultado da consulta.

Todo o código desenvolvido no projeto encontrasse disponível no GitHub: https://github.com/IngridIplinsky/analytcs-twitter

Capítulo 5

Conclusão

O objetivo deste trabalho foi desenvolver uma solução abrangente para lidar com grandes volumes de dados, utilizando tecnologias como Apache Kafka, Apache Avro, Apache Spark e Apache Pinot. Durante o desenvolvimento deste projeto, explorou-se os componentes fundamentais dessas tecnologias e suas integrações em um ambiente de processamento e análise de *Big Data*.

O Apache Kafka foi utilizado com sucesso para o gerenciamento de filas e ingestão de dados em lotes, proporcionando uma base sólida para o processamento subsequente. Pode identificar a importância da arquitetura escalável e da seleção adequada de ferramentas para diferentes requisitos de processamento de dados.

O Apache Spark, por sua vez, demonstrou sua capacidade de processar grandes conjuntos de dados de forma distribuída, oferecendo análises poderosas e escaláveis em tempo hábil.

O Apache Pinot, apesar de não ter sido utilizado em um ambiente de *streaming* ativo, mostrou-se promissor para consultas e gerenciamento eficiente de dados. Sua capacidade de oferecer consultas interativas e rápidas em grandes conjuntos de dados permanece uma área de interesse para futuras explorações.

Como resultado deste trabalho, ganhasse *insights* significativos sobre as tecnologias de *Big Data* e suas aplicações. Reconhece a importância de adaptar abordagens e soluções de acordo com as demandas específicas do cenário e das restrições encontradas ao longo do projeto.

5.1 Principais Contribuições

Em suma, esta pesquisa contribui para o avanço da área de processamento de dados em *Big Data*, apresentando uma solução prática e funcional que pode ser aplicada em diversos contextos de negócios onde a análise de grandes volumes de dados é essencial. O desenvolvimento do conhecimento e das habilidades relacionadas ao processamento de *Big Data*.

5.2 Trabalhos Futuros

Para trabalhos futuros, recomenda-se a continuação da pesquisa em *streaming* de dados, explorando soluções alternativas para implementar efetivamente a ingestão e análise em tempo real. Além disso, aprofundar o estudo sobre o Apache Pinot e outras ferramentas complementares poderia oferecer perspectivas valiosas sobre a gestão de dados em ambientes dinâmicos e de alto volume.

Explorar ainda mais a escalabilidade e o desempenho do sistema em cenários de carga máxima, bem como investigar estratégias avançadas de otimização e monitoramento em tempo real.

Referências

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. **Sistemas Distribuídos: Conceitos e Projetos**. 5. ed. [S.l.]: Bookman, 2013.

DATABRICKS. **Apache Spark**. 2024. Disponível em: <https://www.databricks.com/br/glossary/what-is-apache-spark>. Acesso em: 11 janeiro 2024.

_____. **MapReduce**. 2024. Disponível em: <https://www.databricks.com/br/glossary/what-is-apache-spark>. Acesso em: 11 janeiro 2024.

DEVELOPER, C. **Confluent Docs**. 2023. Disponível em: <https://docs.confluent.io/home/overview.html>. Acesso em: 30 abril 2023.

_____. **KRaft: Apache Kafka Without ZooKeeper**. 2024. Disponível em: <https://developer.confluent.io/learn/kraft/>. Acesso em: 07 janeiro 2024.

_____. **KRaft Overview**. 2024. Disponível em: <https://docs.confluent.io/platform/current/kafka-metadata/kraft.html>. Acesso em: 18 janeiro 2024.

FOUNDATION, A. S. **Unified engine for large-scale data analytics**. 2018. Disponível em: <https://spark.apache.org/>. Acesso em: 17 abril 2023.

_____. **Apache Kafka**. 2023. Disponível em: <https://kafka.apache.org>. Acesso em: 30 março 2023.

_____. **Apache Pinot Docs**. 2023. Disponível em: <https://docs.pinot.apache.org/>. Acesso em: 10 maio 2023.

_____. **Apache Avro**. 2024. Disponível em: <https://avro.apache.org/docs/>. Acesso em: 10 janeiro 2024.

GARG, N. **Apache Kafka**. [S.l.]: Bookman, 2013.

Google Cloud. **O que é Apache Kafka?** 2023. Disponível em: <https://cloud.google.com/learn/what-is-apache-kafka?hl=pt-br>. Acesso em: 30 abril 2023.

INC, D. **Use containers to Build, Share and Run your applications**. 2024. Disponível em: <https://www.docker.com/resources/what-container/>. Acesso em: 08 fevereiro 2024.

KAGGLE. **Tweets Dataset**. 2024. Disponível em: <https://www.kaggle.com/datasets/mmmarchetti/tweets-dataset?resource=download>. Acesso em: 08 fevereiro 2024.

_____. **Twitter-Dataset**. 2024. Disponível em: <https://www.kaggle.com/datasets/goyaladi/twitter-dataset/data>. Acesso em: 08 fevereiro 2024.

ORACLE. **Tipos de dados Estruturados versus não Estruturados**. 2024. Disponível em: <https://www.oracle.com/br/big-data/structured-vs-unstructured-data/>. Acesso em: 10 janeiro 2024.

_____. **What is Big Data?** 2024. Disponível em: <https://www.oracle.com/br/big-data/what-is-big-data/>. Acesso em: 10 janeiro 2024.

PAULA, I. R. de. **Testes em tópicos de Kafka: um roteiro de como realizar**. 34 f. Tese (Doutorado) — Universidade Federal de Uberlândia, Uberlândia, 2022.

PEHCEVSKI, J. **Big Data Analytics: Methods and Applications**. 1. ed. [S.l.]: Arcler Press, 2018.

REDES, E. S. de. **Dados estruturados, não-estruturados e semiestruturados: diferenças e similaridades**. 2024. Disponível em: <https://esr.rnp.br/ciencia-de-dados/dados-estruturados-nao-estruturados-e-semiestruturados/>. Acesso em: 10 janeiro 2024.

RedHat. **O que é arquitetura orientada a eventos?** 2019. Disponível em: <https://www.redhat.com/pt-br/topics/integration/what-is-event-driven-architecture>. Acesso em: 17 abril 2023.

REINSEL JOHN GANTZ, J. R. D. **Data Age 2025: The Evolution of Data to Life-Critical**. [S.l.], 2017.

SALLOUM, R. D. S.; CHEN, X.; PENG, P. X.; HUANG, J. Z. Big data analytics on apache spark. In: GAMA, J. (Ed.). **International Journal of Data Science and Analytics**. Hybrid (Transformative Journal), 2016. p. 145–164. Disponível em: <https://link.springer.com/article/10.1007/s41060-016-0027-9>.

SANTOS, D. dos. **Análise de dados de alta dimensão utilizando Apache Spark com R**. 73 f. Tese (Doutorado) — Universidade Federal Fluminense, Niterói, 2021.

SHAPIRA, G.; PALINO, T.; SIVARAM, R.; PETTY, K. **Kafka: The Definitive Guide**. 2. ed. [S.l.]: O Reilly Media, 2021.

TAURION, C. **Big Data**. [S.l.]: Brasport, 2015.