RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 4 Nro. 1 2023

DESENVOLVIMENTO DE UMA PIPELINE DE DADOS UTILIZANDO SOLUÇÕES OPEN-SOURCE EM UM AMBIENTE DE BIG DATA

Maria Fernanda Moge dos Reis
Graduada em Engenharia de Software – Uni-FACEF
ferdinandamoge@gmail.com

Cassiano Nakaoka
Graduado em Ciências da Computação – Uni-FACEF
cassiano.nakaoka@gmail.com

Geraldo Henrique Neto
Docente do Uni-FACEF
geraldo@facef.br

Resumo

Este artigo apresenta o desenvolvimento de uma *pipeline* de dados de ponta a ponta usando apenas soluções *open-source* em um contexto de *Big Data*. O propósito é integrar diferentes tecnologias emergentes para solucionar desafios na criação de um fluxo de processamento de dados de alto desempenho. O artigo emprega *Docker* para conteinerização, *Trino* para consultas de dados e arquivos no formato *Parquet* para um armazenamento eficaz e rastreável. Também utiliza *Airflow* para orquestração de fluxos de trabalho e *Spark* para processamento de dados distribuídos. Os resultados demonstram que é possível desenvolver uma *pipeline* sem o uso de ferramentas proprietárias, oferecendo assim, uma opção para pequenas e médias empresas, que por sua vez lidam com um orçamento limitado. O projeto é capaz de fornecer uma *pipeline* autogerenciada e totalmente dinâmica, mas que demanda um consumo significativo de recursos, destacando a importância de otimização para escalabilidade em ambientes de produção.

Palavras-chave: *Airflow. Big Data. Docker. Open-source. Parquet. Pipeline. Spark. Trino.*

Abstract

This article presents the development of an end-to-end data pipeline using only open-source solutions in a Big Data context. The purpose is to integrate different emerging technologies to solve challenges in creating a high-performance data processing flow. The article employs Docker for containerization, Trino for data queries and files in Parquet format for efficient and traceable storage. It also uses Airflow for workflow orchestration and Spark for distributed data processing. The results demonstrate that it is possible to develop a pipeline without the use of paid tools, thus offering an option for small and medium businesses, which in turn deal with a limited budget. The project is able to provide a self-managed and fully dynamic pipeline, but that requires a significant consumption of resources, highlighting the importance of optimization for scalability in production environments.

Keywords: *Airflow. Big Data. Docker. Open-source. Parquet. Pipeline. Spark. Trino.*

Submissão: 23/09/2023. **Aprovação:** 30/09/2023.

1 Introdução

Na era da informação, o gerenciamento de grandes volumes de dados tornou-se um componente crítico para o sucesso de organizações em variados setores. Esse cenário, conhecido como Big Data, envolve desafios significativos em termos de armazenamento, processamento e análise. Conforme aponta Marques e França (2020, p. 2), "os softwares amplamente utilizados e que se destacam têm custos tão elevados que sua aquisição se torna inviável para muitos estudantes, pesquisadores e empresas". Portanto, embora soluções proprietárias possam oferecer robustez e suporte técnico, frequentemente são economicamente inviáveis para organizações de menor porte ou projetos com recursos limitados. Em contrapartida, as ferramentas de código aberto, por serem gratuitas, proporcionam uma alternativa economicamente viável, embora possam representar desafios técnicos e conceituais.

Considerando o que foi exposto, este artigo tem como objetivo geral o desenvolvimento de uma *pipeline* de dados em ambiente de *Big Data*, empregando unicamente tecnologias de código aberto, de forma a estabelecer uma alternativa de baixo custo que seja eficiente para armazenamento, processamento e análise de grandes conjuntos de dados.

Para atingir esse objetivo geral, o artigo tem os seguintes objetivos específicos:

- Identificar e selecionar as ferramentas *open-source*: Avaliar diversas soluções de código aberto disponíveis e selecionar as mais adequadas para o contexto específico do estudo, com base em critérios como eficiência e escalabilidade.
- Desenvolver a arquitetura da *pipeline*: Planejar e documentar a arquitetura da *pipeline*, detalhando como os componentes selecionados interagem e contribuem para a eficácia da solução em um ambiente de *Big Data* específico.
- Implementar uma versão piloto: Construir uma versão de prova de conceito da *pipeline* para testar e validar sua funcionalidade.

É fundamental observar que o uso de ferramentas de código aberto, embora potencialmente menos oneroso do ponto de vista financeiro, pode ser tecnicamente mais desafiador e exigir um maior investimento de tempo e esforço. Ao concretizar esses objetivos específicos, o artigo almeja demonstrar que é possível desenvolver uma *pipeline* de dados eficiente para ambientes de *Big Data*, mesmo quando se enfrentam desafios técnicos adicionais associados ao uso de ferramentas de código aberto.

2 Fundamentação Teórica

Antes de aprofundarmos na tarefa de criar e implementar a *pipeline* de dados, é essencial entender claramente os diferentes aspectos e complexidades associados a esta solução. Esta seção tem como objetivo estabelecer um alicerce teórico sólido, abordando tópicos críticos como o que compõe uma *pipeline* de dados, os prós e contras de adotar soluções *open-source* e as características específicas de trabalhar com *Big Data.* Esta base teórica servirá como um ponto de partida para os objetivos e métodos de pesquisa que serão explorados neste estudo.

2.1. Pipeline de Dados

Para automatizar um fluxo de dados de ponta a ponta, são criadas *pipelines* para lidar com o processamento de dados em ambientes de *Big Data*. De acordo com Marz e Warren (2015), uma *pipeline* de dados é uma sequência de etapas interconectadas que envolvem a extração, transformação e carga (ETL) de dados de diversas fontes, a fim de fornecer informações relevantes para análise e tomada de decisão. Essa abordagem permite que organizações lidem com o crescente volume, variedade e velocidade dos dados, além de garantir sua qualidade e confiabilidade.

A *pipeline* de dados é composta por diferentes estágios, que podem incluir a ingestão de dados brutos, limpeza, transformação, enriquecimento, análise e carregamento em um repositório final. Esses estágios são essenciais para garantir que os dados sejam processados de forma eficiente, otimizando o fluxo de informações e permitindo a geração de *insights* valiosos.

No contexto da construção de uma *pipeline* de dados utilizando soluções *open-source*, várias ferramentas podem ser exploradas. Algumas das ferramentas populares incluem o *Apache Airflow*, que oferece recursos avançados de agendamento e monitoramento de tarefas; o *Trino*, que permite consultas rápidas e interativas em diferentes fontes de dados; o *Apache Superset*, que oferece recursos de visualização de dados interativos e painéis de controle; e o *Apache Spark*, um poderoso *framework* de processamento distribuído que suporta diversas tarefas de processamento e análise de dados de forma distribuída.

2.2. Soluções Open-Source

Neste projeto, a compreensão do conceito de soluções *open-source*, ou *softwares* de código aberto, é fundamental. Segundo a *Open Source Initiative* (OSI, 2007), o termo *"open source"* descreve *softwares* com código-fonte acessível ao público, possibilitando a sua livre modificação e distribuição. Esta filosofia baseia-se em transparência, colaboração e compartilhamento de conhecimento, conferindo aos usuários maior controle sobre as tecnologias que empregam.

O benefício financeiro é um dos atrativos mais chamativos das soluções *open-source* na construção de *pipelines* de dados. Como apontado por Opensource.com (s.d), *softwares* comerciais usualmente demandam investimentos consideráveis, incluindo a aquisição de licenças, treinamento especializado e suporte técnico. Para organizações com recursos limitados, esses custos podem ser um obstáculo. Em contraste, soluções *open-source* oferecem uma alternativa mais viável, permitindo que empresas de todos os portes possam acessar recursos de processamento e análise de dados sem a necessidade de grandes investimentos iniciais.

O modelo open-core, como discutido por Perens (1999), refere-se a uma abordagem onde a versão base do *software open-source* é oferecida de forma gratuita. Contudo, versões avançadas ou extensões do software que contêm recursos adicionais ou oferecem suporte especializado são comercializadas. É essencial considerar que, apesar das soluções *open-source*, muitas vezes, demandarem um investimento inicial mais acentuado por não possuírem suporte dedicado, elas podem se equiparar em eficiência às alternativas pagas. Com a expertise apropriada, um time de TI pode, progressivamente, ultrapassar os desafios iniciais de implementação e gerenciar o software de maneira independente.

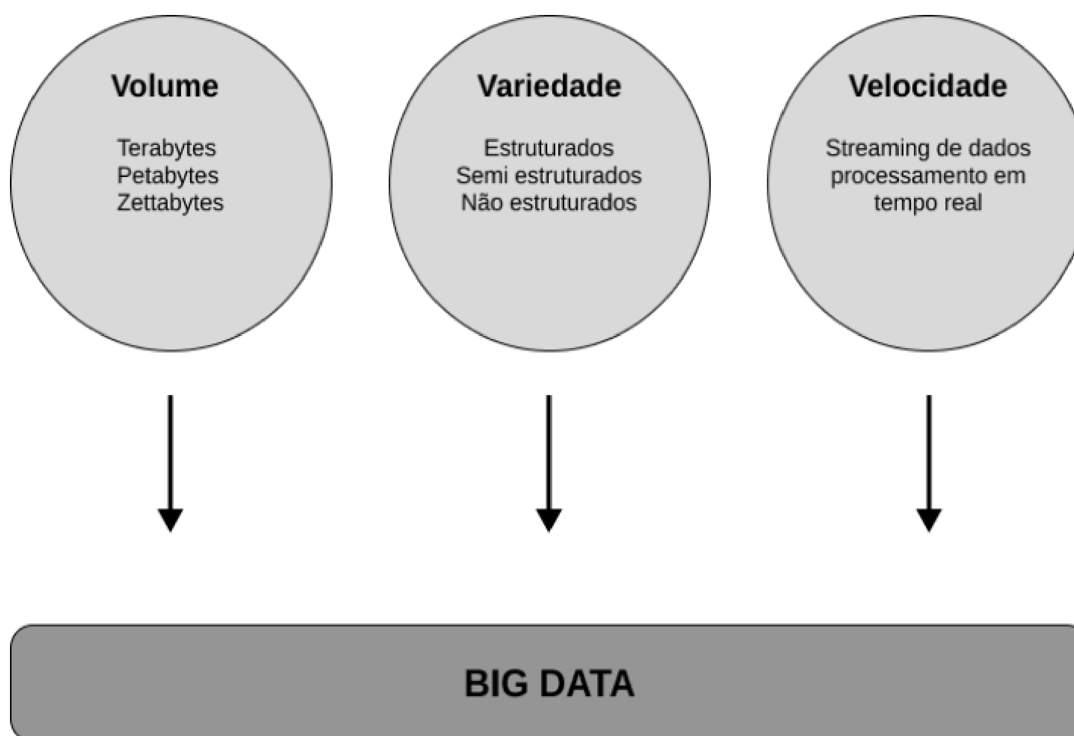
Outras vantagens das soluções *open-source* na construção de *pipelines* de dados incluem a liberdade para personalizar e adaptar as ferramentas às necessidades específicas dos usuários. Isso proporciona maior flexibilidade e liberdade para experimentar diferentes abordagens e otimizar a performance da *pipeline* de dados. Como mencionado por Opensource.com (s.d.), a comunidade de desenvolvedores *open-source* frequentemente contribui com melhorias e correções de *bugs*, resultando em atualizações mais rápidas e um *software* mais robusto.

A utilização de soluções *open-source* na construção de *pipelines* de dados oferece uma série de vantagens que ultrapassam a mera economia de recursos. A flexibilidade, a capacidade de personalização, a robustez proporcionada pela constante contribuição da comunidade e a possibilidade de superar a dependência de suporte dedicado tornam as soluções *open-source* uma escolha eficiente e eficaz. Portanto, elas, apesar dos desafios, provam ser uma opção vantajosa no longo prazo, destacando-se como uma escolha preferencial para a construção dessa *pipeline* de dados.

2.3. Ambiente de Big Data

Apesar de o termo *Big Data* referir-se principalmente a conjuntos de dados extremamente grandes que excedem a capacidade de processamento e armazenamento dos sistemas convencionais de banco de dados, sua definição pode ser considerada mais abrangente. Assim como diversos autores, para a autora Marquesone (2018) esses conjuntos de dados são caracterizados por três principais aspectos conhecidos como os "3Vs" de *Big Data*: volume, variedade e velocidade. Na Figura 1 exemplificar-se os "Vs" citados:

Figura 1. 3Vs de Big Data.



Fonte: (Marquesone, 2018, p.9)

De acordo com Marquesone (2018), o volume refere-se à enorme quantidade de dados gerados diariamente por diversas fontes, como redes sociais, dispositivos móveis, sensores, transações comerciais, entre outros coletores de dados. A variedade refere-se à diversidade de tipos e formatos de dados que podem ser encontrados em um ambiente de *Big Data*. Esses dados podem incluir texto, imagens, vídeos, áudio, *logs* de servidores, dados geoespaciais, entre outros. E a velocidade refere-se à necessidade de processar e analisar os dados em tempo real ou próximo disso. Em muitos casos, a tomada de decisões ou ações precisam ser realizadas rapidamente com base nos dados disponíveis. Além dos "3Vs" citados pela autora, MACHADO (2018) em seu livro "*Big Data: O Futuro dos Dados e Aplicações*" aponta dois novos "Vs" essenciais no contexto de *Big Data*:

> Dessa forma, para Big Data, temos de considerar como base cinco grandes pilares; são os 5 Vs: velocidade, volume, variedade, veracidade e valor. Acrescentamos dois Vs importantíssimos: veracidade dos dados e valor destes para a empresa.
> [...] A veracidade é a necessidade que temos de garantir que os dados coletados sejam autênticos (com relação à fonte da informação) e que são verdadeiros naquele momento. Devemos lembrar que nem tudo postado em uma rede social é verdadeiro e confiável, assim como todos os sistemas podem possuir dados com erros. [...] O valor é o ponto mais destacado em relação a aplicações de Big Data. Nada dos

> conceitos e exemplos citados anteriormente faz sentido se não for possível extrair valor dos dados que seja útil para análises sobre os negócios de uma empresa (MACHADO, 2018).

Portanto, os cinco pilares fundamentais para o *Big Data* são: velocidade, volume, variedade, veracidade e valor. Esses aspectos se complementam e são essenciais para lidar com o imenso fluxo de dados e obter informações significativas para a tomada de decisões nas empresas. Ao considerar esses pilares, é possível aproveitar todo o potencial dos dados e impulsionar o crescimento e o sucesso das organizações.

2.4. Integração entre Pipeline e Big Data

A fusão de pipelines de dados com o ambiente de *Big Data* é fundamental para a gestão eficaz e a maximização do uso do potencial dos dados. Desenvolver uma pipeline de dados que funcione em um ambiente de *Big Data* requer um entendimento profundo dos desafios e peculiaridades trazidos por esses grandes volumes de informações. Uma *pipeline* de dados em Big Data é geralmente vista como um conjunto especializado de soluções ETL (*Extract, Transform and Load*) capaz de manipular dados estruturados, semiestruturados e não estruturados, originados de várias fontes. Esta abordagem versátil permite extrair dados de quase qualquer fonte, processando-os através de várias transformações, como enriquecimento, deduplicação e agregação, antes de enviá-los a diferentes destinos, como *Data Lakes*, bancos de dados relacionais e *Data Warehouses*.

O propósito final de qualquer *pipeline* de dados em um ambiente de *Big Data* é extrair informações que impulsionem a análise e, por consequência, o sucesso das organizações. Isso envolve a transformação de dados brutos em *insights* significativos e acionáveis, um processo que requer a harmonização eficaz entre a *pipeline* e o contexto de *Big Data* em que opera. De acordo com a IBM (2023) existem dois tipos principais de *pipelines* de dados: processamento em lote e processamento em fluxo. O processamento em lote consiste em mover grupos de dados do ponto A para o ponto B em intervalos regulares ou pontuais, geralmente durante horários fora do pico. O processamento em fluxo permite o movimento de dados em tempo real, coletando continuamente dados de fontes como sensores, sistemas de mensagens ou bancos de dados. Cada tipo tem suas vantagens e desvantagens, dependendo da necessidade e da velocidade da análise.

Portanto, a integração entre *pipelines* de dados e ambientes de *Big Data* é um casamento vital para organizações que buscam desbloquear o potencial total de suas vastas reservas de informações. Ao compreender e otimizar os componentes essenciais dessa conexão, as empresas podem não apenas processar dados em escala, mas também gerar valor tangível a partir deles, impulsionando assim sua vantagem competitiva e sucesso a longo prazo. Conforme o cientista de dados Hassan (2023), uma *pipeline* de dados em *Big Data* deve ser capaz de lidar com os três principais atributos do *Big Data*: volume, variedade e velocidade. Além disso, deve-se considerar aspectos como segurança, qualidade, governança e custo dos dados. Para isso, é preciso escolher as ferramentas adequadas para cada etapa da *pipeline*, bem como definir as melhores práticas e padrões para garantir a confiabilidade e a consistência dos dados.

3 Ferramentas

Nesta seção, é detalhada a arquitetura da *pipeline* de dados escolhida, bem como as soluções *open-source* selecionadas para este projeto, também o passo a passo do desenvolvimento, desde a escolha das ferramentas até a implementação prática, visando fornecer uma visão transparente da metodologia adotada, permitindo a replicação e adaptação da *pipeline* em diferentes contextos ou conjuntos de dados.

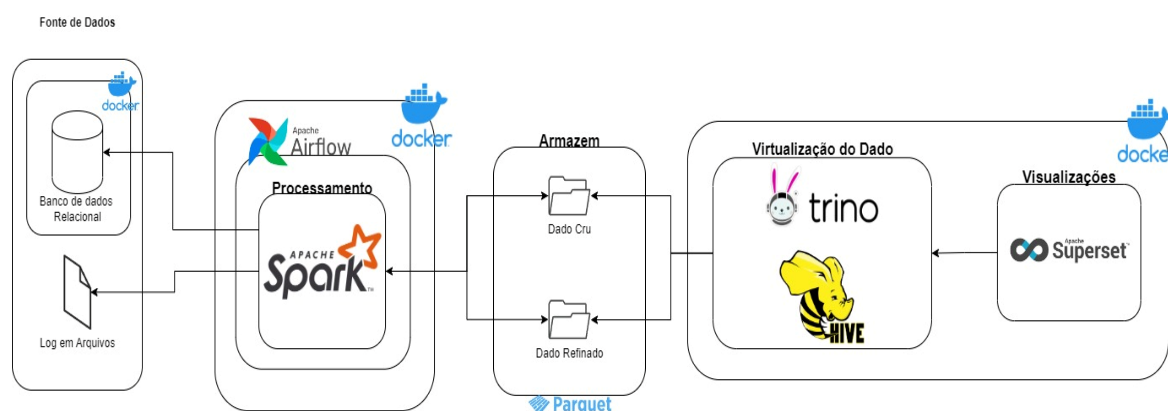
3.1. Definição da Arquitetura

Anteriormente, discutimos os cinco conceitos fundamentais do *Big Data* e quais os requisitos necessários que uma arquitetura de uma *pipeline* de dados precisa para funcionar de forma concisa. Estes pilares não apenas definem a solução que será apresentada, mas também ditam as necessidades e exigências para realizar o processamento e análise de dados em grande escala.

Para construir uma arquitetura coesa e eficiente, a interação de diversos componentes é tão crucial quanto suas funcionalidades individuais. Cada elemento foi escolhido não apenas pelo que pode fazer individualmente, mas também pela forma como se integra ao ecossistema geral, garantindo que os cinco pilares do *Big Data* sejam atendidos de maneira holística.

A arquitetura foi definida como apresentada na Figura 2.

Figura 2. Arquitetura da Pipeline.



Fonte: Imagem do autor.

Os aspectos considerados para a definição da *pipeline* nas próximas seções.

3.1.1 Entrada e Fontes de Dados

De acordo com Kimball et al. (2008), a entrada dos dados é um passo crucial em qualquer projeto de processamento e análise de dados. As fontes e seus respectivos tipos de arquivos não apenas determinam a qualidade e a relevância dos *insights* extraídos, mas também ditam os métodos e as ferramentas que serão utilizados para o tratamento, armazenamento e consulta desses dados. Provost e Fawcett (2013) também apontam para a importância de selecionar fontes adequadas para atender aos requisitos de projetos de *Big Data*.

Considerando o contexto de *Big Data*, foram definidos três tipos de fontes que a *pipeline* será capaz de receber:

- Arquivos JSON (*JavaScript Object Notation*): Amplamente utilizados em várias aplicações e plataformas, são flexíveis e oferecem uma estrutura de dados fácil de ler e escrever. Devido ao seu formato auto-descritivo, eles são especialmente úteis para armazenar dados estruturados e semiestruturados.
- Arquivos CSV (*Comma-Separated Values*): Esse é um dos formatos mais simples e antigos de armazenar dados tabulares, tornando-o universalmente aceito e utilizado. Devido à sua simplicidade, eles são ideais para uma rápida ingestão e

processamento, especialmente quando a estrutura dos dados é conhecida antecipadamente.

- Bancos de Dados Relacionais: Os RDBMS (Bancos de Dados Relacionais) têm sido a espinha dorsal do armazenamento de dados por várias décadas. Seu modelo estruturado e a capacidade de estabelecer relações entre tabelas os tornam ideais para armazenar e consultar grandes conjuntos de dados estruturados. Esses bancos de dados são geralmente caracterizados por sua robustez, confiabilidade e recursos avançados de gerenciamento de transações.

Estas fontes foram selecionadas não apenas por suas características individuais, mas também pela forma como elas se complementam em um *pipeline* de dados. Juntas, elas oferecem um conjunto flexível de opções para a ingestão, armazenamento e consulta de dados, atendendo grande parte das necessidades dos projetos de *Big Data* de hoje.

3.1.2 Processamento de Dados

É nesta fase que os dados brutos são transformados em informações úteis que podem ser utilizadas para tomar decisões informadas. O processamento pode ser dividido em várias etapas, frequentemente abordadas por meio de processos de ETL e processamento em *batch.* O objetivo da camada de processamento é oferecer uma ou mais plataformas para processamento distribuído e análise de grandes conjuntos de dados (CIÊNCIA E DADOS, 2018).

O ETL (Extração, Transformação e Carregamento) é um dos métodos mais comuns para o processamento de dados. A Extração envolve a coleta de dados de várias fontes. A Transformação envolve a limpeza, agregação e, muitas vezes, a reestruturação dos dados. O carregamento é o estágio final, em que os dados transformados são armazenados de volta em um *data warehouse* ou outro sistema de armazenamento para análise futura (KIMBALL & CASERTA, 2004). A importância do ETL não pode ser subestimada, pois ele permite que as organizações convertam dados brutos em um formato mais estruturado, facilitando as operações subsequentes de análise e relatório.

E o processamento em *batch* é uma técnica que envolve o processamento de grandes volumes de dados acumulados ao longo de um período. Ao contrário do processamento em tempo real, onde os dados são processados quase instantaneamente à medida que entram no sistema, o processamento em *batch* lida com dados em intervalos fixos. Esta abordagem é especialmente útil para cenários onde é necessário processar grandes volumes de dados de maneira eficiente, garantindo que os recursos computacionais sejam utilizados de forma otimizada. É notável que o processamento em *batch* é adotado por organizações por exigir interação humana mínima, tornando as tarefas repetitivas mais eficientes.

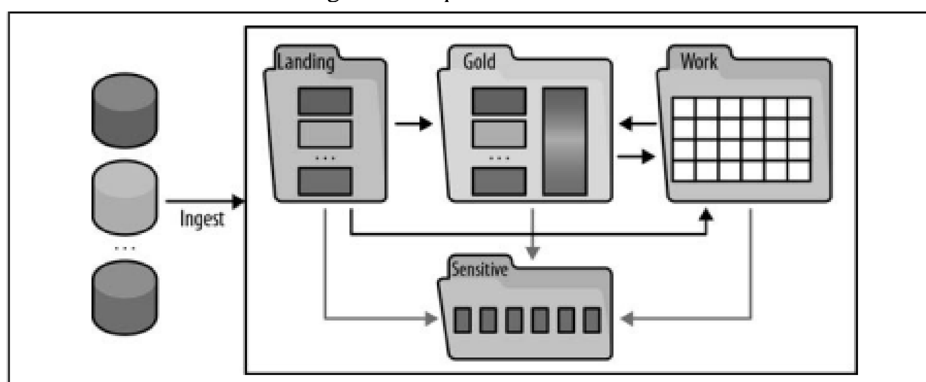
Os lotes de trabalhos são configurados para serem processados quando a capacidade computacional está mais prontamente disponível, minimizando o estresse nos sistemas. Além disso, este método de processamento moderno requer mínima supervisão humana; em caso de problemas, o sistema notifica automaticamente a equipe responsável (AWS, 2023).

Para que o sistema de processamento de dados funcione de forma eficiente, é crucial que todas as etapas estejam bem coordenadas. Isso implica em estabelecer uma sequência bem definida para as várias tarefas de processamento e garantir que cada etapa seja concluída antes que a próxima comece. Essa coordenação é especialmente crítica em ambientes que lidam com grandes volumes de dados variados, e onde os requisitos de veracidade e valor são altos.

3.1.3 Armazenamento de Dados

O armazenamento é uma das fases mais críticas em qualquer *pipeline* de dados. Ele não apenas serve como a base para a análise de dados, mas também como um repositório para manter dados brutos e transformados de forma eficaz e eficiente. Seguindo o modelo descrito por Alex Gorelik (2019), uma arquitetura de *data lake* típica é organizada em diversas zonas: uma zona *raw ou landing*, uma zona *gold* ou camada *refined*, e uma zona de trabalho ou *dev/work zone*. Na Figura 3 estão exemplificadas as camadas

Figura 3. Arquitetura Data Lake



Fonte: (GORELIK, 2019, p.46)

A zona *raw* ou *landing* atua como um "lago de dados" inicial, onde os dados brutos provenientes de diversas fontes são armazenados em seu estado original. A importância dessa camada reside em sua capacidade de atuar como um registro histórico imutável, o que é crucial para a rastreabilidade, auditoria e reprodutibilidade.

Na zona *gold* ou camada *refined*, os dados são refinados e transformados para análise. De acordo com Alan R. Simon (2021), essa camada é essencial para limpar e tratar dados, além de realizar agregações e outras transformações que tornam os dados mais acessíveis para casos de uso específicos.

Em relação à *sensitive zone*, sua necessidade e aplicação dependem do tipo de dados que estão sendo processados. Essa camada é especialmente dedicada ao armazenamento de dados sensíveis que requerem controles de acesso e medidas de segurança adicionais.

Contrastando com a zona *gold*, a *work zone* serve como um ambiente mais dinâmico e adaptável, projetado para os usuários técnicos, como cientistas de dados e engenheiros de dados, realizarem suas atividades. Esta zona é intimamente ligada à camada de leitura de dados, que serve como a interface entre os dados armazenados e as aplicações ou usuários que os acessam. Conforme descrito por Gorelik (2019), uma vez que o trabalho analítico na *work zone* é finalizado, ele é frequentemente movido para a zona gold. Portanto, a camada de leitura de dados age como um elo crítico entre a *work zone* e a zona *gold*, facilitando a recuperação eficiente e eficaz dos dados para diversos fins, como análise, visualização e relatório.

Para este projeto foi optado por não utilizar da *sensitive zone* e nem da *work zone* já que o conjunto de dados escolhido não apresenta dados sensíveis ou a necessidade de aplicação de regras de negócio complexas, caso seja necessário, a *pipeline* pode ser adaptada de forma a contemplar essas demais camadas.

3.2. Ferramentas Open-Source Presentes na Arquitetura da Pipeline

Na construção da *pipeline*, a seleção das ferramentas corretas é crucial. Elas devem ser escolhidas com base em critérios como escalabilidade, desempenho, facilidade de uso, e suporte. Neste contexto, as soluções *open-source* surgem como opções atraentes, pois

oferecem flexibilidade, robustez e custo-benefício. A seguir, serão apresentadas sete ferramentas *open-source* selecionadas para a construção da *pipeline* de dados neste projeto: *Docker*, Apache Parquet *Apache Hive*, *Apache Airflow*, *Trino, Apache Superset e Apache Spark.* A escolha dessas ferramentas foi baseada em suas características únicas, capacidades e adequação às necessidades específicas da *pipeline*.

3.2.1 Docker

Docker é uma plataforma de conteinerização que simplifica a implementação e execução de aplicações em ambientes isolados. Isso é particularmente relevante para arquiteturas de *data lake*, uma vez que permite uma maior flexibilidade e portabilidade na implementação de serviços e aplicações. A tecnologia *Docker* cria um ambiente isolado chamado contêiner, o qual pode ser usado para empacotar uma aplicação e suas dependências, facilitando assim o processo de implantação e escalabilidade.

Conforme elucidado por James Turnbull (2014), *Docker* é uma ferramenta que pode empacotar uma aplicação e suas dependências em um contêiner virtual, que pode então ser executado em qualquer servidor *Linux*. Isso permite que as aplicações sejam executadas da mesma forma, independentemente de onde elas estão sendo executadas. Este recurso é vital para arquiteturas de *data lake*, nas quais a consistência e a replicação são fundamentais para a manipulação eficaz de grandes volumes de dados.

A tecnologia de contêiner também é discutida no contexto de microserviços por Sam Newman (2015), que ressalta como a conteinerização, especialmente através do *Docker*, permite uma abordagem modular para desenvolvimento e implantação de serviços. Isso facilita a manutenção e escalabilidade do sistema como um todo.

Portanto, a adoção dele em ambientes de *Big Data* pode oferecer vantagens significativas, como isolamento de ambiente, portabilidade e escalabilidade, contribuindo para uma arquitetura mais robusta e flexível.

3.2.2 Apache Parquet

Apache Parquet é um formato de arquivo colunar de código aberto que é otimizado para uso com armazenamento de dados e sistemas de big data. Sua eficiência na armazenagem e leitura de grandes volumes de dados faz dele uma escolha popular em arquiteturas de data lake, especialmente quando se lida com frameworks de análise de dados como Apache Spark, Hadoop e outras ferramentas de processamento de dados em larga escala.

Tom White (2015), destaca que formatos colunares como Apache Parquet são particularmente eficazes para realizar operações analíticas em grandes conjuntos de dados. Eles permitem que os sistemas de leitura pulem rapidamente através de colunas específicas de dados, melhorando assim a eficiência de operações de leitura.

Damji et al (2020), menciona a alta compatibilidade do Apache Parquet com o Apache Spark, reiterando sua utilidade em armazenar dados em um formato que é otimizado tanto para armazenamento quanto para operações analíticas.

A integração do Apache Parquet em um data lake tem o potencial de melhorar significativamente a eficiência na leitura e escrita de dados. Além disso, devido à sua capacidade de armazenar dados em um formato compactado, também contribui para a economia de custos de armazenamento, tornando-se assim uma escolha estratégica para otimizar operações em arquiteturas de data lake.

3.2.3 Apache Hive

Apache Hive é uma plataforma de armazenamento de dados construída sobre o *hadoop* que oferece recursos de consulta e análise de dados. Ele foi projetado para facilitar a leitura, escrita e gerenciamento de grandes volumes de dados distribuídos no sistema de arquivos *HDFS (Hadoop Distributed File System*).

Edward Capriolo, Dean Wampler e Jason Rutherglen (2012), ressaltam que o *Hive* foi projetado para permitir que as pessoas que já estão acostumadas com *SQL* possam realizar operações em larga escala no Hadoop. Isso torna mais fácil para as organizações integrarem suas soluções existentes de *SQL* com plataformas de Big Data.

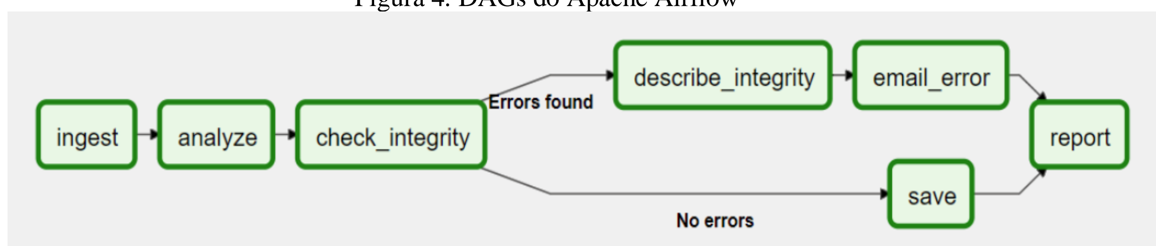
De acordo com Tom White (2015), o *Apache Hive* é útil para executar consultas *ad hoc* e operações de análise em conjuntos de dados grandes, bem como para suportar consultas mais complexas usando diversos algoritmos de junção. Ao implementar o *Apache Hive* em uma arquitetura de *data lake*, a plataforma serve como uma camada intermediária entre os dados armazenados e as aplicações de análise. Isso torna o processo de extração de *insights* de dados brutos mais eficiente e acessível, mesmo para usuários que não são especialistas em programação.

3.2.4 Apache Airflow

O *Apache Airflow* é uma plataforma de orquestração de fluxo de trabalho que permite programar, organizar e monitorar as etapas de uma *pipeline*. Segundo Julian de Ruiter e Bas Harenslak (2021) ele funciona com o conceito de DAGs (*Directed Acyclic Graphs*), que são representações gráficas dos fluxos de trabalho. Cada DAG é composto por um conjunto de tarefas, que são unidades lógicas de trabalho que podem ser executadas por diferentes operadores. Os operadores são objetos que definem como uma tarefa será executada, podendo ser desde simples comandos em *Python* até chamadas a serviços externos. As tarefas são conectadas por dependências, que especificam a ordem e as condições para a execução dos fluxos de trabalho, ora visualizado na Figura 4.

As DAGs podem ser exemplificadas da seguinte maneira:

Figura 4. DAGs do Apache Airflow



Fonte: (AIRFLOW Documentation)

Além disso, pontuam que o *Airflow* possui uma arquitetura modular e extensível, que permite adicionar novos operadores, sensores, ganchos e executores conforme as necessidades do projeto. Os sensores são operadores especiais que esperam por um determinado evento ou condição para disparar uma tarefa. Os ganchos são interfaces para se comunicar com fontes de dados externas, como bancos de dados, APIs ou sistemas de arquivos. Os executores são componentes responsáveis por distribuir e gerenciar a execução das tarefas em diferentes ambientes.

Na documentação e *site* oficial do *Airflow* (2023*)*, é apontado que ele também possui uma *interface web* que permite visualizar e interagir com os DAGs, as tarefas e os *logs*. Através dela, é possível criar, editar, ativar, desativar, testar e monitorar os fluxos de trabalho, bem como receber alertas em caso de falhas ou atrasos. A *interface web* também oferece uma

visão geral do estado dos fluxos de trabalho, das tarefas pendentes, dos recursos utilizados e das métricas de desempenho.

Para este projeto, a escolha do *Airflow* foi motivada por sua capacidade de definir fluxos de trabalho em código, o que permite flexibilidade e reprodutibilidade. Além disso, sua rica interface de usuário e recursos avançados de agendamento e monitoramento tornam o gerenciamento de tarefas mais acessível. A integração com diversas fontes de dados e sua escalabilidade o tornam uma escolha ideal para trabalhar com *Big Data*. O *Airflow* será utilizado para coordenar as tarefas necessárias da *pipeline* de dados.

3.2.5 Trino

O *Trino* é um mecanismo de consulta distribuído de alto desempenho que permite consultas rápidas e interativas em diferentes fontes de dados. Originalmente desenvolvido pelo Facebook sob o nome de *Presto*, o projeto foi posteriormente renomeado para *Trino* pela comunidade *open-source*. Uma das principais características do Trino é sua capacidade de se integrar a diversos sistemas de armazenamento de dados, como bancos de dados relacionais e não relacionais, através de conectores. Estes conectores abstraem as diferenças entre as fontes, permitindo que os usuários realizem consultas federadas, unindo dados de múltiplas origens em uma única busca

Segundo a documentação oficial do *Trino* (2021), além da integração com várias fontes de dados, o *Trino* também se destaca pelo seu desempenho e escalabilidade. Ele utiliza um modelo de execução baseado em *pipelining*, que permite o processamento paralelo e assíncrono das consultas, reduzindo a latência e o consumo de recursos. Esse modelo permite que ele realize consultas rápidas, tornando-o adequado para análises em tempo real e/ou ambientes de *Big Data*. O *Trino* é projetado para escalar horizontalmente, adicionando mais nós ao *cluster* conforme a demanda aumenta. Ele também utiliza um protocolo binário compacto para a comunicação entre os nós, reduzindo o tráfego de rede e o tempo de serialização/deserialização dos dados (TRINO, 2021).

O *Trino* pode ser baixado gratuitamente no *site* oficial ou através do gerenciador de pacotes *pip*, fato que influenciou na escolha da feramenta. Ele será utilizado nesse projeto para consultas e análises de dados, permitindo acesso rápido e unificado a várias fontes de dados. Além disso a escolha do *Trino* também foi motivada por sua capacidade de realizar consultas rápidas e sua escalabilidade para lidar com ambientes de *Big Data.*

3.2.6 Apache Superset

O *Apache Superset* é uma plataforma de visualização de dados de código aberto que oferece uma interface rica e intuitiva para a exploração e análise de dados. Segundo a documentação oficial, o *superset* foi projetado para ser "visual, intuitivo e interativo", permitindo que os usuários criem *dashboards* e relatórios personalizados com facilidade. Uma das características distintivas dele é seu IDE (Ambiente de Desenvolvimento Interativo), que oferece uma interface amigável para a criação de consultas SQL e a visualização de dados. Além disso, o *superset* possui uma segunda interface que facilita a conversão de dados em recursos visuais, como gráficos e tabelas, de forma rápida e eficiente (SUPERSET, 2021).

O *superset* se integra perfeitamente com diversas fontes de dados, incluindo bancos de dados *SQL* e *NoSQL*, bem como outras ferramentas de análise de dados, como o *Trino* e o *Spark*. Isso é possível graças a uma arquitetura modular que permite a adição de novos conectores de dados conforme necessário (HARRIMAN, 2021).

Em termos de desempenho, o *superset* é altamente escalável e pode ser facilmente ajustado para atender às demandas de ambientes de *Big Data*. Ele utiliza técnicas de

cache e otimização de consultas para garantir uma entrega rápida e eficiente dos *dashboards* e relatórios (INFLUXDATA, 2021).

Para este projeto, a escolha do *Apache Superset* foi motivada por sua capacidade de fornecer visualizações de dados ricas e interativas, bem como sua integração fácil com outras ferramentas da *pipeline*, como o *Trino* e o *Airflow*. O *superset* será utilizado para a análise e visualização dos dados, permitindo que os usuários interajam com os dados de forma significativa e obtenham *insights* valiosos.

3.2.7 Apache Spark

O *Apache Spark* é uma plataforma de computação em *cluster* que se destaca por sua velocidade e versatilidade. Segundo Jean-Luca Bez (2015), o *Spark* oferece *APIs* de alto nível em várias linguagens de programação, como *Java*, *Scala*, *Python* e *R*. Além disso, ele suporta uma ampla gama de ferramentas de alto nível, incluindo *Spark SQL* para consultas *SQL*, *MLlib* para aprendizado de máquina, *GraphX* para processamento gráfico e *Spark Streaming* para processamento de dados em tempo real.

Quando se trata de escalabilidade, o Spark é construído para se expandir e se ajustar às demandas de ambientes de Big Data. Sua capacidade de se conectar a várias fontes de dados e sistemas de armazenamento o torna uma opção excelente para atividades de processamento de dados intrincadas.

A trajetória e desenvolvimento do Spark são dignos de nota. Desde o início, o projeto tem demonstrado uma expansão constante, apoiado por uma comunidade engajada e um ecossistema sólido formado por colaboradores e usuários (ZAHARIA et al., 2012).

Para a *pipeline*, a escolha do *Apache Spark* foi motivada por sua capacidade de lidar com uma variedade de tarefas de processamento de dados. Sua versatilidade e escalabilidade o tornam uma ferramenta complementar valiosa para as outras tecnologias da *pipeline*, como *Apache Airflow* e *Trino*.

A combinação dessas ferramentas permite uma abordagem integrada e eficiente para a construção de uma *pipeline* de dados robusta, capaz de lidar com os desafios do ambiente de *Big Data*. A utilização de soluções *open-source* também reforça o compromisso com a inovação e colaboração, alinhando-se com a filosofia de transparência e compartilhamento de conhecimento que permeia este projeto.

3.3. Implementação da Pipeline de Dados Utilizando as Ferramentas Open-Source

Para implementar o projeto, realizamos uma série de passos cuidadosos para assegurar que todos os elementos fundamentais estivessem adequadamente instalados e preparados para funcionar. A seguir, apresentamos o detalhamento do processo:

3.3.1 Ativação do WSL na Máquina e Instalação do Docker

No primeiro estágio do processo, ativamos o subsistema *WSL (Windows Subsystem for Linux)* no computador em que o projeto foi desenvolvido. Esse procedimento foi realizado usando o painel de controle, mais especificamente na seção "Funcionalidades do Windows", para garantir uma integração eficaz entre os sistemas operacionais *Windows* e *Linux*.

Após a bem-sucedida ativação do *WSL*, avançamos para a instalação do *Docker*. Este *software* desempenha um papel fundamental na arquitetura do projeto, visto que é responsável pela criação e pelo gerenciamento de contêineres. Estes contêineres hospedam ferramentas essenciais como o *Apache Airflow* e o *Apache Spark*, permitindo uma execução mais eficiente e isolada das tarefas. A instalação do *Docker* foi efetuada por meio de comandos

executados no terminal do *WSL*, assegurando assim uma integração sólida entre os diferentes componentes do sistema.

3.3.2 Download e Personalização do Docker Image do Airflow para Inclusão do Spark e Criação de Volumes no WSL

Após o passo anterior, a etapa seguinte consistiu no *download* do *Docker Image* específico para o *Apache Airflow*. Utilizamos comandos específicos no terminal para puxar a versão mais recente da imagem do *Airflow*, assegurando que teríamos acesso às funcionalidades mais atualizadas.

Antes de inicializar o contêiner do *Airflow*, efetuamos ajustes estratégicos no arquivo de configuração do *Docker*, com o objetivo de também instalar o *Apache Spark* durante a fase de inicialização do contêiner. Esse passo foi crucial para permitir a execução de *scripts Spark* dentro do ambiente *Airflow*, tornando o sistema mais robusto e versátil.

Para viabilizar a interação entre os *scripts do Spark* e o contêiner do *Airflow*, implementamos volumes de dados que servem como pontes entre as pastas localizadas no ambiente do *WSL* e as correspondentes dentro do contêiner. Essa configuração foi estabelecida com o uso do *Docker Compose*.

Esse arranjo garante uma integração harmoniosa e eficiente entre o *Airflow, o Spark* e o ambiente de trabalho no *WSL*, proporcionando uma arquitetura de sistema sólida e altamente funcional.

3.3.3 Execução de Scripts Spark Por Meio do Apache Airflow

Com a configuração bem-sucedida dos volumes de dados, transferimos os *scripts* do *Spark* para a pasta especificada no ambiente *WSL*. Isso possibilitou que esses *scripts* fossem não apenas acessíveis, mas também executáveis a partir do contêiner do *Apache Airflow*, integrando-se de forma transparente aos fluxos de trabalho automatizados.

Esta etapa estabelece um marco significativo na implementação do projeto, pois permite que tarefas complexas envolvendo o *Apache Spark* sejam gerenciadas e executadas de maneira eficaz através do *Apache Airflow*. Essa integração beneficia a robustez e a eficiência do sistema como um todo, simplificando a orquestração de tarefas e consolidando a arquitetura do projeto.

A integração foi projetada para ser escalável e manutenível, permitindo futuras atualizações e otimizações de *scripts* do *Spark* e de fluxos de trabalho no *Airflow*, sem exigir alterações complexas na estrutura existente.

3.3.4 Implementações de Script PySpark para Processamento de Dados

Na fase de implementação, um dos componentes críticos é o *script PySpark (Python + framework Spark*) destinado ao processamento de *logs* de arquivos no formato *JSON* e *CSV*. Além de manipular arquivos *JSON* e *CSV*, nossa solução é também capaz de extrair dados de bancos de dados relacionais. Isso permite uma maior abrangência na coleta de informações, unificando dados de diferentes naturezas e estruturas em uma única *pipeline*.

Para realizar essa tarefa, utilizamos conectores de banco de dados compatíveis com o *Apache Spark*. Estes conectores facilitam a extração de dados. Isso expande significativamente o escopo de fontes de dados que nossa *pipeline* pode acomodar, tornando nossa solução mais versátil e abrangente.

3.3.5 Otimização e Modularidade em Processos de Dados com PySpark

No contexto do nosso projeto, que utiliza PySpark, optamos por empregar conexões JDBC para interagir com bancos de dados relacionais. Embora o JDBC seja originalmente uma API Java, ele pode ser usado em PySpark através de uma interface Py4J, permitindo que

operações de banco de dados sejam realizadas em Python. A vantagem de usar JDBC é que ele fornece um método padrão e eficiente para se conectar a diferentes bancos de dados relacionais, tornando o código mais limpo e menos dependente do sistema de armazenamento de dados subjacente. Isso nos permite executar operações de CRUD (Criar, Ler, Atualizar, Deletar) e consultas SQL de maneira eficiente.

Adicionalmente, utilizamos a integração da biblioteca Argparse ao script. Essa ferramenta serve para tornar o script mais flexível e capaz de processar múltiplas tabelas com o mesmo código-base. Através de argumentos de linha de comando, conseguimos especificar qual tabela processar, qual estrutura de diretório seguir, entre outras variáveis. Isso eliminou a necessidade de 'hard coding' no script, tornando-o mais modular e fácil de manter.

Ao processar os logs, o script também adiciona uma coluna 'Date' ao DataFrame. Essa coluna é inserida com a data do momento que está sendo processados os dados. A partir dela, realizamos a extração do ano, do mês e da data, que funcionam como chaves de partição para otimizar futuras consultas de dados.

Finalmente, os dataFrames são armazenados no sistema de arquivos local e são particionados por 'Year', 'Month' e 'Date'. Essa estratégia de particionamento é crucial para otimizar o desempenho de leitura e escrita, mesmo em um sistema local, permitindo consultas mais rápidas e uma gestão de dados mais eficiente. Este modelo de armazenamento e particionamento local serve como uma prova de conceito que pode ser escalada para um ambiente distribuído quando o projeto for expandido.

3.3.6 DAG no Apache Airflow e Integração com Spark

No coração da implementação do *Apache Airflow* está o *DAG* (*Directed Acyclic Graph*), que serve como um plano para a execução e orquestração das nossas tarefas de processamento de dados. Neste projeto, utilizamos o *BashOperator* do *Airflow* para invocar comandos *Spark-submit*, que por sua vez, executam nossos *scripts Spark*.

O *BashOperator* permite uma integração natural com o *Spark*, visto que o *Spark-submit* é geralmente invocado a partir de uma linha de comando. Com essa abordagem, conseguimos acionar os *jobs Spark* diretamente do *Airflow*, fornecendo uma solução simplificada, mas poderosa para a nossa arquitetura de processamento de dados.

Os argumentos necessários para rodar os *scripts Spark* são passados dinamicamente por meio do *Airflow*. Isso inclui informações como o nome da tabela a ser processada, o caminho de armazenamento, entre outros. Essa estratégia nos permite manter o *script Spark* genérico e reutilizável, já que todos os parâmetros específicos são injetados no momento da execução.

Dessa forma, conseguimos uma integração robusta e eficiente entre o *Apache Airflow* e o *Apache Spark*, com a flexibilidade de adaptar nossos *pipelines* de dados às necessidades específicas do projeto, tudo isso orquestrado de forma automatizada e confiável.

3.3.7 Desenvolvimento da Camada RAW e Utilização de Volumes Docker

A primeira etapa do nosso processo de ingestão e transformação de dados é povoar a camada *raw*. O *script Spark* que desenvolvemos é projetado para processar arquivos de entrada e armazená-los nesta camada. Os arquivos são mantidos em seu estado original, servindo como a fonte para todas as operações de dados subsequentes.

Uma característica importante da nossa implementação é o uso de volumes *Docker*, que foram criados durante a fase de configuração do nosso ambiente. Ao armazenar os arquivos na camada *raw* dentro deste volume, conseguimos um alto grau de persistência e flexibilidade. Mesmo que o container do *Docker* seja encerrado ou removido, os dados permanecem intactos, permitindo uma retomada eficiente das operações.

O uso de volumes *Docker* também facilita a troca de dados e acesso entre diferentes serviços e componentes do nosso sistema, como o *Apache Hive* e *Apache Trino*, sem a necessidade de cópias redundantes. Isso otimiza o uso do armazenamento e simplifica a arquitetura de dados. Portanto, a integração do *Docker* na nossa estratégia de armazenamento para a camada *raw* não só fornece uma solução de armazenamento robusta, mas também, adiciona uma camada extra de flexibilidade e eficiência ao nosso *pipeline* de dados.

3.3.7 Camada Refined e Uso de Temp View

Na camada *refined* da *pipeline* de dados, focamos em otimizar os dados para análise e consultas rápidas. Uma das técnicas que adotamos é pegar os arquivos *Parquet* da camada *raw* e usar um recurso chamado *temp view* no *PySpark*.

Um *temp view* permite criar uma "vista temporária" de um *DataFrame*, que pode então ser consultada usando a sintaxe SQL padrão. Isso é incrivelmente útil por vários motivos:

- Democratização dos Dados: SQL é uma linguagem conhecida por muitos profissionais que talvez não estejam familiarizados com *Python* ou *PySpark*. Permitir que esses usuários consultem dados usando *SQL* abre as portas para uma ampla gama de profissionais envolvidos em análise de dados.
- Facilidade de Uso: Muitas pessoas acham mais intuitivo elaborar filtros e consultas complexas em *SQL* do que usando a *API* de *DataFrame* do *PySpark*. Isso pode acelerar o desenvolvimento e tornar o código mais acessível e mais fácil de entender para aqueles que estão acostumados com *SQL*.
- Integração com Outras Ferramentas: A disponibilidade de uma vista *SQL* torna mais fácil integrar com outras ferramentas que podem consumir dados via *SQL*, fornecendo assim uma camada adicional de flexibilidade ao sistema.

Portanto, ao criar um *temp view* dos nossos *DataFrames*, ganhamos a capacidade de realizar consultas *SQL* sobre nossos dados, tornando o ambiente mais inclusivo e facilitando a elaboração de filtros e análises.

3.3.8 Otimização em Leitura de Dados e Gestão do Ambiente de Análise

A leitura dos arquivos Parquet é uma etapa vital em nosso pipeline, desempenhando um papel fundamental tanto na camada raw quanto na camada *refined*. Conhecidos por sua excelência em consultas, os arquivos Parquet são otimizados para trabalhar com sistemas de armazenamento em coluna. Esta característica assegura que a leitura e análise sejam realizadas de forma veloz e eficiente.

Com o objetivo de simplificar a configuração e gestão do nosso ambiente de análise de dados, optamos por utilizar containers Docker, que incorporam o Apache Hive MetaStore, Trino e Apache Superset. Esta abordagem proporciona uma implantação ágil e padronizada das ferramentas, facilitando o gerenciamento da infraestrutura e minimizando possíveis erros de configuração.

3.3.9 Hive MetaStore, Criação de Tabelas e Atualização de Partições

O *Apache Hive MetaStore* serve como um repositório centralizado para metadados, o qual foi utilizado para criar tabelas que apontam para os arquivos *Parquet* na camada *raw* e *refined*. Estas tabelas no *Hive MetaStore* fornecem uma *interface* mais amigável e estruturada para interagir com os dados armazenados em formato *Parquet*, otimizando ainda mais o desempenho das consultas.

Um aspecto crucial desse arranjo é a necessidade de manter o *Hive MetaStore* atualizado com as novas partições à medida que são adicionados novos arquivos *Parquet*.

Sempre que novos dados são adicionados às camadas *raw* ou *refined*, precisamos atualizar as metainformações no *Hive MetaStore* para que ele possa identificar as novas partições. Isso é fundamental para que as consultas sejam eficientes e para garantir que todo o conjunto de dados esteja disponível para análise.

Essa prática assegura que o sistema esteja sempre sincronizado com os dados mais recentes, permitindo que as operações de leitura sejam otimizadas. Isso também torna o sistema mais robusto e ágil, garantindo que as informações mais atuais estejam sempre disponíveis para análise e tomada de decisão.

3.3.10 Utilização do Trino e Integração com Hive

Utilizamos o *Trino* para possibilitar a execução de consultas *SQL* de alto desempenho em nosso ambiente de dados. Este motor de consulta é especialmente útil para realizar análises *ad hoc* em grandes volumes de dados armazenados em várias fontes, incluindo o *Hive MetaStore*.

Para garantir a interoperabilidade eficiente entre *Trino* e *Hive*, é necessário o uso de um *plugin* específico que facilita a comunicação entre as duas plataformas. Esse *plugin* já foi instalado durante a fase de configuração do nosso ambiente *Docker*, o que significa que o sistema já está apto para realizar consultas cruzadas entre diferentes armazenamentos de dados de maneira otimizada e eficiente.

O uso do *plugin* assegura que podemos acessar e consultar os metadados armazenados no *Hive MetaStore* diretamente através do *Trino*.

3.3.11 Apache Superset e Conectividade com Trino

O *Superset* atua como a camada final onde os dados refinados e processados são representados de forma gráfica e interativa, possibilitando *insights* mais profundos e decisões orientadas por dados.

Para que o *Apache Superset* possa acessar os dados armazenados e gerenciados pelo *Hive* e *Trino*, é necessária uma conexão especializada entre essas plataformas. Nesse sentido, instalamos um *plugin* específico para *SQLAlchemy* que facilita essa conectividade com o *Trino*. Este *plugin* é vital para assegurar que as consultas *SQL* sejam transmitidas eficientemente através do *Trino* e que os resultados possam ser visualizados no *Apache Superset*.

Essa instalação e configuração foi feita como parte da nossa *stack* de *containers Docker*, garantindo que todas as ferramentas se integrem de forma transparente e eficiente. Ao utilizar *SQLAlchemy* com o *plugin* apropriado, conseguimos uma interação robusta e eficiente entre o *Superset* e o *Trino*, permitindo que análises avançadas e *dashboards* de alta qualidade sejam criados com facilidade.

4 Resultados e Discussão

O escopo desta seção inclui uma avaliação da eficiência, eficácia e escalabilidade da *pipeline*. Também discutiremos as limitações e desafios enfrentados durante a implementação, bem como as soluções adotadas para superar esses obstáculos. Além disso, vamos explorar o impacto desses resultados no campo de estudo relevante e como eles contribuem para o avanço do conhecimento ou para a solução de problemas práticos.

Ao fornecer uma análise detalhada, esta seção visa não apenas validar a qualidade e relevância do trabalho realizado, mas também servir como um recurso informativo para outros pesquisadores ou profissionais interessados em desenvolver ou adaptar uma *pipeline* similar.

O sucesso de qualquer projeto de dados é medido não apenas pela eficiência da coleta e armazenamento de dados, mas também pela eficácia com que esses dados podem ser analisados e interpretados para *insights* úteis.

4.1. Reutilização de Código

Aqui, a flexibilidade e reutilização do código desempenham um papel fundamental, e é nesse contexto que a biblioteca *argparse* mostrou seu valor.

Ao incorporar a biblioteca *argparse* em nossos *scripts PySpark*, pudemos criar um conjunto de parâmetros flexíveis que tornaram nossos códigos muito mais dinâmicos e reutilizáveis. Isso permitiu que o mesmo código-base fosse aplicado a diferentes tabelas, estruturas de dados e fontes, simplesmente alterando os argumentos passados ao *script*. Dessa forma, evitamos o "*hard-coding*" e tornamos nosso projeto mais escalável e fácil de manter.

4.2. Dags Unificadas para Diversos Pipelines

Outro resultado significativo foi a abordagem para a criação de *Directed Acyclic Graphs* (*DAGs*) no *Airflow*. Foram criados *templates* de *DAGs* tanto para o processamento da camada *raw*, que lida com *logs* nos formatos *JSON* e *CSV*, quanto para bancos de dados relacionais. Da mesma forma, temos um template de *DAG* para o *refinamento* dos dados. Esses *templates* são considerados independentes em relação à fonte de dados e à estrutura da tabela, o que significa que criar uma nova *pipeline* se resume basicamente a configurar novos argumentos, como o nome da tabela, entre outros. Isso não apenas simplifica a manutenção, mas também acelera o processo de adição de novas fontes de dados ao nosso *pipeline*, já que os esforços de desenvolvimento são drasticamente reduzidos.

Ao manter um modelo unificado, conseguimos também garantir que as melhores práticas são consistentemente aplicadas em todas as etapas do processo, melhorando assim a qualidade e a confiabilidade dos dados processados.

4.3. Eficiência na Leitura de Dados com Trino e Otimização Utilizando Parquet e Partições

Uma das grandes vantagens observadas na *pipeline* desenvolvida, foi a eficiência quanto a leitura de dados proporcionada pelo *Trino* em conjunto com o uso de arquivos *Parquet*. O *Trino*, um mecanismo de consulta distribuída, permitiu a execução de consultas *SQL* complexas de forma muito mais rápida, graças à sua capacidade de realizar otimizações de plano de consulta. Esta eficiência foi ainda mais acentuada quando os dados foram armazenados no formato *Parquet*, que é otimizado para sistemas de armazenamento em coluna e oferece vantagens significativas em termos de compressão e velocidade de leitura.

O uso de partições em arquivos *Parquet* também contribuiu para a eficiência na leitura de dados. Ao utilizar partições como filtros em nossas *queries*, pudemos direcionar o *Trino* para ler apenas os blocos de dados específicos necessários para responder a uma consulta, em vez de ler todo o arquivo. Isso resultou em uma melhoria significativa na eficiência e na velocidade das consultas, permitindo que operações mais complexas fossem realizadas em menos tempo e com menor uso de recursos.

4.4. Desafios e Aprendizados na Implementação e Configuração

Na jornada de implementação das ferramentas, enfrentamos desafios significativos, particularmente ao configurar o Hive Metastore. A documentação existente não era tão elucidativa quanto desejávamos, nos conduzindo a um ciclo de tentativa e erro para atingir a configuração ideal. Embora isso tenha prolongado a configuração, o aprendizado adquirido é inestimável e será valioso para futuras iniciativas.

Adicionalmente, ao utilizar um computador avançado para gerenciar nossos containers, percebemos um consumo substancial de recursos ao ativar as três instâncias do Docker simultaneamente através do Docker-compose. Esta experiência nos proporcionou uma perspectiva dos recursos necessários ao pensar na expansão deste projeto para um ambiente produtivo mais extenso.

4.5. Comparativo Open-Source vs Open-Core

Ao implementar as ferramentas de código aberto, mesmo considerando sua acessibilidade financeira, nos deparamos com a necessidade de um estudo detalhado da documentação e uma configuração manual mais exigente do ambiente da *pipeline*. Foi essencial compreender que, em alguns casos, seria imperativo investir em máquinas e infraestrutura mais robustas conforme o uso específico da nossa *pipeline*.

Em contrapartida, nossa experiência anterior em empresas de grande porte nos conferiu uma experiencia na utilização de ferramentas gerenciadas. Apesar de apresentarem menor customização, essas alternativas se destacam por sua intuitividade e pronto uso, o que nos acostumou com a praticidade das soluções *open-core*.

O desenvolvimento deste projeto, embora exigente em termos de esforço e curva de aprendizado, proporcionou-nos valiosa autonomia ao não dependermos do gerenciamento da infraestrutura por terceiros.

5 Conclusão

O desenvolvimento e implementação da *pipeline* de dados em ambiente de *Big Data*, baseada exclusivamente em tecnologias de código aberto, representa um marco significativo na busca por alternativas economicamente viáveis para o gerenciamento de grandes conjuntos de dados. Os resultados obtidos evidenciam não apenas a eficiência, mas também a escalabilidade e a flexibilidade proporcionadas por essa abordagem.

A reutilização de código, facilitada pela integração da biblioteca *argparse*, demonstrou ser um elemento-chave na dinamização e adaptabilidade dos *scripts PySpark*, eliminando a necessidade de "*hard-coding*" e promovendo a manutenção simplificada do projeto. Além disso, a implementação de *DAGs* unificadas no *Airflow* não só simplificou a adição de novas fontes de dados, como também garantiu a aplicação consistente das melhores práticas em todas as etapas do processo.

Destaca-se ainda a eficiência na leitura de dados, impulsionada pela combinação do *Trino* e do formato *Parquet*, que resultou em ganhos expressivos de desempenho. O uso de partições em arquivos *Parquet* se revelou uma estratégia eficaz para otimizar a leitura de dados, direcionando consultas de forma precisa e economizando recursos.

Entretanto, é importante mencionar as dificuldades encontradas na configuração do *Hive Metastore*, um desafio que, embora tenha demandado tempo e esforço, proporcionou valiosos aprendizados para projetos futuros. Da mesma forma, a observação do consumo considerável de recursos durante a implantação dos *containers* alerta para a necessidade de planejamento meticuloso ao escalar o projeto para ambientes de produção mais amplos.

Em síntese, os resultados alcançados nesta pesquisa demonstram que a adoção de tecnologias de código aberto é não apenas economicamente viável, mas também técnica e conceitualmente robusta para o gerenciamento eficiente de grandes volumes de dados em ambientes de *Big Data*. Esta abordagem oferece uma alternativa acessível e eficaz para organizações de menor porte ou projetos com recursos limitados, contribuindo significativamente para o avanço do conhecimento e para a solução de desafios práticos no campo do gerenciamento de dados em larga escala.

6 Referências Bibliográficas

APACHE AIRFLOW. **Documentação.** Disponível em: https://Airflow.apache.org/docs/ Acesso em: 04 ago. 2023.

AWS. **O que é processamento em lote?** Disponível em: https://aws.amazon.com/pt/what-is/batch-processing/. Acesso em: 03 set. 2023.

BEZ, Jean Luca. **Plataformas de Big Data: Spark, Storm e Flink**. Universidade Federal do Rio Grande do Sul, 2015. Disponível em: https://www.researchgate.net/publication/280052790_Plataformas_de_Big_Data_Spark_Storm_e_Flink. Acesso em: 20 set. 2023.

CAPRIOLO, E., RUTHERGLEN, J., & WAMPLER, D. **Programming Hive**. O'Reilly Media. 2012

CIÊNCIA E DADOS**. Data Lake: A evolução do armazenamento e processamento de dados.** 2018. Disponível em: https://www.cienciaedados.com/data-lake-a-evolucao-do-armazenamento-e-processamento-de-dados Acesso em: 05 ago. 2023.

DAMJI, J. et al. **Learning Spark: Lightning-Fast Data Analytics.** O'Reilly Media, 2020.

DE RUITER, Julian; HARENSLAK, Bas. **Data Pipelines with Apache Airflow.** Manning, 2021.

GORELIK, A. **The Enterprise Big Data Lake**. 2019.

HARRIMAN. **Ferramentas de visualização de dados: uma solução com Apache Superset para o Projeto RADIS-UFMT.** UFMT, Cuiabá, 2021. Disponível em: https://engenhariadecomputacao.cba.ifmt.edu.br/media/filer_public/18/0b/180b6998-4499-4cd8-bcf0-46e0f110c5f8/tcc_harriman_fonte.pdf Acesso em: 20 ago. 2023.

HASSAN. **What Is a Data Pipeline in Big Data?** Disponível em: https://turbofuture.com/computers/What-is-Data-Pipeline-in-Big-Data Acesso em: 11 jul. 2023.

IBM**. What is a data pipeline.** Disponível em: https://www.ibm.com/topics/data-pipeline Acesso em: 10 jul. 2023.

INFLUXDATA. **Introduction to Apache Superset.** 2021. Disponível em: https://www.influxdata.com/blog/introduction-apache-superset. Acesso em: 20 ago. 2023.

KIMBALL, R.; CASERTA, J. **The Data Warehouse ETL Toolkit.** Wiley Publishing, 2004.

KIMBALL, R. et al. **The data warehouse lifecycle toolkit**. 2. ed. Wiley, 2008.

MACHADO, F. N. R. **Big Data: o futuro dos dados e aplicações.** São Paulo: Saraiva Educação S.A., 2018.

MARQUES, F.; FRANÇA, B. **Qualitative Data Analytics: Desenvolvimento de ferramenta para análise qualitativa de dados. 2022. 22 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)** – Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2022. Disponível em: https://ic.unicamp.br/~reltech/PFG/2020/PFG-20-32.pdf Acesso em: 20 set. 2023.

MARQUESONE, Rosangela. **Big Data: Técnicas e tecnologias para extração de valor dos dados**. Casa do Código, 2016. eBook Kindle.

MARZ, N.; WARREN, J. **Big data: princípios e melhores práticas de sistemas de dados em tempo real escaláveis.** Manning Publications, 2015.

NEWMAN, Sam. **Building Microservices.** O'Reilly Media, 2015.

OPEN SOURCE INITIATIVE**. The Open Source Definition.** 2007. Disponível em: https://opensource.org/osd/ Acesso em: 24 maio 2023.

OPENSOURCE.COM. **What is open source?** Disponível em: https://opensource.com/resources/what-open-source Acesso em: 24 maio 2023.

PERENS, Bruce. **The Open Source Definition**. In: DI BONA, Chris; OCKMAN, Sam; STONE, Mark (Eds.). Open sources: voices from the open source revolution. 1. ed. Sebastopol: O'Reilly Media, 1999. p. 79-86.

PROVOST, F.; FAWCETT, T. **Data Science for Business**. O'Reilly Media, 2013.

SIMON, R Allan. **Data Lakes for Dummies**. 1ª ed. For Dummies, 25 jun. 2021.

TURNBULL, James. **The Docker Book: Containerization is the New Virtualization.** 2014.

TRINO. **Documentação.** Disponível em: https://trino.io/docs/ Acesso em: 04 ago. 2023.

WHITE, T. **Hadoop: The Definitive Guide.** O'Reilly Media, 2015.

ZAHARIA, M. et al. **Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. 2012.** Disponível em: https://dl.acm.org/doi/10.5555/2228298.2228301 Acesso em: 20 ago. 2023.