Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

LUCAS BARBOSA BRANCALHÃO

ESTUDOS ESTRUTURAIS EM BIG DATA

Assis/SP
2024

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

LUCAS BARBOSA BRANCALHÃO

ESTUDOS ESTRUTURAIS EM BIG DATA

Trabalho de Conclusão de Curso apresentado ao curso de Bacharel em Ciência da Computação do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando: Lucas Barbosa Brancalhão
Orientador: Almir Rogério Camolesi

Assis/SP
2024

Brancalhão, Lucas Barbosa

B816e Estudos estruturais em big data / Lucas Barbosa Brancalhão. -- Assis, 2024.

47p. : il.

Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) -- Fundação Educacional do Município de Assis (FEMA), Instituto Municipal de Ensino Superior de Assis (IMESA), 2024.
Orientador: Prof. Dr. Almir Rogério Camolesi.

1. Ciência de dados. 2. Repositórios digitais. 3. Python. I Camolesi, Almir Rogério. II Título.

CDD 004

Elaborada por Anna Carolina Antunes de Moraes – Bibliotecária – CRB-8/10982

ESTUDOS ESTRUTURAIS EM BIG DATA

LUCAS BARBOSA BRANCALHÃO

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, avaliado pela seguinte comissão examinadora:

Orientador:	
	Almir Rogério Camolesi
Examinador:	
	Célio Desiró

Assis/SP
2024

DEDICATÓRIA

Dedico esse trabalho a Deus, minha família e meus amigos.

AGRADECIMENTOS

Agradeço a faculdade e a todos os que me incentivaram a continuar meus estudos.

RESUMO

Este projeto de conclusão de curso explora a integração de *Apache Spark*, *Terraform* e *AWS Glue* para a gestão eficiente de grandes volumes de dados com a utilização de um *Data Lake*. Ele detalha o *Apache Spark* e sua relação com *Hadoop*, o uso de *Terraform* para infraestrutura como código, e serviços da *AWS* como *S3*, *Glue* e *Athena* para processamento e catalogação de dados.

Também são discutidas arquiteturas de dados, como *Pipes and Filters* e o modelo *Medallion*. A implementação prática usa *Terraform* para conectar à nuvem e automatizar processos com *AWS Glue*. O projeto reflete sobre a relevância dessas tecnologias no cenário atual de *Big Data*.

Palavras-chave: 1. Data Lake. 2. Dados. 3. Ciência de Dados. 4. Python. 5. Big Data. 6. Análise. 7. Nuvem. 8. Arquitetura. 9. Terraform.

ABSTRACT

This final project explores the integration of Apache Spark, Terraform, and AWS Glue for the efficient management of large volumes of data using a Data Lake. It details Apache Spark and its relationship with Hadoop, the use of Terraform for infrastructure as code, and AWS services such as S3, Glue, and Athena for data processing and cataloging.

Data architectures such as Pipes and Filters and the Medallion model are also discussed. The practical implementation uses Terraform to connect to the cloud and automate processes with AWS Glue. The project reflects on the relevance of these technologies in the current big data landscape.

Keywords: 1. Data Lake 2. Data 3. Data Science. 4. Python. 5. Big Data. 6. Analytics. 7. Cloud. 8. Architecture. 9. Terraform

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

1. **GAD (GRAFO ACÍCLICO DIRECIONADO)**

2. **AMAZON S3 (SIMPLE STORAGE SERVICE)**

3. **API (APPLICATION PROGRAMMING INTERFACE – INTERFACE DE PROGRAMAÇÃO DE APLICAÇÃO)**

4. **ETL (EXTRACT, TRANSFORM, LOAD – EXTRAIR TRANSFORMAR E CARREGAR)**

Sumário

1. INTRODUÇÃO

Trocar informações e aprender sempre foi um aspecto importante de todas as pessoas. Acessamos nossas redes sociais. Aprendemos a conversar, por saber mais sobre outros e às vezes sobre nós mesmos, ou quando vamos trabalhar, trocamos informações com nossos colegas, e aprendendo mais sobre como agir melhor em nosso trabalho, gerando mais eficiência em diferentes áreas do nosso cotidiano.

Pensando nisso, por um olhar mais abrangente, hoje em dia, os dados e informações se tornaram de grande importância, sendo considerados de extremo valor, gerando insights, trazendo mais eficiência para produções e empresas. O *Big Data* é uma realidade, sendo utilizado grandemente na ciência de dados em análises. Além disso, com o advento do *IOT (Internet of Things)*, *Big Data* pode também vir de objetos e itens como carros, caminhões, lâmpadas, relógios, entre outros.

Segundo *Oracle* (2023), destaca-se que nos últimos anos, pessoas e objetos têm gerado quantidades massivas de dados, tendo essa quantidade dobrado a cada dois anos. 50 a 80 por cento do tempo de analistas de dados gastam o seu tempo curando dados. Algumas tecnologias populares para o manuseio de grandes massas de informações são o *Apache Hadoop*, um software de armazenamento massivo para qualquer tipo de dado, o Import.io, uma plataforma que serve para para extrair dados *open-source*, sem precisar digitar códigos de acesso, se tornando um grande banco de dados, o *Oracle Data Mining*, sendo um software para mineração e “peneira” de dados e o *Apache Spark*, um mecanismo de processamento de dados de software livre para grandes conjuntos de dados

Dada essa necessidade, o trabalho com aplicações escaláveis que possam manusear informações de forma eficiente se torna cada vez mais necessário e importante. Estruturas como *Data Lakes*, *Data Warehouses*, entre outros, tem se tornado cada vez mais comuns no mercado, e cada vez mais a estruturação desses sistemas tem se mostrado de grande valor.

1. 1. OBJETIVOS GERAIS

O presente trabalho tem como objetivo realizar um estudo de caso, criando uma estrutura simples e abstrata para a criação e execução de um *Data Lake* utilizando a nuvem, detalhando as ferramentas utilizadas para o desenvolvimento, como Terraform, AWS e seus serviços específicos, como S3, Athena, Glue e Glue Crawler. Assim com padrões arquiteturais de uma forma clara e objetiva para um desenvolvimento eficiente.

1. 2. OBJETIVOS ESPECÍFICOS

Como objetivo específico, foram definidos os seguintes critérios para a conclusão:

- Definição e desenvolvimento de uma estrutura simples, coesa e funcional da aplicação.
- Detalhamento de arquiteturas.
- Detalhamento de ferramentas utilizadas no processo de criação da aplicação.
- Apresentação de uma aplicação simples, abstrata e funcional, permitindo a execução em diferentes cenários.

1. 3. JUSTIFICATIVA

O trabalho se justifica, pois, a crescente demanda na área de dados tem cada vez mais se agravado, crescendo exponencialmente nas últimas décadas. Nesse fluxo, diversas empresas têm dedicado cada vez mais seus esforços para gerar valor para os negócios com essas informações. Além disso, a aplicação aqui proposta gera valor para a engenharia de software, já que detalha os passos necessários, além das boas práticas para o desenvolvimento.

1. 4. MOTIVAÇÕES

A motivação se dá pelo estudo técnico de processos de desenvolvimento e arquiteturais de sistemas de dados, e pelo estudo de ferramentas que permitem o desenvolvimento desses mesmos sistemas. Esses mesmos sistemas podem permitir que uma determinada empresa possa ter maior controle de todas as fases da sua produção, reconhecer falhas em tempo real, minimizar perdas e gerar mais receita a médio e longo prazo.

1. 5. PERSPECTIVAS DE CONTRIBUIÇÃO

Além da contribuição pessoal, espera-se que o presente trabalho possa contribuir de forma positiva na área de dados, mesmo que de forma simbólica, realizando um estudo interessante, embora não novo, para colaborar com a solução de um problema arquitetural e rastreio do processo de dados como um todo, desde a ingestão de dados brutos, até a distribuição dos dados refinados e tratados, gerando mais receita para uma determinada empresa ou pessoa.

1. 6. METODOLOGIA

A metodologia empregada para o presente trabalho consistirá das seguintes etapas:

- Levantamento bibliográfico e estudo exploratório para o desenvolvimento do *Data Lake*.
- Análise das ferramentas levantadas, e detalhamento teórico delas.
- Fase de implementação do projeto, usando o ecossistema de nuvem AWS. Ele provê aplicações para cada momento do desenvolvimento, evitando o uso de aplicações de terceiros.
- Avaliação dos resultados obtidos.
- Conclusão

1.7. ESTRUTURA DO TRABALHO

A introdução apresenta os objetivos e as motivações do trabalho, bem como a justificativa para a escolha do tema. A seguir, o capítulo 2 aprofunda o conceito de *Apache Spark*, detalhando sua arquitetura, a história da ferramenta, sua relação com o *Hadoop*, os benefícios e os principais casos de uso no contexto de Big Data.

O capítulo 3 aborda a nuvem, explicando suas aplicações e apresentando o *Terraform*, uma ferramenta de infraestrutura como código, destacando seu funcionamento e componentes. Também é introduzido o *AWS* (*Amazon Web Services*), com foco nos serviços S3, *Glue* e *Athena*, descrevendo como essas ferramentas são usadas para armazenar, catalogar e processar grandes volumes de dados.

Em seguida, o capítulo 4 trata das diferentes arquiteturas de dados, como *Pipes and Filters* e Medalhão, essenciais para organizar e processar os dados de forma estruturada.

No desenvolvimento, o capítulo 5 descreve a implementação das funcionalidades utilizando o *Terraform*, destacando a conexão com a nuvem, a configuração do *AWS Glue*, das camadas de dados do modelo Medalhão, e a automação dos processos com *Glue Crawler* e *jobs*.

Finalmente, o TCC é concluído com uma reflexão sobre a aplicabilidade dessas tecnologias e sua importância no cenário atual de gerenciamento de dados em grande escala.

2. O QUE É DATA LAKE?

Um data lake é um repositório central que armazena grandes volumes de dados em seu formato original, permitindo que eles sejam processados e utilizados para uma variedade de necessidades analíticas. Sua arquitetura flexível e escalável é capaz de acomodar dados de diferentes tipos e fontes, desde dados estruturados (como tabelas de banco de dados e planilhas), até semiestruturados (como arquivos XML e páginas da web), e dados não estruturados (como imagens, áudios e postagens em redes sociais). Esses dados são geralmente armazenados em diferentes zonas – bruta, limpa e organizada – permitindo que diferentes perfis de usuários acessem os dados de acordo com suas necessidades específicas. Os data lakes possibilitam a consistência dos dados entre diversos aplicativos e são essenciais para análises de Big Data, aprendizado de máquina, análises preditivas e outras formas de inteligência empresarial. (Microsoft, O que é um data lake?, 2024)

No ambiente atual, que é cada vez mais orientado por dados, os data lakes desempenham um papel crucial ao oferecer uma solução centralizada para armazenar, integrar e proteger grandes volumes de dados, muitas vezes usando plataformas robustas como o Azure Data Lake. Essas soluções eliminam a fragmentação dos dados e oferecem um armazenamento escalável e acessível, permitindo que as empresas utilizem esses dados em diversas cargas de trabalho, incluindo processamento de Big Data, consultas SQL, análise de streaming e aprendizado de máquina. Além disso, esses dados alimentam sistemas de visualização e relatórios personalizados. Plataformas como o Azure Synapse Analytics possibilitam uma abordagem completa para gerenciar arquiteturas de Big Data em torno de data lakes. (Microsoft, O que é um data lake?, 2024)

Casos de Uso *Data Lake:* Com uma solução bem planejada, o uso de *data lakes* pode levar à inovação em diversos setores. Alguns exemplos incluem:

- ***Streaming* de mídia**: Plataformas de streaming usam *data lakes* para analisar o comportamento dos usuários e melhorar algoritmos de recomendação. (Microsoft, O que é um data lake?, 2024)
- **Finanças**: Bancos de investimento armazenam dados de mercado em tempo real para otimizar a gestão de riscos de portfólio. (Microsoft, O que é um data lake?, 2024)

- **Saúde**: Hospitais utilizam grandes volumes de dados históricos para melhorar o atendimento e reduzir custos. (Microsoft, O que é um data lake?, 2024)
- **Varejo omnicanal**: Varejistas consolam dados de diferentes pontos de interação, como dispositivos móveis, redes sociais e interações presenciais, para melhorar a experiência do cliente. (Microsoft, O que é um data lake?, 2024)
- ***Internet das Coisas (IoT)***: Sensores emitem grandes quantidades de dados que são centralizados em um *data lake* para análises futuras. (Microsoft, O que é um data lake?, 2024)
- **Cadeia de fornecimento digital**: Fabricantes integram dados de vários formatos, como *EDI*, *XML* e *JSON*, em um único repositório. (Microsoft, O que é um data lake?, 2024)
- **Vendas**: Cientistas de dados criam modelos preditivos a partir de dados em *data lakes* para entender o comportamento do consumidor e reduzir a rotatividade. (Microsoft, O que é um data lake?, 2024)

3. ARQUITETURA *SPARK*

Apache Spark é uma *engine* de computação unificada e um conjunto de bibliotecas para o processamento paralelo de dados em *clusters*. Spark é uma das *engines* mais desenvolvidas do mercado, *open source*, se tornando uma ferramenta padrão para a manipulação de *Big Data*. Spark suporta múltiplas linguagens de programação, sendo elas: *Python*, *Java*, *Scala* e *R*. Também inclui bibliotecas que permitem a manipulação, desde *SQL* até *streaming* e *machine learning*, podendo ser executado em uma grande quantidade de máquinas, desde computadores pessoais, até centenas de servidores empresariais. (O'Reilly Media, Learning Spark: Lightning-Fast Data Analytics. 2020)

Quando pensamos em computação, normalmente pensamos em apenas uma máquina, que podemos usar para acessar a internet, fazer pesquisas, nos divertir, e trabalhar. Normalmente, para fins básicos, apenas um computador já basta, mas, quando precisamos realizar tarefas mais complexas, como por exemplo, trabalhar com muitos dados brutos, minerando e processando para que possamos tirar conclusões deles, um único computador já não basta. Precisamos de mais poder computacional. E é exatamente isso que conseguimos atingir ao utilizarmos vários computadores ao mesmo tempo. Podemos combinar o poder de vários computadores para que uma operação específica seja executada em menos tempo. (O'Reilly Media, Learning Spark: Lightning-Fast Data Analytics. 2020)

Um grande problema que podemos encontrar é a organização desses computadores. Assim como uma equipe empresarial coordenada precisa trabalhar junto para que entregue bons resultados, computadores precisam trabalhar juntos para que bons resultados sejam entregues. É aqui que o *Apache Spark* entra, coordenando vários computadores, ou como podemos chamar nesse caso, *clusters*, para que eles possam trabalhar juntos em sincronia. (O'Reilly Media, Spark: The Definitive Guide: Big Data Processing Made Simple. 2018)

Para isso, criamos aplicações *Spark*, que enviaremos para um gerenciador de *cluster* interno, como o *Spark Standalone Cluster Manager* (Gerenciador de Cluster Autônomo do Spark), o *YARN* ou *Mesos*, que nos darão os recursos necessários para que o trabalho seja feito. (O'Reilly Media, Learning Spark: Lightning-Fast Data Analytics. 2020)

As aplicações Spark funcionam da seguinte forma: Um processo pai fica alojado em um nó do *Cluster*, rodando sua função *main()*, e é responsável por três coisas:

1. **Manter informações sobre a aplicação *Spark*.**
2. **Responder as entradas de um usuário ou de um programa.**
3. **Analisar, distribuir e agendar trabalhos pelos executores.**

Aqui fica concentrado o coração de uma aplicação *Spark* e aqui ficam contidas todas as informações relevantes do projeto. Podemos ver mais sobre a arquitetura na figura abaixo:

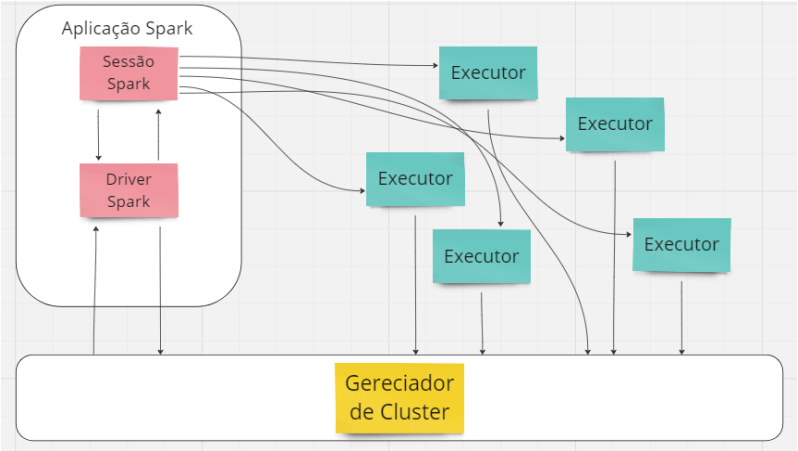

Figura 1: Arquitetura Interna *Spark*.

Fonte: Imagem criada pelo software Miro (Learning Spark: Lightning-Fast Data Analytics. 2020)

Acima podemos entender melhor como o *Spark* funciona “por baixo dos panos”.

Spark Driver: É a parte da aplicação *spark* responsável por instanciar a Seção *Spark*. O *Driver Spark* tem vários papéis: Se comunica com o gerenciador de *clusters*, requisita recursos (CPU, Memória etc.) do gerenciador de clusters para os executores e transforma todas as operações *Spark* em computações no formato GAD (Grafo Acíclico Direcionado), os agenda para serem executados, e distribui as suas execuções por todos os executores. Uma vez que os recursos estão todos alocados, o *Driver Spark* se comunica direto com os executores. (O'Reilly Media, Learning Spark: Lightning-Fast Data Analytics. 2020)

Seção Spark: A partir da versão 2.0 do *Spark*, a Seção *Spark* se tornou um condutor de todas as operações *Spark*, o que tornou o trabalho mais simples e fácil, agregando em si mesmo outros contextos de entrada, como *SparkContext*, *SQLContext*, *HiveContext*, *SparkConf* e o *StreamingContext*. (Mesmo que ainda seja possível acessar cada um desses contextos individualmente). Em aplicações *Spark standalone*, é possível criar seções em

APIs de alto nível na linguagem de programação da preferência do usuário, sendo possível criar várias operações *Spark*. Abaixo um exemplo de como criar uma seção *spark*.

```
# Importar as bibliotecas necessárias
from pyspark.sql import SparkSession

# Criar uma sessão Spark
spark = SparkSession.builder \
    .appName("Ler Tabela de Produtos") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

# Ler os dados da tabela de produtos
produtos_df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://localhost:3306/seu_banco_de_dados") \ # Substitua pelo URL do seu
banco de dados
    .option("dbtable", "tabela_de_produtos") \  # Substitua pelo nome da tabela de produtos
    .option("user", "seu_usuario") \  # Substitua pelo seu nome de usuário do banco de dados
    .option("password", "sua_senha") \  # Substitua pela sua senha do banco de dados
    .load()

# Mostrar os dados da tabela de produtos
```

Figura 2: Exemplo de configuração de Seção Spark.

Fonte: Exemplo gerado pelo ChatGPT.

Gerenciador de *Cluster*: Responsável por gerenciar e alocar recursos para os nós do *cluster*, no qual a aplicação *Spark* roda.

Executores: Trabalham em cima de nós trabalhadores e se comunicam com o *driver* do programa. Responsáveis por executar tarefas.

3.1. O PROBLEMA DO *BIG DATA*

Na história da computação, sempre foi visível os grandes avanços na potência dos computadores ao longo dos anos. A cada ano, os computadores ficavam mais e mais potentes, permitindo aplicações mais complexas que faziam uso de *hardware* cada vez menor.

Por um tempo, essa ideia se manteve forte e, empresas cada vez mais tiravam proveito desse padrão, que teve uma diminuição do seu ritmo no ano de 2005 graças a problemas com o tamanho dos componentes físicos. Isso levou empresas a buscarem outras formas de computação, levando a computação paralela e distribuída, o que permitiu

que computadores novamente tivessem um salto no seu processamento e capacidade de computação, não pelo tamanho de seus componentes internos, mas sim pelo trabalho em conjunto de várias máquinas.

Isso, porém, gerou outro desafio: o paralelismo. Grandes sistemas precisavam agora trabalhar em conjunto para que suas funções fossem executadas. Eram necessários novos modelos computacionais.

Além de tudo isso, o preço de armazenamento não parou de cair com a parada no ano de 2005, mas continuou a diminuir, permitindo que empresas pudessem guardar *terabytes* de dados a um preço reduzido.

Tudo isso levou à criação de novas ferramentas para a manipulação de dados e o processamento paralelos, como primeiramente o *Hadoop*, e mais tarde, o *Spark*.

A história do *Apache Spark* começou em 2009, sendo um projeto de pesquisa no AMPLab da UC Berkley, uma colaboração entre estudantes, pesquisadores e professores, que era focada no domínio de aplicações com uso intensivo de dados. O objetivo do Spark era criar uma estrutura, otimizada para processamento iterativo rápido, como *machine learning* e análise iterativa de dados, mantendo a escalabilidade e tolerando falhas do *Hadoop MapReduce*.

O *Hadoop MapReduce* é um modelo de programação para processar conjuntos de *Big Data* com um algoritmo distribuído paralelo, permitindo que os desenvolvedores escrevam operadores massivos paralelizados, sem se preocupar com a distribuição do trabalho e a tolerância de falhas. Porém, como citado brevemente acima, o *MapReduce* continha um desafio.

Em cada etapa, o *MapReduce* lê os dados do *Cluster*, executa operações e grava os resultados no disco. Devido a latência da E/S do disco, as operações são mais lentas.

O *Spark* foi criado para resolver essas limitações do *MapReduce*, realizando processamento na memória, diminuindo o número de etapas em uma tarefa e utilizando dados em várias operações que são paralelas.

O *Spark* permitiu que fosse necessária apenas uma etapa em que os dados são lidos na memória, as operações são executadas e os resultados são gravados de volta, tornando a execução muito mais rápida. O *Spark* também reutiliza dados usando um cache na memória para acelerar de forma considerável os algoritmos de *machine learning* que

chamam repetidamente uma função no mesmo conjunto de dados. Isso é possível pela criação de *DataFrames*, uma abstração sobre o Conjunto de dados resiliente distribuído (RDD), que é, basicamente, uma coleção de objetos armazenados em cache na memória e reutilizados em várias operações do *Spark*, reduzindo em muito a latência, fazendo com que o *Spark* seja muitas vezes mais rápido que o *MapReduce*, principalmente ao realizar machine learning e análises interativas.

O primeiro artigo foi lançado em junho de 2010, com o título "*Spark: Cluster Computing with Working Sets*" (Spark: Computação em CLuster com Conjuntos de Trabalho"). O código era aberto, com uma licença BSD.

Em 2013, o projeto obteve o status de incubação na *Apache Software Foundation* (ASF) e se estabeleceu como um projeto de alto nível da *Apache* em fevereiro de 2014. O *Spark* pode ser executado de forma independente, no *Apache Mesos*, ou com mais frequência, no *Apache Hadoop*.

3.2. *APACHE SPARK* VS *APACHE HADOOP*

Apesar das diferenças no *design* do *Spark* e do *Hadoop MapReduce*, foi descoberto que essas estruturas de *big data* podem se complementar.

O *Hadoop* é uma estrutura de código aberto que tem o Sistema de Arquivos Distribuído do *Hadoop* (HDFS) como armazenamento, o *YARN* como gerenciador de recursos de computação e a implementação do modelo de programação *MapReduce* como forma de execução. Em uma implementação padrão do *Hadoop*, diferentes mecanismos de execução também são usados, como *Spark*, *Tez* (Projeto para a criação de grafos direcionados e acíclicos) e *Presto* (Uma *engine* para *query*, ou busca, SQL).

Já o *Spark* é uma estrutura de código aberto focada em consultas interativas, *machine learning* e *workloads* (Cargas de Trabalho) em tempo real. Não tem um sistema de armazenamento próprio, mas executa análises em outros sistemas de armazenamento, como o *HDFS*, *Cassandra*, entre outros. O *Spark* no *Hadoop* aproveita o *YARN* para compartilhar um *cluster* e um conjunto de dados comuns como outros mecanismos do *Hadoop*.

3.3. BENEFÍCIOS DO *APACHE SPARK*

Rápido: Usando armazenamento em *cache* na memória e uma execução otimizada de consultas, o *Spark* oferece consultas analíticas rápidas em dados de qualquer tamanho.

Para Desenvolvedores: O *Apache Spark* suporta de modo nativo *Java*, *Scala*, *R* e *Python*, oferecendo várias linguagens para a criação de aplicativos. Essas *APIs* facilitam para os desenvolvedores, escondendo a complexidades do processamento distribuído atrás de operadores mais simples e de mais alto nível que reduzem de forma muito drástica a quantidade de código que é preciso.

Vários *workloads*: O *Apache Spark* tem a capacidade de executar várias *workloads*, como consultas interativas, análises em tempo real, *machine learning* e processamento de grafos. Uma mesma aplicação pode combinar vários desses usos facilmente.

3.4. ESTRUTURA DO *APACHE SPARK*

A estrutura do *Spark* consiste em:

- *Spark Core* como base para a plataforma.
- *Spark SQL* para consultas interativas.
- *Spark Streaming* para análises em tempo real.
- *Spark MLib* para machine learning.
- *Spark GraphX* para processamento de Gráficos.

***Spark Core*:** A base da plataforma. É responsável por gerenciar a memória, recuperação de falhas, programação, distribuição e monitoramento de tarefas, além da interação com sistemas de armazenamento. O *Spark Core* é exposto usando uma interface de programação de aplicações *(APIs)*, criada para *Java*, *Scala*, *Python* e *R*. Essas *APIs* escondem a complexidade do processamento distribuído.

***MLib (Machine Learning)*:** Uma biblioteca de algoritmos para realizar *machine learning* em dados de grande escala. Os modelos podem ser treinados por cientistas de dados com *R* ou *Python* em qualquer fonte de dados do *Hadoop*, salvos usando o *MLib* e importados para um *pipeline* baseado em *Java* ou *Scala*. O Spark foi projetado para ter uma computação rápida e interativa executada na memória, permitindo que o *machine learning*

seja executado rapidamente. Os algoritmos são capazes de fazer classificação, regressão, agrupamento, filtragem colaborativa e mineração de padrões.

***Spark Streaming (Tempo Real)*:** O *Spark Streaming* aproveita a capacidade de agendamento rápido do *Spark Core* para fazer análises de *streaming*. A solução consome os dados em pequenos lotes, ou conjuntos, permitindo a análise desses dados com o mesmo código de aplicação escrito para análises em lotes, melhorando a produtividade do desenvolvedor, já que ele pode usar o mesmo código para processamento em lotes e para aplicações de *streaming* em tempo real. A tecnologia é compatível com dados do *Twitter*, *Kafka*, *Flume*, *HDFS* e *ZeroMQ*, além de outros contidos no ecossistema de pacotes do *Spark*.

***Spark SQL* (Consultas Interativas):** O *Spark SQL* é um mecanismo de consulta distribuído, fornecendo consultas interativas de baixa latência que são até 100 vezes mais rápidas que o *MapReduce*. Ele inclui um otimizador que é baseado em custos, armazenamento colunar e geração de códigos para consultas rápidas, e, escaláveis. É possível usar *APIs*, disponíveis no *Scala*, *Java*, *Python* e *R*. Fornece suporte para várias fontes de dados prontos para uso, como JDBC, ODBC, JSON, HDFS, Hive ORC e Parquet.

***GraphX* (Processamento de Grafos):** O *Spark GraphX* permite o processamento de grafos pelo uso de uma estrutura distribuída sobre o *Spark*. O *GraphX* fornece ETL, análise exploratória e computação gráfica iterativa para permitir que usuários criem e transformem de forma interativa uma estrutura de dados gráficos em grande escala. A tecnologia já vem com uma *API* flexível e algoritmos gráficos distribuídos pré-selecionados.

3.5. CASOS DE USO DO *APACHE SPARK*

Sendo um sistema de processamento distribuído de uso geral para *big data*, o *Apache Spark* foi implantado em todos os tipos de casos de uso de *big data* para detectar padrões e fornecer informações em tempo real, como por exemplo:

Serviços Financeiros: O *Spark* é usado no setor financeiro para prever a rotatividade de clientes e recomendar novos produtos financeiros. Também pode ser usado para analisar os preços das ações para prever tendências futuras em bancos de investimentos.

Saúde: O *Spark* é usado para criar um atendimento abrangente ao paciente, disponibilizando dados a profissionais de saúde da linha de frente para interações com o paciente. Também pode ser usado para prever ou recomendar um determinado tratamento para o paciente.

Manufatura: O *Spark* é utilizado para eliminar o tempo de inatividade de equipamentos que são conectados com a internet, recomendando como fazer a manutenção preventiva.

Varejo: O *Spark* é utilizado para atrair e manter clientes por meio de serviços e ofertas personalizadas.

4. NUVEM, TERRAFORM E AWS

O conceito de nuvem pode parecer um pouco obscuro, mas, pode ser entendido como um termo para descrever uma rede de servidores conectados ao redor do mundo, que operam de forma unificada como um único ecossistema.

Esses servidores são responsáveis por armazenar e gerenciar dados e fornecer conteúdos e serviços, como por exemplo, mídias sociais, transmissões de vídeos, entre outros. Em vez de acessar arquivos localmente do seu computador, você pode acessar arquivos e serviços remotamente, em praticamente qualquer lugar do mundo, em qualquer hora. (Microsoft Azure, 2024)

Devido a necessidade do desenvolvimento e gerenciamento de sistemas baseados em nuvem, ao acessar um determinado serviço para tal, é possível que a empresa disponibilize uma interface gráfica para um usuário. Porém, realizar esse trabalho manualmente pode não ser eficiente e escalável. Para solucionar esse problema foi criado o *Terraform*.

O *Terraform* é uma ferramenta de “Infraestrutura como Código”, criada pela HashiCorp, que permite que programadores criem, alterem e versionem infraestrutura com segurança. Descreve a nuvem de "estado final" desejado ou nenhuma infraestrutura local para executar uma aplicação. Em seguida, gera um plano para alcançar esse estado final e o executa para prover a infraestrutura. (IBM, O que é Terraform, 2024)

Podemos dividir o *Terraform* de duas formas: *Terraform Core* e *Terraform Plugins*

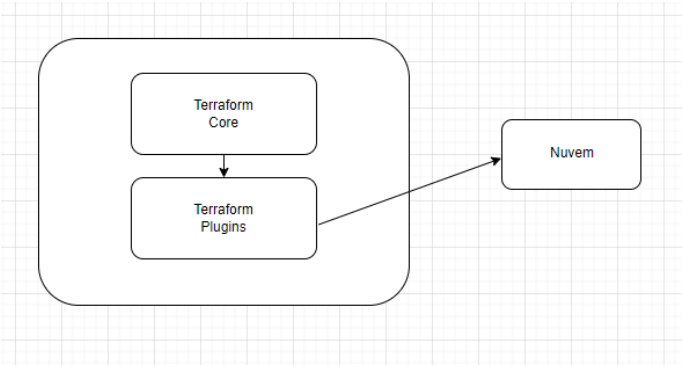

Figura 3: Arquitetura Interna *Terraform*.

Fonte: Imagem criada pelo software draw.io (EdukTI,Terraform: O que é? Por que usar? Como funciona? 2023)

Terraform Core: O *Terraform Core* é responsável pelo gerenciamento do ciclo de vida da estrutura. É o binário, de código aberto, que usamos na linha de comando.

- Primeiro, é levado em consideração o estado atual, e avaliado em relação a configuração desejada.
- Após isso, é proposto um plano para adicionar ou remover componentes da infraestrutura conforme necessário.
- Finalmente, ele cuida do provisionamento ou desativação de quaisquer recursos.

Terraform Plugins: Os plugins do *Terraform* fornecem os meios para que o *Terraform Core* se comunique com seu provedor de infraestrutura ou provedores *SaaS (Software as a Service)*.

Os *Providers* e os Provisionadores do *Terraform* são exemplos de *plug-ins*. Para utilizar os *Plugins*, o *Terraform* utiliza uma chamada de procedimento remoto. (RPC). (EdukTI,Terraform: O que é? Por que usar? Como funciona?, 2023)

Para que o *Terraform* funcione corretamente, três etapas são necessárias:

- **Definição da Infraestrutura como Código:** A infraestrutura é definida pelo usuário usando a linguagem própria do *Terraform* definida pela HCL. Nesse arquivo são definidas as informações de forma declarativa que vão ser criadas pelo *Terraform* na nuvem, como servidores, instâncias de bancos de dados, entre outros. (EdukTI,Terraform: O que é? Por que usar? Como funciona? 2023)
- **Inicialização e Revisão do Plano Terraform:** Após a definição do esquema de referência, é necessário iniciar o *Terraform*. Ao ser iniciado, ele verifica as

dependências e baixa os provedores necessários.

```
> terraform init
 Initializing the backend...
 Initializing modules...

 Initializing providers plugins..
 - Reusing previous version of hashicorp/aws from the dependecy lock file
 - Using previously-installed hashicorp/aws v4.64.0

 Terraform has benn successfully initialized!
```

Figura 4: Comando de inicialização *Terraform*.

Fonte: (EdukTI,Terraform: O que é? Por que usar? Como funciona? 2023)

- Em seguida, é necessário utilizar o comando "***terraform plan***". Será criada uma comparação em relação a estrutura definida no código, e a estrutura existente na nuvem desejada. (EdukTI, Terraform: O que é? Por que usar? Como funciona? 2023)

```
> terraform plan

 ...

 No changes. Your infrastructure matches the configuration.

 Terraform has compared your real infrastructure against your
 configuration and found no differences, so no changes are needed.
```

Figura 5: Comando *terraform plan*.

Fonte: (EdukTI,Terraform: O que é? Por que usar? Como funciona? 2023)

- **Aplicação das alterações:** Após revisar as alterações, o usuário pode aplicar as alterações definidas, remover ou alterar recursos para que o estado desejado seja atingido. Durante esse estágio, o *Terraform* se comunica com a nuvem desejada e efetua as alterações desejadas definidas no código.

(EdukTI,Terraform: O que é? Por que usar? Como funciona? 2023).

```
> terraform apply

...

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Figura 6: Comando *terraform apply*.
Fonte: (EdukTI,Terraform: O que é? Por que usar? Como funciona? 2023)

4.1. AWS

Amazon Web Services, ou AWS, é a plataforma de nuvem mais adotada e mais abrangente do mundo, oferecendo mais de 200 serviços completos de datacenters em todo o mundo. (AWS, O que é AWS?, 2023).

Consiste em um serviço de computação na nuvem desenvolvido pela Amazon. O serviço fornece essas funcionalidades sob demanda através da internet com pagamento conforme o uso da empresa ou do usuário. A intenção é auxiliar na gestão virtual de qualquer aplicação de qualquer aplicação sem custos iniciais ou compromissos fixos, sendo oferecidos serviços em computação, armazenamento e banco de dados, *machine learning*, inteligência artificial, *data lakes*, análises, internet das coisas, entre outros. (Tecnoblog, O que é AWS? [Amazon Web Services], 2023).

4.2. Amazon S3

O Amazon S3, ou *Amazon Simple Storage Service* é um serviço de armazenamento de objetos que oferece escalabilidade e disponibilidade. (AWS, Conceitos básicos do Amazon S3, 2024)

Ele armazena dados como objetos dentro de *buckets*. Um objeto é formado por um arquivo, e, opcionalmente, os metadados que descrevem esse arquivo. Para armazenar um objeto no S3, é necessário carregar os arquivos nos arquivos desejados em um *bucket* e definir as permissões no objeto em quaisquer metadados. (AWS, Conceitos básicos do Amazon S3, 2024)

Buckets são os contêineres para objetos. É possível ter um ou mais *buckets*. Para cada *bucket*, é possível controlar o acesso a ele (quem pode criar, excluir e listar objetos nele), exibir logs de acesso para ele e seus objetos, além disso, escolher a região geográfica onde o S3 armazenará o *bucket* e seu conteúdo. (AWS, Conceitos básicos do Amazon S3)

4.3. Amazon *Glue*

Quando trabalhamos com um projeto de análise, ou *machine learning*, precisamos preparar nossos dados para obter resultados de qualidade. O *AWS Glue* é um serviço de integração de dados com tecnologia sem servidor que torna a preparação de dados mais simples, rápida e barata. É possível descobrir e se conectar com mais de 70 fontes de dados diversos, gerenciar dados em um catálogo de dados centralizados e criar, executar e monitorar visualmente *pipelines ETL* para carregar dados em *data lakes*.

É possível escolher seu mecanismo de integração de dados preferido no AWS Glue para atender aos seus usuários e workloads. (AWS, AWS Glue, 2024)

Figura 7: Estrutura Opções de mecanismo de Integração de dados.

Fonte: (AWS, AWS Glue, 2024)

O AWS *Glue* pode executar seus trabalhos de extração, transformação e carregamento (ETL) à medida que chegam novos dados. Por exemplo, é possível

configurar o AWS *Glue* para iniciar trabalhos de *ETL* a serem executados assim que novos dados estão disponíveis no Amazon S3. (AWS, AWS Glue, 2024)

Figura 8: ETL orientado por eventos.

Fonte: (AWS, AWS Glue, 2024)

Você pode usar o catálogo de dados para descobrir e pesquisar rapidamente diversos conjuntos de dados da AWS sem mover os dados. Quando são catalogados, os dados são disponibilizados imediatamente para pesquisa e consulta, sendo possível usar o *Amazon Athena*, *Amazon EMR* ou o *Amazon Redshift Spectrum.* (AWS, AWS Glue, 2024)

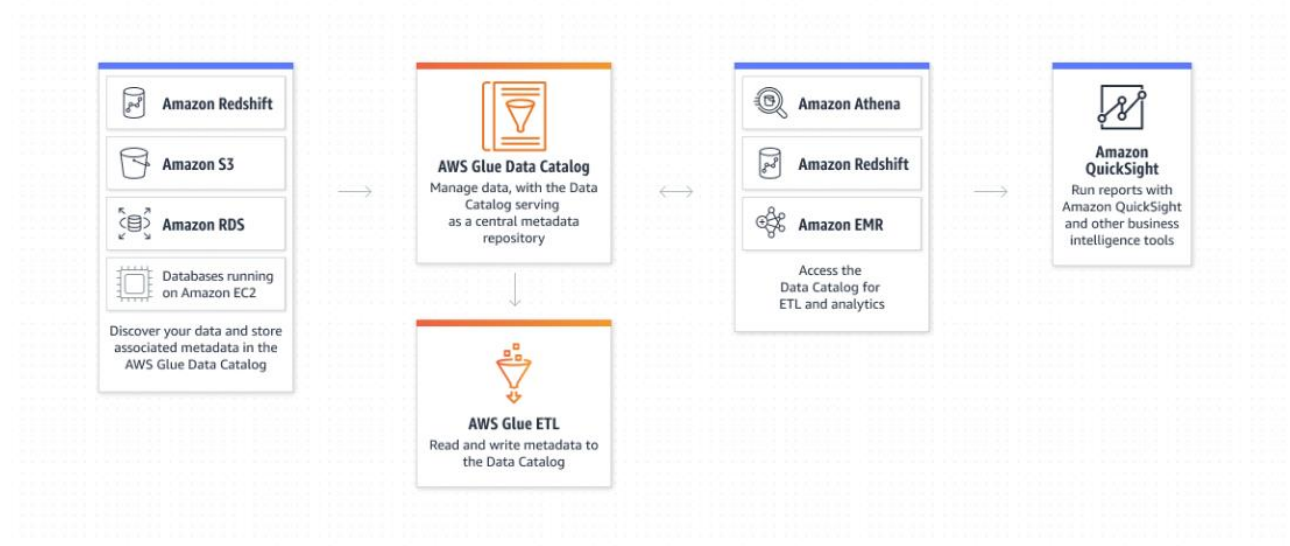

Figura 9: Catálogo de Dados AWS Glue.

Fonte: (AWS, AWS Glue, 2024)

O AWS Glue Studio torna mais fácil criar, executar e monitorar visualmente os trabalhos *ETL* do AWS Glue. Você pode criar trabalhos ETL que migram e transformam os dados usando um editor do tipo arrastar e soltar, e o AWS *Glue* gera automaticamente o código. (AWS, AWS Glue, 2024)

Figura 10: Trabalhos de *ETL* sem código.

Fonte: (AWS, AWS Glue, 2024)

A Qualidade dos dados do AWS Glue automatiza a criação, o gerenciamento e o monitoramento de regras de qualidade de dados para ajudar a garantir dados de alta qualidade nos *data lakes* e *pipelines*. (AWS, AWS Glue, 2024)

Figura 11: Gerencie e Monitore a Qualidade de Dados.

Fonte: (AWS, AWS Glue, 2024)

Com o AWS *Glue DataBrew*, é possível explorar e fazer experimentos dos dados diretamente de um *data lake*, *data warehouses* e bancos de dados, incluindo o *Amazon S3*, o *Amazon Redshift*, o *AWS Lake Formation*, o *Amazon Aurora* e o *Amazon Relational Database Service (RDS)*. É possível escolher entre mais de 250 transformações predefinidas no *DataBrew* para automatizar tarefas de preparação de dados, como filtragem de anomalias, padronização de formatos e correções de valores inválidos.

O AWS *Glue* também oferece uma ferramenta de preparação de dados visual que permite preparar dados usando uma interface visual estilo apontar e clicar, evitando a criação de código.

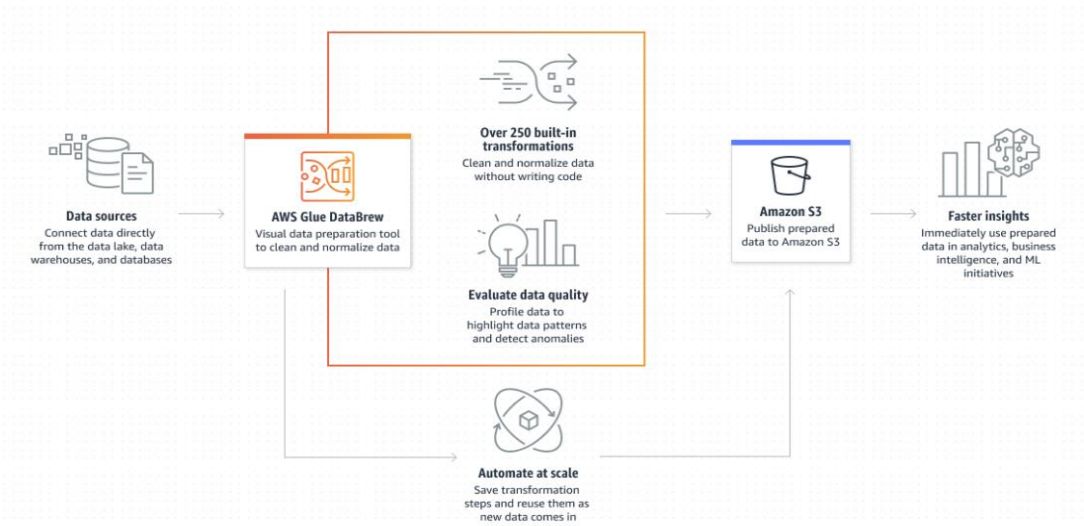

Figura 12: Fluxo de Preparação de Dados.

Fonte: (AWS, AWS Glue, 2024)

4.4. AWS Glue Crawler

É possível usar “*Crawlers*” para preencher o AWS *Glue Catalog* com bancos de dados e tabelas. Um *Crawler* pode ler múltiplas fontes de dados de uma única vez. Ao finalizar a leitura, o *Crawler* cria ou atualiza uma ou mais tabelas no Data Catalog, extrai, transforma e carrega (*ETL*) que são definidos no AWS Glue para serem usados nas tabelas como fontes e alvos. O trabalho *ETL* lê de uma fonte, e escreve em outra que é especificada nas tabelas fonte e alvo do *Data Catalog*. (AWS, Usar crawlers para preencher o catálogo de dados, 2024).

O fluxo de trabalho do AWS *Glue Crawler* funciona da seguinte forma:

1. Um *crawler* executa todos os classificadores personalizados que são escolhidos para inferir o formato e o esquema dos dados. O código é fornecido para classificadores personalizados, e eles são executados na ordem especificada. O primeiro classificador personalizado a reconhecer com sucesso é usado para criar um esquema. Os classificadores personalizados nas listas inferiores são ignorados. (AWS, Usar crawlers para preencher o catálogo de dados, 2024)
2. Se nenhum classificador personalizado corresponder ao esquema dos dados, os classificadores integrados tentarão reconhecê-lo. Um exemplo de classificador embutido é aquele que reconhece *JSON*. (AWS, Usar crawlers para preencher o catálogo de dados, 2024)
3. O *crawler* se conecta ao armazenamento de dados. Alguns armazenamentos de dados requerem propriedades de conexão para o acesso ao *crawler*. (AWS, Usar crawlers para preencher o catálogo de dados, 2024)
4. O esquema referido é criado para os seus dados. (AWS, Usar crawlers para preencher o catálogo de dados, 2024)
5. O *crawler* grava os metadados no *Data Catalog*. Uma definição de tabela contém metadados sobre os dados no seu armazenamento de dados. A tabela é gravada em um banco de dados, que é um contêiner de tabelas no *Data Catalog*. Os atributos de uma tabela incluem a classificação, que é um rótulo criado pelo classificador que inferiu o esquema da tabela. (AWS, Usar crawlers para preencher o catálogo de dados, 2024)

Quando um *crawler* é executado, ele obtém as ações a seguir para consultar um armazenamento de dados:

- **Classifica dados para determinar o formato, o esquema e as propriedades associadas de dados brutos:** Você pode configurar os resultados de classificação criando um classificador personalizado.
- **Agrupa dados em tabelas ou partições:** Os dados são agrupados com base na heurística do *crawler*.
- **Grava metadados no *Data Catalog*:** É possível configurar como *crawler* adiciona, atualiza e exclui tabelas e partições.

Ao definir um *crawler*, é possível escolher um ou mais classificadores que avaliam o formato dos seus dados para inferir um esquema, quando o *crawler* é executado, o primeiro

classificador da lista que reconhece com sucesso seu armazenamento de dados é usado para criar um esquema para a tabela. É possível usar classificadores integrados ou definir seu próprio.

As tabelas de metadados que um *crawler* cria ficam contidas em um banco de dados quando o *crawler* é definido. Se um *crawler* não especificar um banco de dados, suas tabelas serão colocadas em um banco de dados padrão.

O *crawler* gera os nomes das tabelas criadas. Os nomes das tabelas armazenadas no AWS *Glue Data Catalog* seguem estas regras:

- São permitidos somente caracteres alfanuméricos e sublinhados (_).
- Prefixos personalizados não podem conter mais do que 64 caracteres.
- O comprimento máximo do nome não pode ser superior a 128 caracteres. O *crawler* trunca nomes gerados para ajustá-los de acordo com o limite.
- Se forem encontrados nomes de tabelas duplicadas, o *crawler* adiciona um sufixo de *string hash* a esse nome.

Se seu *crawler* for executado mais de uma vez (talvez em um algoritmo), ele irá procurar arquivos ou tabelas novos ou alterados no armazenamento de dados. A saída do *crawler* inclui novas tabelas e partições encontradas desde a execução anterior.

A partir daqui, podemos utilizar ferramentas de consulta de dados como o Amazon *Athena*, que é um serviço de análise interativo e sem servidor, criado em frameworks de código aberto, com suporte a formatos de tabela e arquivos abertos. O Athena fornece uma maneira simplificada e flexível de analisar *petabytes* de dados onde eles residem. Devido ao tempo, não iremos nos aprofundar nesse tópico. (AWS, Amazon Athena, 2024)

5. Arquitetura de Dados

Uma das mais comuns definições de arquitetura de software fala que arquitetura se preocupa com “projeto em mais alto nível”, ou seja, o foco deixa de ser a organização e interfaces de classes individuais, e passa a ser em unidades de maior tamanho, sendo elas pacotes, componentes, módulos, subsistemas, camadas ou serviços. (VALENTE, Engenharia de Software Moderna, 2022)

Além de possuírem um “maior tamanho”, os componentes arquiteturais devem ser relevantes para que um sistema atenda a seus objetivos. (VALENTE, Engenharia de Software Moderna, 2022)

Existe ainda uma segunda definição para arquitetura de software: Arquitetura de software inclui as decisões de projeto mais importantes de um sistema. Essas decisões são tão importantes que, uma vez tomadas, dificilmente poderão ser revertidas no futuro. Essa segunda definição de arquitetura é mais ampla que a primeira, já que considera arquitetura não apenas como um conjunto de módulos, mas como um conjunto de decisões tomadas. (VALENTE, Engenharia de Software Moderna, 2022).

Neste trabalho, serão abordados dois padrões de arquitetura específicos para sistemas de dados, a Arquitetura *Pipes and Filters*, e a Arquitetura Medalhão.

5.1. Arquitetura *Pipes and Filters*

É um tipo de arquitetura orientada a dados, na qual os programas, chamados filtros (*Filters*), tem como função processar os dados recebidos na entrada e gerar uma nova saída. Os filtros são conectados por meio de canos (*Pipes*), que agem como *buffers*. Isto é, canos são usados para armazenar a saída de um filtro, enquanto ela não é lida pelo próximo filtro da sequência. Com isso, os filtros não precisam conhecer seus antecessores e sucessores, o que torna esse tipo de arquitetura bastante flexível, permitindo as mais variadas combinações de programas. Além disso, por construção, filtros podem ser executados em paralelo. (VALENTE, Engenharia de Software Moderna, 2022).

Um exemplo clássico de arquitetura baseada em *pipes and filters* são os comandos de sistema Unix. Por exemplo, a linha de comando **ls | grep csv | sort** que especifica a execução de três comandos *pipes* (filtros) que são conectados por dois *pipes* (barras verticais). No caso dos comandos *Unix*, as entradas são sempre arquivos de texto. (VALENTE, Engenharia de Software Moderna, 2022).

5.2. Arquitetura Medalhão

Quando trabalhamos com um projeto em *big data*, irão aparecer situações em que não iremos ter definições exatas, sendo preciso um pouco mais de paciência e reflexão. Uma dessas questões é como organizar uma estrutura dentro de um *Data Lake*, distribuindo os dados em várias camadas/ níveis diferentes, e ao mesmo tempo, que possam atender necessidades diferentes do negócio.

Camada Transitória: A camada transitória é responsável pela primeira ingestão de dados no *data lake*, servindo como um local provisório para armazenar diferentes variações de arquivos e dados (*RDBMS*, *XML*, *CSV*, *JSON* etc.) provenientes de múltiplas fontes (*ERP*, *WMS*, *E-commerce*, Redes Sociais, *IoT*, entre outras). Esta camada é particularmente útil para organizar os dados conforme suas origens, permitindo que, ao serem ingeridos no *data lake*, já possam ser transformados para formatos de melhor desempenho. (DataSide, Afinal, o que é a “Arquitetura Medalhão?”, 2023)

Nesta fase, os dados são mantidos temporariamente. Após a transformação bem-sucedida para a camada bronze, esses dados podem ser removidos da camada transitória, cumprindo seu papel de transição. (DataSide, Afinal, o que é a “Arquitetura Medalhão?”, 2023)

Camada Bronze: Na camada Bronze, os dados são armazenados em seu formato bruto, exatamente como foram ingeridos da camada transitória. Esta camada serve para garantir que uma cópia fiel dos dados originais esteja disponível para referência. É comum que engenheiros de dados tenham acesso prioritário a essa camada, porém, em alguns casos, cientistas de dados também podem utilizá-la para análises que necessitam de dados não transformados, em busca de insights iniciais. (DataSide, Afinal, o que é a “Arquitetura Medalhão?”, 2023)

Incrementos posteriores podem ser carregados nesta camada utilizando o formato *Delta Table*, garantindo que os dados continuem atualizados de forma incremental. (DataSide, Afinal, o que é a “Arquitetura Medalhão?”, 2023) Por motivos de tempo, não será abordado este formato em detalhes neste trabalho.

Camada Prata / Silver: A camada Prata é responsável por transformar os dados brutos da Bronze em entidades centralizadas e mais refinadas. Aqui, processos de desnormalização, enriquecimento de dados, padronização e controle de qualidade (*Data Quality*) são aplicados. Campos são renomeados de acordo com padrões definidos, como a substituição de abreviações ou a tradução de termos de acordo com a necessidade de negócio. (DataSide, Afinal, o que é a “Arquitetura Medalhão?”, 2023)

Nesta camada, há uma transição da visão técnica (sistemas) para uma visão de negócio, onde dados de diversas tabelas são unificados para criar entidades completas. Por exemplo, tabelas como "Produtos", "Produtos_Categorias" e "Produtos_Fornecedores" podem ser unidas em uma única entidade "Produto", oferecendo uma visão mais organizada e coesa das informações. (DataSide, Afinal, o que é a “Arquitetura Medalhão?”, 2023)

Camada Ouro / Gold: A camada Ouro é a etapa final de preparação dos dados para consumo pelas áreas de negócio. Nessa camada, são aplicadas regras e transformações adicionais, criando tabelas com modelagem, agregações, cálculos e indicadores necessários para atender objetivos específicos. (DataSide, Afinal, o que é a “Arquitetura Medalhão?”, 2023)

6. Desenvolvimento da Aplicação

Conexão com a Nuvem: Primeiro é necessário definir a conexão com a plataforma de nuvem, nesse caso, AWS. É necessário referenciar a plataforma nos provedores requeridos no código *Terraform* para que as funcionalidades da AWS possam ser acessadas. (FreeCodeCamp. Terraform Course - Automate your AWS cloud infrastructure, 2020) (Terraform Registry, AWS Provider, 2024)

Após isso, é configurado a conexão com o provedor, passando uma chave de acesso, e após isso, uma chave secreta. (Terraform Registry, AWS Provider, 2024)

```
# Importar as bibliotecas necessárias
from pyspark.sql import SparkSession

# Criar uma sessão Spark
spark = SparkSession.builder \
    .appName("Ler Tabela de Produtos") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

# Ler os dados da tabela de produtos
produtos_df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://localhost:3306/seu_banco_de_dados") \ # Substitua pelo URL do seu
banco de dados
    .option("dbtable", "tabela_de_produtos") \  # Substitua pelo nome da tabela de produtos
    .option("user", "seu_usuario") \  # Substitua pelo seu nome de usuário do banco de dados
    .option("password", "sua_senha") \  # Substitua pela sua senha do banco de dados
    .load()

# Mostrar os dados da tabela de produtos
```

Figura 13: Configuração em código com Nuvem AWS usando *Terraform*.

Fonte: Exemplo gerado pelo ChatGPT.

Configuração do AWS Glue: Abaixo é configurado a funcionalidade do AWS *Glue*, além de um banco de dados para ser usado. (Terraform Registry, AWS Glue Job, 2024) (Terraform Registry, AWS IAM Role, 2024) (Terraform Registry, AWS Glue Catalog Database, 2024)

```
resource "aws_iam_role" "glue_role" {
  name = "glue-datalake-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Principal = {
          Service = "glue.amazonaws.com"
        },
        Action = "sts:AssumeRole"
      }
    ]
  })
}

# Glue Database
resource "aws_glue_catalog_database" "datalake_db" {
  name = "datalake_db"
}
```

Figura 14: Configuração do AWS *Glue Role*.

Fonte: Exemplo gerado pelo ChatGPT.

Configuração das Camadas Medalhão: Abaixo são definidas as três camadas do *Data Lake*, a camada Bronze, *Silver* (Prata) e *Gold* (Ouro), além de um *bucket* que armazena os *jobs* em *Python* que irão realizar as transformações. (Terraform Registry, AWS AWS S3 Buckets, 2024)

```
# Buckets S3 para a arquitetura em camadas (medalhão)
resource "aws_s3_bucket" "bronze_bucket" {
  bucket = "datalake-bronze-layer"
}

resource "aws_s3_bucket" "silver_bucket" {
  bucket = "datalake-silver-layer"
}

resource "aws_s3_bucket" "gold_bucket" {
  bucket = "datalake-gold-layer"
}

resource "aws_s3_bucket" "glue_jobs_bucket" {
  bucket = "meu-datalake-glue-scripts"
}
```

Figura 15: Configuração das camadas bronze, prata e ouro.

Fonte: Exemplo gerado pelo ChatGPT.

Configuração do *Glue Crawler*: São configurados os *crawlers* que vão analisar cada camada e criar as tabelas necessárias. Abaixo também, são definidos os *jobs* que vão ser aplicados em cada transformação. (Terraform Registry, AWS Glue Crawler, 2024)

```
resource "aws_glue_crawler" "bronze_crawler" {
  database_name = aws_glue_catalog_database.datalake_db.name
  name          = "bronze-layer-crawler"
  role          = aws_iam_role.glue_role.arn

  s3_target {
    path = "s3://${aws_s3_bucket.bronze_bucket.bucket}/"
  }
}

resource "aws_glue_crawler" "silver_crawler" {
  database_name = aws_glue_catalog_database.datalake_db.name
  name          = "silver-layer-crawler"
  role          = aws_iam_role.glue_role.arn

  s3_target {
    path = "s3://${aws_s3_bucket.silver_bucket.bucket}/"
  }
}

resource "aws_glue_crawler" "gold_crawler" {
  database_name = aws_glue_catalog_database.datalake_db.name
  name          = "gold-layer-crawler"
  role          = aws_iam_role.glue_role.arn

  s3_target {
    path = "s3://${aws_s3_bucket.gold_bucket.bucket}/"
  }
}
```

Figura 16: Configuração do *Glue Crawler* para cada camada.

Fonte: Exemplo gerado pelo ChatGPT.

Configuração dos *Jobs*: Os *jobs* definidos no Terraform referenciam os Jobs simbólicos criados, utilizando Spark e AWS *Glue*. Como foram criados localmente, eles precisam ser enviados para o bucket que irá guardar todos os *jobs* que serão executados entre as camadas. Os comandos abaixo são usados para isso. (Terraform Registry, AWS Glue Job, 2024)

```
# Glue Job para transformar dados da camada Bronze para Silver
resource "aws_glue_job" "bronze_to_silver_job" {
  name     = "bronze-to-silver-job"
  role_arn = aws_iam_role.glue_role.arn

  command {
    name            = "glueetl"
    script_location = "s3://${aws_s3_bucket.glue_jobs_bucket.bucket}/glue-scripts/bronze_to_silver.py"
    python_version  = "3"
  }
}

# Glue Job para transformar dados da camada Silver para Gold
resource "aws_glue_job" "silver_to_gold_job" {
  name     = "silver-to-gold-job"
  role_arn = aws_iam_role.glue_role.arn

  command {
    name            = "glueetl"
    script_location = "s3://${aws_s3_bucket.glue_jobs_bucket.bucket}/glue-scripts/silver_to_gold.py"
    python_version  = "3"
  }
}
```

Figura 17: Configuração Job para serem aplicados.

Fonte: Exemplo gerado pelo ChatGPT.

```
import sys
from pyspark.context import SparkContext
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.dynamicframe import DynamicFrame
from awsglue.context import GlueContext

# Inicializar o GlueContext
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Parâmetros do Job
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
glueContext.log.info(f"Starting job: {args['JOB_NAME']}")

# Carregar dados da camada Bronze (S3)
bronze_path = "s3://datalake-bronze-layer/"
bronze_data = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": [bronze_path]},
    format="parquet"  # Ou outro formato suportado como "csv", "json", etc.
)

# Limpeza básica: remover duplicatas e nulos
cleaned_data = bronze_data.drop_nulls().drop_duplicates()

# Salvar na camada Silver
silver_path = "s3://datalake-silver-layer/"
glueContext.write_dynamic_frame.from_options(
    frame=cleaned_data,
    connection_type="s3",
    connection_options={"path": silver_path},
    format="parquet"
)

glueContext.log.info("Job completed: Data moved from Bronze to Silver")
```

Figura 18: *Job* bronze para prata. Exemplo para ser aplicado nas bases de dados.

Fonte: Exemplo gerado pelo ChatGPT.

```
import sys
from pyspark.context import SparkContext
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from pyspark.sql.functions import col

# Inicializar o GlueContext
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Parâmetros do Job
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
glueContext.log.info(f"Starting job: {args['JOB_NAME']}")

# Carregar dados da camada Silver (S3)
silver_path = "s3://datalake-silver-layer/"
silver_data = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": [silver_path]},
    format="parquet"
)

# Converter para DataFrame do Spark para aplicar funções complexas
silver_df = silver_data.toDF()

# Exemplo de transformação: agregação por categoria
aggregated_df = silver_df.groupBy("categoria").agg(
    {"valor": "sum"}  # Exemplo: somar uma coluna chamada "valor"
)

# Converter de volta para DynamicFrame
aggregated_dynamic_frame = DynamicFrame.fromDF(
    aggregated_df, glueContext, "aggregated_dynamic_frame")

# Salvar na camada Gold
gold_path = "s3://datalake-gold-layer/"
glueContext.write_dynamic_frame.from_options(
    frame=aggregated_dynamic_frame,
    connection_type="s3",
    connection_options={"path": gold_path},
    format="parquet"
)

glueContext.log.info("Job completed: Data moved from Silver to Gold")
```

Figura 19: *Job* prata para outubro. Exemplo para ser aplicado nas bases de dados.

Fonte: Exemplo gerado pelo ChatGPT.

```
aws s3 cp ./glue-scripts/bronze_to_silver.py s3://meu-datalake-glue-script/glue-scripts/
aws s3 cp ./glue-scripts/silver_to_gold.py s3://meu-datalake-glue-script/glue-scripts/
```

Figura 20: Comando para o envio dos *Jobs* para a AWS.

Fonte: Exemplo gerado pelo ChatGPT.

Após isso, é necessário usar dois comandos do *Terraform* para que as atualizações sejam feitas na nuvem, O ***terraform plan***, que irá analisar todas as mudanças de código e as comparar com a infraestrutura da nuvem usada, fazendo uma lista de alterações a serem feitas.

```
$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # aws_athena_workgroup.athena_workgroup will be created
  + resource "aws_athena_workgroup" "athena_workgroup" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "datalake-athena-workgroup"
      + state         = "ENABLED"
      + tags_all      = {
          + "Environment" = "DataLake-test"
          + "Project"     = "DataLake-infraestructure"
        }

      + configuration {
          + enforce_workgroup_configuration    = true
          + publish_cloudwatch_metrics_enabled = true
          + requester_pays_enabled             = false

          + result_configuration {
              + output_location = "s3://datalake-bronze-layer/athena-results/"
            }
        }
    }
...
```

Figura 21: Comando “terraform plan”.

Fonte: (Developer Hashicorp, Command Plan, 2024)

E o ***terraform apply***, que realmente irá criar os dados na nuvem.

```
$ terraform apply
aws_iam_role.glue_role: Refreshing state... [id=glue-datalake-role]
aws_glue_catalog_database.datalake_db: Refreshing state... [id=654654318164:datalake_db]
aws_s3_bucket.bronze_bucket: Refreshing state... [id=datalake-bronze-layer]
aws_s3_bucket.silver_bucket: Refreshing state... [id=datalake-silver-layer]
aws_s3_bucket.gold_bucket: Refreshing state... [id=datalake-gold-layer]
aws_s3_bucket.glue_jobs_bucket: Refreshing state... [id=meu-datalake-glue-scripts]
aws_iam_policy_attachment.glue_s3_policy: Refreshing state... [id=glue-s3-policy-attachment]
aws_glue_crawler.bronze_crawler: Refreshing state... [id=bronze-layer-crawler]
aws_glue_job.silver_to_gold_job: Refreshing state... [id=silver-to-gold-job]
aws_glue_job.bronze_to_silver_job: Refreshing state... [id=bronze-to-silver-job]
aws_glue_crawler.silver_crawler: Refreshing state... [id=silver-layer-crawler]
aws_glue_crawler.gold_crawler: Refreshing state... [id=gold-layer-crawler]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Figura 22: Comando “***terraform apply***”.

Fonte: (Developer Hashicorp, Command Apply, 2024)

Ao final, é possível ver as mudanças feitas. No caso acima, nenhuma mudança foi registrada devido a um ***apply*** anterior.

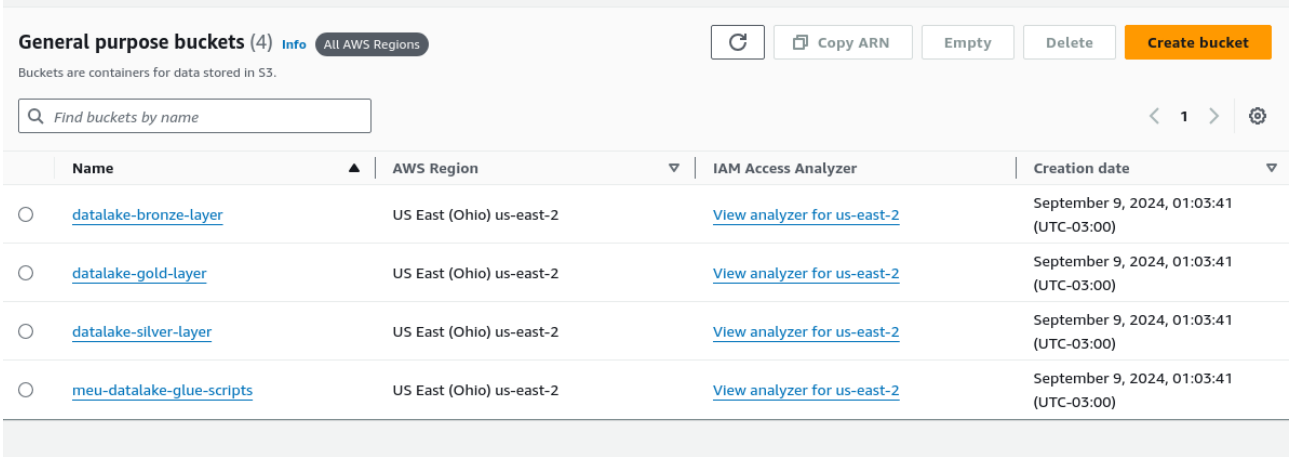

Figura 23: Tabela de Buckets.

Fonte: (Amazon AWS Console, Tabela de Buckets, 2024)

Finalmente, é possível observar as camadas criadas pelo *Terraform*. A partir daqui, pode-se dizer que foi criada uma estrutura prática de um *Data Lake*.

7. Conclusão

O trabalho apresentado demonstra o passo a passo para a definição e criação da estrutura básica para um *Data Lake*. Os passos aqui não têm o intuito de criar uma aplicação totalmente funcional, mas sim, definir as bases necessárias e a fundação para a construção dela, definindo arquiteturas, como a Arquitetura Medalhão, ferramentas e conceitos essenciais que permitem um desenvolvimento escalável, como as ferramentas *AWS*, *Terraform* e outros. A partir dos passos apresentados, é possível partir para novas direções, melhorando o projeto conforme a necessidade de cada negócio.

8. Referências

Microsoft. O que é um Data Lake. Disponível em: O que é um data lake? Data lake versus data warehouse | Microsoft Azure. Acesso em 17 de Outubro de 2024.

Amazon AWS Console. Tabela de Buckets. Disponível em: Buckets do S3 | S3 | sa-east-1 (amazon.com). Acesso em 09 de Setembro de 2024.

Developer Hashicorp. Command Apply. Disponível em: Command: apply | Terraform | HashiCorp Developer. Acesso em 09 de Setembro de 2024.

Developer Hashicorp. Command Plan. Disponível em: Command: plan | Terraform | HashiCorp Developer. Acesso em 09 de Setembro de 2024.

Terraform Registry. AWS Glue Catalog Database. Disponível em: aws_glue_catalog_database | Resources | hashicorp/aws | Terraform | Terraform Registry. Acesso em 09 de Setembro de 2024.

Terraform Registry. AWS IAM Role. Disponível em: aws_iam_role | Resources | hashicorp/aws | Terraform | Terraform Registry. Acesso em 09 de Setembro de 2024.

Terraform Registry. AWS Glue Crawler. Disponível em: aws_glue_crawler | Resources | hashicorp/aws | Terraform | Terraform Registry. Acesso em 09 de Setembro de 2024.

Terraform Registry. AWS S3 Bucket. Disponível em: aws_s3_bucket | Resources | hashicorp/aws | Terraform | Terraform Registry. Acesso em 09 de Setembro de 2024.

Terraform Registry. AWS Glue Job. Disponível em: aws_glue_job | Resources | hashicorp/aws | Terraform | Terraform Registry. Acesso em 09 de Setembro de 2024.

Terraform Registry. AWS Provider. Disponível em: Docs overview | hashicorp/aws | Terraform | Terraform Registry. Acesso em 09 de Setembro de 2024.

FreeCodeCamp. Terraform Course - Automate your AWS cloud infrastructure. Disponível em: https://www.youtube.com/watch?v=SLB_c_ayRMo&t=4998s. Acesso em 09 de Setembro de 2024.

DataSide. Afinal, o que é a Arquitetura Medalhão?. Disponível em: Afinal, o que é a “Arquitetura Medalhão?” | Dataside. Acesso em 08 de Setembro de 2024

VALENTE, Marco Túlio. Engenharia de Software Moderna, Primeira Edição, Minas Gerais: Editora Independente, 2022.

AWS. Amazon Athena. Disponível em: AWS Athena - Serviço de consultas interativas sem servidor (amazon.com). Acesso em 08 de Setembro de 2024

AWS. Usar crawlers para preencher o catálogo de dados. Disponível em:Usar crawlers para preencher o catálogo de dados - AWS Glue (amazon.com). Acesso em 08 de Setembro de 2024.

AWS. AWS Glue. Disponível em: https://aws.amazon.com/pt/glue/. Acesso em 07 de Setembro de 2024.

AWS. Conceitos básicos do Amazon S3. Disponível em: https://aws.amazon.com/pt/s3/getting-started/. Acesso em 03 de Setembro de 2024.

Tecnoblog. O que é AWS? [Amazon Web Services]. Disponível em: https://tecnoblog.net/responde/o-que-e-a-aws-amazon-web-services/. Acesso em 03 de Setembro de 2024.

AWS. O que é AWS? Como funciona o Amazon Web Services? Disponível em: https://aws.amazon.com/pt/what-is-aws/ . Acesso em 03 de Setembro de 2024.

EdukTI. Terraform: O que é? Por que usar? Como funciona?. Disponível em: https://blog.edukti.com/p/terraform Acesso em 03 de Setembro de 2024.

IBM. O que é Terraform. Disponível em: https://www.ibm.com/br-pt/topics/terraform Acesso em 21 de Agosto de 2024.

Microsoft Azure. O que é nuvem. Disponível em: https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-the-cloud/ Acesso em 18 de Agosto de 2024.

Amazon AWS. O que é Apache Spark. Disponível em: https://aws.amazon.com/pt/what-is/apache-spark/ . Acesso em: 06 de Março de 2024.

Apache Tez. Introduction. Disponível em: https://tez.apache.org/. Acesso em: 06 de Março de 2024.

Apache Presto. What is Presto? Disponível em: https://prestodb.io/ Acesso em 06 de Março de 2024.

IBM. O que é Apache Spark. Disponível em https://www.ibm.com/br-pt/topics/apache-spark#:~:text=IBM&text=IBM%20watsonx.data-

,O%20que%20%C3%A9%20o%20Apache%20Spark%3F,c%C3%B3digo%20aberto%20em%20big%20data. . Acesso em: 23 de Outubro de 2023.

MEDIUM. Spark 101: Introdução ao framework de processamento de dados distribuídos. Disponível em: https://medium.com/gabriel-luz/spark-101-introdu%C3%A7%C3%A3o-ao-framework-de-processamento-de-dados-distribu%C3%ADdos-1f959e596024 . Acesso em: 06 de Novembro de 2023.

MINDMINERS. Análise de dados: conheça as 8 principais ferramentas de Big Data para usar nos negócios. Disponível em: https://mindminers.com/blog/ferramentas-de-big-data/ . Acesso em: 23 de Outubro de 2023.

ORACLE. O que é Big Data? Local. Disponível em: https://www.oracle.com/br/big-data/what-is-big-data/ . Acesso em: 23 de Outubro de 2023.

FRANÇA, Tiago Cruz. Tópicos em Gerenciamento de Dados e Informações 2014, Primeira Edição, Porto Alegre, SP: Sociedade Brasileira de Computação, 2017.

CHAMBERS, Bill; ZAHARIA, Matei. Spark: The Definitive Guide: Big Data Processing Made Simple. 2018, Primeira Edição, O'Reilly Media.

DAMJI, Jules S.; WENIG, Brooke; DAS, Tathagata; LEE, Denny. Learning Spark: Lightning-Fast Data Analytics. 2020, Segunda Edição, O'Reilly Media.