Universidade Católica de Pernambuco
Escola de Tecnologia e Comunicação

C3

José Almir Mariano Figueiredo Junior

Arquitetura e Implementação de um Data Lakehouse na Nuvem

Trabalho de Conclusão de Curso de Graduação

Recife, 2025

José Almir Mariano Figueiredo Junior

Arquitetura e Implementação de um Data Lakehouse na Nuvem

Trabalho apresentado ao Colegiado do Curso de Ciência da Computação da Universidade Católica de Pernambuco como parte dos requisitos para a obtenção do Grau de bacharelado em Ciência da Computação.

Universidade Católica de Pernambuco

Orientador: Prof. Assis Tiago de Oliveira Filho

Recife

2025

José Almir Mariano Figueiredo Junior

Arquitetura e Implementação de um Data Lakehouse na Nuvem/ José Almir Mariano Figueiredo Junior. – Recife, 2025-

p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Assis Tiago de Oliveira Filho

Trabalho de Conclusão de Curso de Graduação – Universidade Católica de Pernambuco, 2025.

1. Engenharia de Dados, Nuvem, Apache Spark, Arquitetura *Medallion*, Plataforma de Dados. 2. Data Engineering, Cloud, Apache Spark, *Medallion* Architecture, Data Platform. I. Assis Tiago de Oliveira Filho. II. Universidade Católica de Pernambuco. III. Escola de Tecnologia e Comunicação. IV. Arquitetura e Implementação de um Data Lakehouse na Nuvem

CDU 02:141:005.7

Dissertação defendida e aprovada em 10 de Dezembro de 2025 pela comissão avaliadora constituída pelos professores e professoras:

Prof. Assis Tiago de Oliveira Filho
Orientador

Prof. Paulo Henrique De Alcantara Rocha.
Convidado

Recife, 2025

Agradecimentos

A realização deste trabalho só foi possível graças ao apoio de pessoas e à fé que me sustentou em tantos momentos na vida.

Agradeço primeiramente a Deus, por me dar saúde, discernimento e por permitir que eu chegasse até aqui mesmo com os desafios.

Aos meus pais, José Almir e Wanderlea Oliveira, agradeço pelo amor incondicional, pela educação que me deram e pelo suporte em todas as fases da minha vida. Sem o apoio e ajuda de vocês, esta caminhada teria sido muito mais difícil.

Ao meu professor orientador, Prof. Assis Tiago de Oliveira Filho, sou grato pela orientação objetiva, paciência, pelas contribuições técnicas e pelas conversas que ampliaram minha visão ao longo do desenvolvimento não só deste trabalho como da minha vida acadêmica.

Aos demais professores que fizeram parte da minha graduação, em especial aos que tive maior contato na jornada acadêmica: Prof. Marcos Canejo, Prof. Lucas Rodolfo, Prof. Diego Pinheiro e Prof. Sergio Murilo. Gratidão por todas as conversas e aprendizados.

Aos colegas e amigos que caminharam comigo durante a graduação, meu sincero obrigado pela parceria, pelas trocas de conhecimento e pelo apoio mútuo em tantos momentos desafiadores na vida acadêmica: Marcelo Coelho, José Carlos, Maria Eduarda Santana, Isadora Xavier, Heloisa Fernanda, Lucca Milano, Rian Delou, Gabriel Torres, João Grimaldi, Humberto Carneiro, Marone Carvalho e Pedro Pepeu.

Por fim, agradeço especialmente a Arthur Luz e Sidney Cirqueira, que foram fundamentais no meu início de carreira. Seu apoio, direcionamento e disposição em compartilhar experiência fizeram diferença nos meus primeiros passos profissionais e contribuíram para que eu seguisse adiante com mais segurança e clareza. Sou profundamente grato por essa presença e incentivo.

"Sem dados, você é apenas mais uma pessoa com uma opinião." (W. Edwards Deming)

Resumo

Este trabalho visa projetar e implementar uma plataforma de dados na nuvem Microsoft Azure, fundamentada na arquitetura *Medallion* e no uso do framework Apache Spark. A solução abrange todas as etapas essenciais de um *pipeline* analítico, desde a geração e ingestão de dados em modo *Batch* até sua organização para consumo analítico. A infraestrutura necessária para o ambiente, incluindo armazenamento, recursos de processamento e orquestração, é provisionada por meio do Terraform, garantindo reprodutibilidade e consistência no ciclo de vida dos recursos.

A plataforma é estruturada segundo o modelo de *Data Lakehouse*, que integra a flexibilidade de armazenamento de um *Data Lake* com as garantias transacionais e a camada semântica típicas de sistemas analíticos consolidados. Esse modelo é viabilizado pelo uso de tabelas Delta Lake e do catálogo nativo do Spark, que juntos fornecem gerenciamento de metadados, controle de versões, eficiência de consulta e padronização para o consumo dos dados.

A metodologia empregada consiste em um estudo de caso orientado à avaliação da escalabilidade e do desempenho do pipeline. Para os experimentos, foram utilizados dados sintéticos de IoT gerados especificamente para este trabalho, permitindo controlar a volumetria nos cenários de 10 GB, 40 GB e 100 GB. Essa abordagem garante consistência entre os testes e possibilita análises comparativas mais precisas. A organização em camadas Bronze, Silver e Gold promove uma progressão gradual de qualidade, desde o armazenamento bruto até views semânticas preparadas para análise.

Entre os resultados esperados está a demonstração de como o *pipeline* se comporta diante de diferentes volumetrias de dados, evidenciando a capacidade de escalabilidade inerente ao processamento distribuído. Espera-se observar como o aumento do volume afeta o tempo de execução das etapas de transformação e consolidação, bem como a forma pela qual o Spark distribui carga entre os nós disponíveis para manter eficiência relativa mesmo em cenários mais extensos. A organização em camadas da arquitetura *Medallion* e o uso do *Delta Lake* também contribuem para maior previsibilidade e confiabilidade na execução, permitindo avaliar como uma arquitetura de *Data Lakehouse* suporta *workloads* crescentes de maneira consistente.

Palavras-chaves: *Data Lakehouse*, Apache Spark, *Delta Lake*, Arquitetura *Medallion*, Terraform.

Abstract

This work aims to design and implement a cloud-based data platform on Microsoft Azure, grounded in the *Medallion* Architecture and the Apache Spark framework. The proposed solution covers all essential stages of an analytical data pipeline, from the *Batch* generation and ingestion of data to its organization for analytical consumption. The underlying infrastructure, including storage, processing resources, and orchestration components, is provisioned through *Terraform*, ensuring reproducibility and consistency throughout the resource lifecycle.

The platform is structured according to the *Data Lake*house model, which combines the flexibility of *Data Lake* storage with the transactional guarantees and semantic organization commonly found in consolidated analytical systems. This model is enabled by Delta Lake tables and the native Spark Catalog, which together offer metadata management, version control, query efficiency, and standardized access to curated datasets.

The methodology is based on a case study focused on evaluating both the scalability and the performance of the pipeline. For the experiments, synthetic IoT data were generated specifically for this work, allowing precise control over the dataset sizes of 10 GB, 40 GB, and 100 GB. This strategy ensures consistency across tests and supports more accurate comparative analysis. The organization into Bronze, Silver, and Gold layers provides a gradual refinement of data quality, from raw storage to semantic views prepared for analysis without physical data duplication.

The expected results include demonstrating how the pipeline behaves under different data volumes, highlighting the inherent scalability of distributed processing. The study aims to observe how increasing volumes affect execution times in transformation and consolidation stages, as well as how Spark distributes *workloads* across available nodes to maintain relative efficiency even in larger scenarios. The *Medallion* Architecture and the use of Delta Lake also contribute to greater predictability and reliability, offering insights into how a *Data Lake*house architecture can consistently support growing analytical *workloads*.

Key-words: *Data Lake*house, Apache Spark, Delta Lake, Arquitetura *Medallion*, *Terraform*.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
ADLS	Azure Data Lake Storage
ADF	Azure Data Factory
BI	Business Intelligence
API	Application Programming Interface
HCL	HarshiCorp Configuration Language
ETL	Extract, Transform and Loading
ELT	Extract, Loading and Transform
Iac	Infrastructure as Code
SQL	Structured Query Language
ORC	Optimized Row Columnar
IoT	Internet of things
CLI	Command Line Interface
GUI	Graphical User Interface
ACID	Atomicity, Consistency, Isolation, Durability

Lista de símbolos

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence
$\sum$	Somatório

Sumário

Parte I

Preparação da pesquisa

1 Introdução

Nos últimos anos, a quantidade de dados gerados e armazenados cresceu substancialmente, sobretudo no mundo corporativo(KHANRA; DHIR; MÄNTYMÄKI, 2020). Esse crescimento exponencial é resultado da digitalização acelerada de processos em diversos setores, do avanço de dispositivos baseados em Internet das Coisas (IoT) e do uso massivo de serviços digitais(FUTUREIOT, 2020)(SHARMA; SHARMA, 2021). Nesse cenário, a geração de dados ocorre em diversos âmbitos e setores, abrangendo grande parte das atividades realizadas por pessoas e corporações.

Empresas que antes utilizavam exclusivamente bancos OLTP (*Online Transaction Processing*) para armazenar dados transacionais passaram a complementar essas estruturas com sistemas voltados a análises (OLAP), reconhecendo o valor estratégico dos dados (DELOITTE, 2021). Nesse contexto, as organizações passaram a adotar soluções analíticas, com o objetivo de obter insights de processos internos e do mercado de atuação. As soluções analíticas trouxeram uma série de casos de uso no ambiente corporativo, como identificar padrões, personalizar estratégias e otimizar processos (MOHAMED, 2016).

Com o amadurecimento das soluções analíticas, as organizações passaram a integrar técnicas mais avançadas de análise, resultando no uso de Inteligência Artificial no contexto empresarial (SNOWFLAKE, 2024). Nesse contexto, os dados tornaram-se ativos ainda mais relevantes, uma vez que passaram a possibilitar entender tendências futuras, assim como realizar automações mais complexas. Consequentemente, organizações que investem em infraestrutura de dados ampliam sua capacidade de inovação e ganham vantagem competitiva em relação àquelas que permanecem com estruturas tradicionais (HARVARD BUSINESS REVIEW, 2023).

Apesar disso, muitos desafios ainda persistem: fragmentação das fontes, diversidade de formatos, ausência de governança, elasticidade limitada de recursos e dificuldade em escalar pipelines robustos. Segundo o relatório Rethink Data, da Seagate em parceria com a IDC (2020), aproximadamente 68% dos dados disponíveis para as empresas não são utilizados (SEAGATE, 2020). Esses obstáculos dificultam a transformação dos dados em informações confiáveis para análise e tomada de decisão.

Nesse cenário, torna-se essencial compreender e implementar arquiteturas modernas de dados capazes de lidar com variados volumes e formatos, mantendo escalabilidade, confiabilidade e padronização. Entre essas abordagens, destaca-se o modelo de *Data Lakehouse* (DATABRICKS, 2025b), que integra a flexibilidade de armazenamento típica dos *Data Lakes* com características transacionais, gerenciamento consistente de metadados e uma camada semântica unificada, tradicionalmente encontrada em ambientes de *Data*

Warehouse. Essa convergência tem se consolidado como uma das principais estratégias para lidar com o crescimento dos dados e com a necessidade de processamento distribuído em larga escala (ARMBRUST; GHODSI et al., 2021).

Com base nessa perspectiva, este trabalho propõe o desenvolvimento de uma plataforma moderna de dados baseada nos serviços da nuvem Microsoft Azure. A solução adota o modelo de *Data Lakehouse*, combinando a flexibilidade de armazenamento do *Data Lake* com mecanismos transacionais e uma camada semântica unificada. Essa abordagem é implementada pela arquitetura *Medallion* e pelo uso de ferramentas como Azure Data Factory, Data Lake Storage, Azure Synapse e Apache Spark para orquestração, armazenamento e processamento distribuído.

Para fins de demonstração prática e avaliação de desempenho, são utilizados dados sintéticos de IoT gerados especificamente para este estudo, permitindo controlar a volumetria e reproduzir cenários realistas de ingestão, transformação e consumo analítico.

Este trabalho foca na ingestão de dados em modo *Batch*, não abordando técnicas de streaming ou processamento em tempo real. O escopo está limitado à construção e demonstração de uma arquitetura moderna de dados, desde a geração e ingestão dos dados sintéticos até sua organização em camadas analíticas. Essa escolha concentra o estudo na camada fundamental da plataforma, que envolve infraestrutura, armazenamento e processamento distribuído, sustentando cenários analíticos e de *machine learning* em ambientes de larga escala.

1.1 Justificativa

Este projeto fundamenta-se na oportunidade de demonstrar a implementação de uma plataforma de dados eficiente e escalável, voltada ao armazenamento, processamento e análise de variados volumes de informações, capaz de sustentar a geração de percepções e apoiar a tomada de decisão.

Apesar da relevância crescente dos dados, muitas organizações ainda enfrentam limitações técnicas e operacionais para estruturar seus pipelines. A fragmentação das fontes, a heterogeneidade dos formatos e a dificuldade em garantir confiabilidade e governança ao longo do processo tornam o uso estratégico dos dados um desafio constante (KARUNAKARAN; AGARWAL, 2025).

Neste trabalho, propõe-se uma solução que integra automação, governança e elasticidade, utilizando o ecossistema da Microsoft Azure e organizando os dados segundo a arquitetura *Medallion*. A proposta visa não somente construir um *pipeline* funcional, mas também aplicar boas práticas que assegurem rastreabilidade, reuso e consistência entre camadas. Para fins de demonstração, adota-se um cenário ilustrativo baseado em dados

sintéticos de IoT, gerados especificamente para este trabalho.

A contribuição do trabalho está tanto na aplicação prática quanto no embasamento conceitual, podendo servir como referência para profissionais e acadêmicos interessados em arquitetura de dados moderna, especialmente no contexto de computação em nuvem.

1.2 Problema e Questão de Pesquisa

Embora as tecnologias de análise e armazenamento tenham evoluído, a maioria das empresas ainda enfrenta barreiras para estruturar pipelines de dados escaláveis, automatizados e confiáveis. Em arquiteturas tradicionais, existe uma grande dificuldade em lidar com a rigidez do ambiente, assim como governança, elasticidade e escalabilidade (AGARWAL, 2025) (KARUNAKARAN; AGARWAL, 2025).

Em sistemas *on-premises*, as empresas precisam provisionar recursos necessários para suportar momentos de pico de uso, resultando em uma ociosidade de recursos computacionais em períodos de menor demanda. Ao adotar uma infraestrutura em nuvem, que utiliza a abordagem de cobrança por utilização de recursos, é possível controlar o custo de recursos computacionais conforme os cenários.

Além disso, persistem os seguintes problemas estruturais (AGARWAL, 2025) (KARUNAKARAN; AGARWAL, 2025):

- Dificuldade na ingestão contínua de dados heterogêneos de múltiplas fontes;
- Falta de padronização e controle de qualidade durante o tratamento dos dados;
- Ausência de rastreabilidade, versionamento e governança das etapas do pipeline;
- Baixa integração entre ferramentas e serviços utilizados, dificultando a automação e manutenção da arquitetura.

Esses obstáculos impactam diretamente a capacidade das organizações de transformar dados em valor, comprometendo a agilidade de times que utilizam dados para a tomada de decisão.

Diante desse contexto, surge a seguinte questão de pesquisa:

Questão de pesquisa: Como a implementação de uma Plataforma Moderna de Dados na nuvem Microsoft Azure, baseada no framework Apache Spark, no modelo de Lakehouse e na arquitetura *Medallion*, pode otimizar pipelines de dados e, ao mesmo tempo, garantir governança, rastreabilidade, elasticidade operacional e eficiência de custos?

1.3 Objetivos

Esta seção visa abordar os objetivos do trabalho que está sendo desenvolvido. Inicialmente, será definido o objetivo geral, que representa a abordagem central da pesquisa e norteia todas as etapas da construção da solução proposta.

Em seguida, serão descritos os objetivos específicos, que detalham as ações práticas e necessárias para alcançar o propósito principal. O detalhamento dos objetivos é fundamental para definir a delimitação do escopo do estudo realizado, que envolve a implementação de uma plataforma de dados moderna na nuvem.

1.3.1 Objetivo Geral

Implementar uma Plataforma Moderna de Dados na nuvem Microsoft Azure, utilizando o framework Apache Spark, o modelo de *Lakehouse* e a arquitetura *Medallion*, com foco em demonstrar escalabilidade, elasticidade, governança e eficiência de custos em pipelines analíticos.

A solução proposta engloba desde a ingestão até a disponibilização dos dados, evidenciando como a automação e a alocação dinâmica de recursos podem eliminar a ociosidade e otimizar o uso da infraestrutura em cenários analíticos modernos.

1.3.2 Objetivos Específicos

1. Ingerir dados sintéticos de IoT em diferentes volumetrias (10 GB, 40 GB e 100 GB), permitindo simular cenários variados de carga e avaliar o comportamento da plataforma.

2. Armazenar os dados em um Azure Data Lake Storage, no formato original e em Delta Lake.

3. Criar pipelines com Apache Spark, utilizando o Azure Synapse, para transformação e processamento de dados.

4. Aplicar a arquitetura de camadas *Medallion* (Bronze, Silver e Gold).

5. Disponibilizar os dados tratados por meio de uma *One Big Table* (OBT) na camada Gold.

1.4 Hipóteses

Este estudo é guiado pelas seguintes hipóteses de engenharia:

- H1: A utilização de serviços gerenciados (PaaS) com armazenamento desacoplado (*Data Lake*) permitirá uma escalabilidade linear de custos em relação ao volume de dados.
- H2: A latência de orquestração do Azure Data Factory será constante e independente da volumetria, tornando-se desprezível em cargas de trabalho de longa duração (*Batch*), mas limitante para cenários de tempo real.

1.5 Organização e estrutura

Este trabalho está organizado em nove capítulos, conforme descrito a seguir:

- **Capítulo 1 – Introdução:** Apresenta o contexto geral do projeto, a motivação, a justificativa, a definição do problema, a questão de pesquisa, os objetivos e a estrutura do documento.
- **Capítulo 2 – Conceitos Fundamentais:** Discute os principais conceitos relacionados à engenharia de dados moderna, abordando ingestão, processamento, organização em camadas, evolução dos repositórios analíticos, modelos arquiteturais (Lambda, Kappa e *Medallion*), além de fundamentos de *Data Lakehouse*, governança e boas práticas de engenharia.
- **Capítulo 3 – Trabalhos Relacionados:** Apresenta estudos, abordagens e soluções identificadas na literatura que dialogam com a proposta deste trabalho, destacando contribuições, limitações e lacunas relevantes.
- **Capítulo 4 – Tecnologias Envolvidas:** Descreve as tecnologias que compõem a plataforma desenvolvida, com ênfase nos serviços da nuvem Microsoft Azure, detalhando suas funcionalidades, características e justificativas para sua adoção no projeto.
- **Capítulo 5 – Metodologia:** Expõe o delineamento metodológico do estudo, incluindo a caracterização da pesquisa, o uso de estudo de caso, a abordagem quantitativa e a forma de condução dos experimentos.
- **Capítulo 6 – Configuração dos Experimentos:** Detalha o ambiente experimental utilizado, contemplando a geração dos dados sintéticos, o provisionamento da infraestrutura via Terraform e a configuração dos recursos necessários para a execução dos pipelines.
- **Capítulo 7 – Resultados:** Apresenta os resultados experimentais obtidos, analisando o desempenho, a escalabilidade e o comportamento da arquitetura sob diferentes volumetrias e etapas do pipeline.

- **Capítulo 8 – Discussão e Análise:** Interpreta criticamente os resultados apresentados, destacando implicações práticas, limitações observadas e o impacto das decisões de arquitetura na eficiência e na previsibilidade operacional.
- **Capítulo 9 – Principais Contribuições:** Sintetiza os achados centrais do trabalho, evidenciando as contribuições técnicas, metodológicas e práticas decorrentes da implementação da plataforma proposta.
- **Capítulo 10 – Trabalhos Futuros:** Apresenta oportunidades de evolução, extensões potenciais e novas linhas de investigação relacionadas a arquiteturas de dados modernas em nuvem.

Parte II

Referencial teórico

2 Conceitos Fundamentais

2.1 ETL, ELT e a Construção de Pipelines de Dados

2.1.1 ETL e ELT

Em engenharia de dados, lidamos constantemente com a movimentação, organização e processamento de dados provenientes de diferentes fontes e com diferentes níveis de qualidade. Esses dados, antes de se tornarem úteis para análises ou aplicações operacionais, precisam passar por etapas que assegurem padronização, consistência e rastreabilidade. Nesse contexto, duas abordagens tradicionais se destacam: ETL (*Extract, Transform, Load*) e ELT (*Extract, Load, Transform*) (SEENIVASAN, 2022).

Embora o termo "extração" faça parte da siglas ETL e ELT, arquiteturas modernas utilizam o conceito mais amplo de ingestão. Segundo (REIS; HOUSLEY, 2022), ingestão é o processo de mover dados da origem para o ambiente analítico, englobando transporte, serialização, destino, tratamento mínimo e registro operacional. Assim, a ingestão engloba a extração, mas não se limita a ela.

A distinção entre ETL e ELT está diretamente relacionada ao ponto em que ocorre a carga dos dados. No modelo ETL tradicional (Figura 1), amplamente empregado em ambientes relacionais, os dados eram extraídos e transformados antes de serem carregados no *Data Warehouse* (KIMBALL; ROSS, 2013). Essa abordagem prioriza limpeza e padronização antecipada, porém tende a introduzir latência e limitações de escalabilidade, especialmente quando aplicada a grandes volumes ou a formatos de dados heterogêneos.

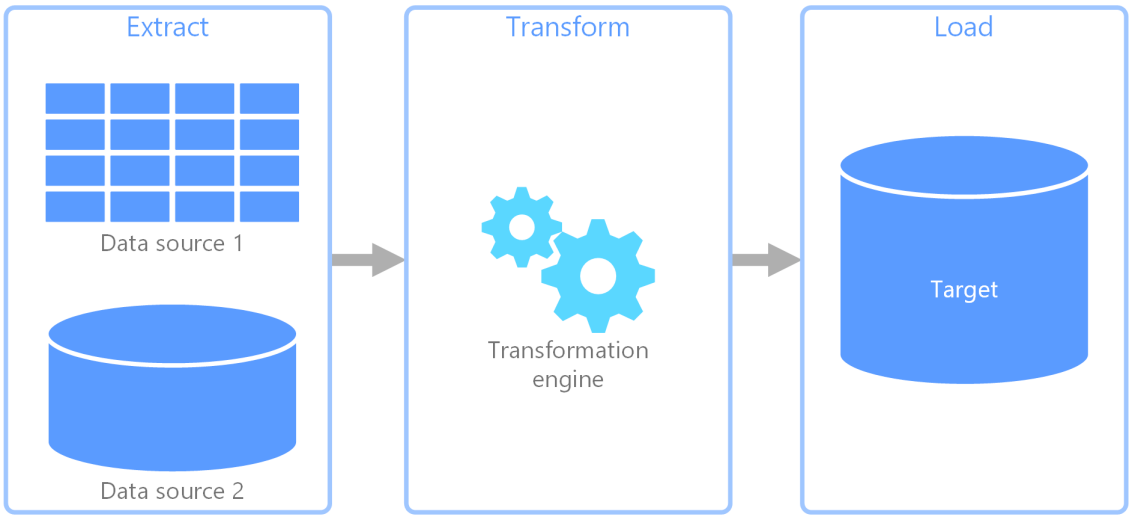

Figura 1 – ETL. Fonte: Adaptado de (MICROSOFT, 2025).

Com a consolidação de arquiteturas em nuvem, o aumento da diversidade de formatos e a disponibilidade de motores de processamento distribuído, como o Apache Spark, ganhou força o modelo ELT (Figura 2). Nele, os dados são carregados em sua forma bruta para ambientes como *Data Lakehouses*, e somente então passam pelo processamento e pelas transformações necessárias (REIS; HOUSLEY, 2022).

Ademais, o processamento (ou transformação) assume papel central no modelo ELT, sendo responsável por converter os dados brutos em representações estruturadas e analiticamente úteis. Inclui padronização de tipos, controle de schema, deduplicação e agregações. Essa etapa sustenta a organização por camadas da arquitetura *Medallion*, que progride de Bronze para Silver e Gold.

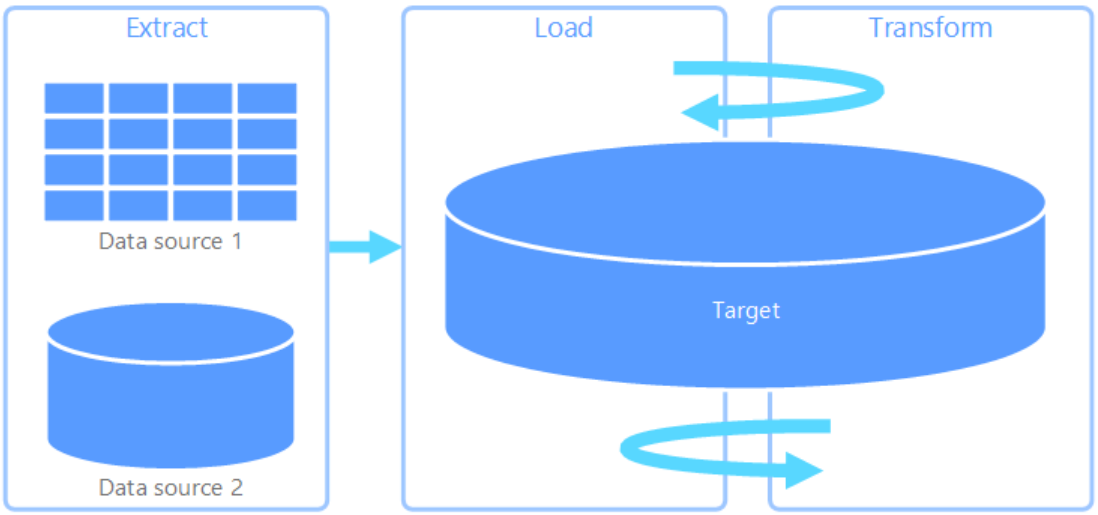

Figura 2 – ELT. Fonte: Adaptado de (MICROSOFT, 2025).

2.1.2 Construção de Pipelines de Dados

Os pipelines de dados compreendem fluxos automatizados responsáveis por conduzir a movimentação e a organização das informações em uma plataforma analítica (Figura 3). Esses fluxos estruturam a passagem dos dados por diferentes etapas do ciclo de engenharia, como atividades de ingestão, carregamento, processamento e disponibilização, a partir de regras previamente definidas (NARAYANAN, 2024).

Em arquiteturas modernas, como *Data Lakes* e *Data Lakehouses*, os pipelines assumem papel fundamental ao permitir a execução dessas etapas de forma abstraída, previsível e reprodutível, independentemente da volumetria ou da complexidade das fontes envolvidas. Essa abstração favorece a padronização do fluxo de dados e reduz a necessidade de intervenções manuais.

Além disso, os pipelines permitem utilizar orquestradores dedicados, que realizam a coordenação das tarefas, o controle de dependências, a observabilidade e o monitoramento

das execuções. Esse controle centralizado facilita a identificação de falhas, a criação de rotinas de reprocessamento e a aplicação de boas práticas de automação, tornando o fluxo de dados mais confiável e alinhado aos requisitos de escalabilidade e governança dos ambientes analíticos contemporâneos.

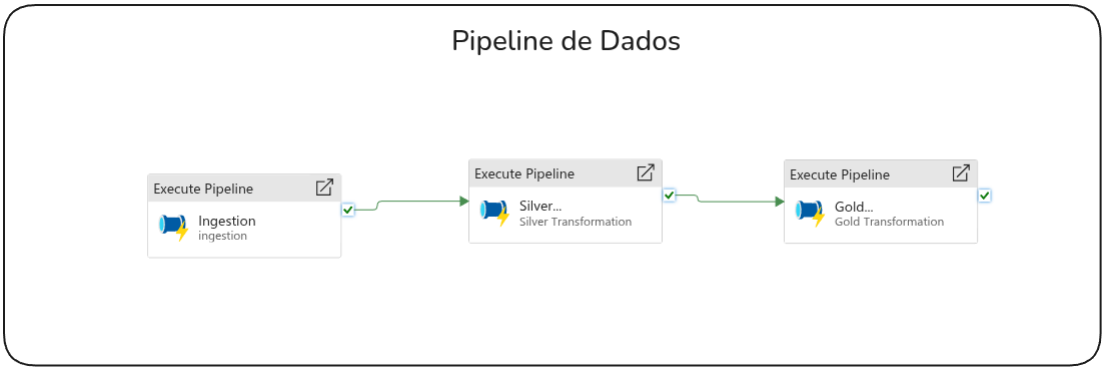

Figura 3 – Pipeline de Dados no Azure Data Factory. Imagem gerada pelo próprio autor.

2.2 Fontes de Dados

A definição das fontes de dados é um elemento central na construção de uma plataforma de dados por determinar as características estruturais, volumétricas e semânticas dos conjuntos processados ao longo da arquitetura. No contexto deste trabalho, optou-se por utilizar dados sintéticos de IoT, que reproduzem padrões típicos de sistemas de telemetria, como medições contínuas, geração de eventos e atributos relacionados ao estado operacional de dispositivos.

Esse tipo de dado é amplamente utilizado em pesquisas aplicadas, como em (AHMED et al., 2020), (ABOUZAID et al., 2025) e (ZAHARIA et al., 2016), por permitir a criação de cenários controlados e a reprodução precisa de diferentes níveis de carga. Assim, permite-se avaliar o comportamento do pipeline sob múltiplas volumetrias sem depender de fontes externas ou de variações imprevisíveis de dados reais.

2.2.1 Geração de Dados Sintéticos

Para viabilizar os experimentos e garantir consistência entre os cenários avaliados, foi implementado um mecanismo de geração de dados sintéticos de IoT utilizando Apache Spark. O gerador produz arquivos em formato Parquet, seguindo um esquema compatível com eventos de telemetria, incluindo identificadores de dispositivos, métricas ambientais, indicadores operacionais e carimbos temporais.

A geração é realizada em modo *Batch*, com controle explícito da volumetria, permitindo criar conjuntos de 10 GB, 40 GB e 100 GB de forma determinística. Essa abordagem

possibilita avaliar o impacto do volume sobre o desempenho da plataforma, além de garantir reprodutibilidade experimental, uma vez que os mesmos dados podem ser gerados novamente sob as mesmas condições.

2.3 Ingestão de Dados

A ingestão de dados é a etapa inicial de um pipeline de dados. É por meio dela que se extrai informações de uma ou mais fontes e se carrega em outro ambiente, como um *Data Lakehouse*. Essa etapa garante que os dados cheguem ao destino de forma contínua, segura e consistente, servindo de base para as próximas fases da arquitetura analítica.

2.3.1 Modalidades de Ingestão: *Batch* e Streaming

O processamento em *Batch* ocorre em intervalos definidos, agrupando dados em blocos antes da execução das operações. Essa modalidade é adequada quando a latência tolerada é de minutos ou horas, o volume de dados é elevado e há necessidade de reprocessamentos consistentes, como em rotinas de consolidação periódica ou integrações agendadas (MEEHAN et al., 2017).

O streaming, por sua vez, trata eventos de forma contínua, assim que eles chegam ao sistema. Essa abordagem prioriza baixa latência sendo indicada para cenários que dependem de respostas imediatas, como monitoramento em tempo quase real, detecção de anomalias e processos sensíveis ao tempo (MEEHAN et al., 2017).

Tabela 1 – Diferenças práticas entre *Batch* e Streaming

Aspecto	***Batch***	**Streaming**
Latência	Minutos → horas	Milissegundos → segundos
Custo	Mais barato por volume	Maior custo por volume
Consistência	Reprocesso simples	Tratar ordem e duplicação
Complexidade	Menor	Maior
Casos típicos	Relatórios D-1	Alertas, Monitoramento

2.3.2 Estratégias de Carga: Total e Incremental

Em pipelines de dados, a estratégia de carga define como os dados são introduzidos ou atualizados em um sistema de armazenamento ou processamento. As duas abordagens mais comuns são a carga *total* e a carga *incremental*.

A **carga total** consiste na substituição completa do conjunto de dados a cada execução. Todo o conteúdo da fonte é lido e reprocessado, independentemente de alterações. Essa abordagem é simples de implementar e garante um estado final sempre reconstruído

do zero, porém implica maior consumo de recursos e tempo de processamento à medida que a volumetria cresce.

A **carga incremental** processa apenas os registros novos, modificados ou removidos desde a última carga. Para isso, depende de mecanismos como carimbos de tempo, colunas de auditoria, logs de alteração ou estruturas de metadados que permitam identificar deltas. Essa estratégia reduz o volume de dados processado, melhora a eficiência operacional e diminui a latência, mantendo o estado do sistema continuamente atualizado.

A escolha entre carga total e incremental impacta diretamente o desempenho, o custo e a consistência das operações de dados, sendo um elemento fundamental na modelagem de pipelines escaláveis e confiáveis.

2.4 Processamento de Dados

2.4.1 Apache Hadoop

O Apache Hadoop é um framework open source desenvolvido pela Apache Foundation para o processamento distribuído e escalável de grandes volumes de dados.O Hadoop permite dividir tarefas massivas de processamento em blocos menores, distribuindo-os por diversos nós de um cluster, garantindo alta disponibilidade e tolerância a falhas (AZEROUAL; FABRE, 2021). Sua arquitetura é composta principalmente pelo Hadoop Distributed File System (HDFS), responsável pelo armazenamento redundante dos dados, e pelo MapReduce, modelo de processamento paralelo baseado em funções de mapeamento e redução (AZEROUAL; FABRE, 2021).

Essa abordagem oferece um ganho substancial de desempenho em aplicações analíticas de larga escala, especialmente em cenários de dados heterogêneos e de rápida expansão, como os observados durante a pandemia da COVID-19, quando a necessidade de processar fluxos massivos e não estruturados de informações tornou-se crítica (AZEROUAL; FABRE, 2021).

2.4.2 Apache Spark

Com o grande volume de dados existentes, processar tornou-se uma tarefa complexa e passou a exigir soluções distribuídas e escaláveis. Nesse contexto, o Apache Spark se apresenta como um mecanismo de computação em cluster projetado para ser rápido e generalista. Sua arquitetura permite distribuir os processamentos de dados entre múltiplos nós, possibilitando a execução de múltiplas tarefas de maneira paralela e eficiente (CHAMBERS; ZAHARIA, 2018).

O Apache Spark adota a arquitetura driver–executors: o driver coordena a aplicação, planejando e distribuindo as tarefas, enquanto os executors executam essas tarefas

em paralelo, armazenam dados em cache e retornam resultados ao driver 4.

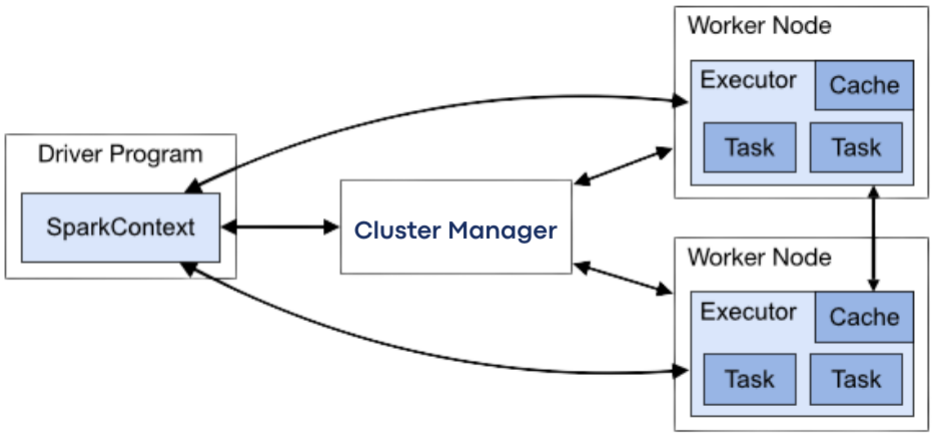

Figura 4 – Apache Spark. Fonte: Adaptado de (FOUNDATION, 2025).

Ao contrário de abordagens baseadas em disco, como o Hadoop MapReduce, o Spark opera majoritariamente em memória, proporcionando ganhos significativos de desempenho (MOSTAFAEIPOUR et al., 2021). Além disso, o Spark oferece uma série de funcionalidades e abstrações que facilitam o desenvolvimento e a otimização.

Uma das principais abstrações do Spark é o RDD (*Resilient Distributed Dataset*) que representa uma coleção imutável de objetos distribuídos por um cluster. Como funcionalidades, o Spark oferece APIs avançadas, como o *DataFrame* e o Spark SQL, que permitem consultas declarativas, semelhantes a SQL, sobre dados estruturados (CHAMBERS; ZAHARIA, 2018).

2.5 Arquiteturas de Processamento de Dados

As arquiteturas de dados surgiram para atender aos desafios crescentes de processamento em larga escala, sobretudo diante da popularização de fluxos de dados contínuos e variados. Entre as abordagens mais discutidas na literatura (LIN, 2017) (FEICK; KLEER; KOHN, 2018) (KIRAN et al., 2015), estão as arquiteturas Lambda e Kappa, ambas voltadas a sistemas que exigem o processamento eficiente de dados em tempo real ou quase em tempo real.

Por outro lado, a abordagem *Medallion*, embora também seja considerada uma arquitetura, se diferencia por sua natureza lógica e conceitual. Em vez de definir uma infraestrutura técnica ou operacional, a arquitetura *Medallion* propõe uma organização em camadas (Bronze, Silver e Gold) com foco na progressão da qualidade, tratabilidade e confiabilidade dos dados ao longo do pipeline analítico (DATABRICKS, 2025a).

2.5.1 Arquitetura Lambda

A arquitetura Lambda visa combinar o processamento em lote (*Batch*) com o processamento em tempo real (streaming), de modo que o *pipeline* tem duas camadas distintas: uma camada de *Batch*, com resultados completos e precisos, e uma camada de velocidade que fornece informações com menor latência (KIRAN et al., 2015).

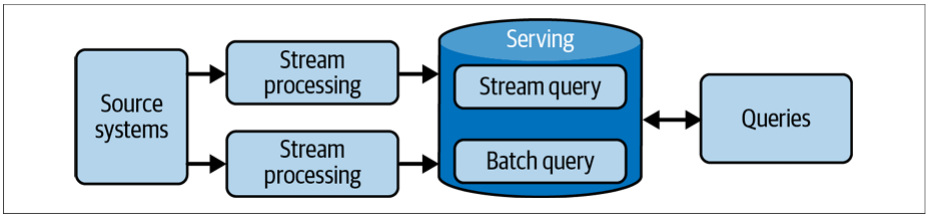

Figura 5 – Arquitetura Lambda. Fonte: Adaptado de (REIS; HOUSLEY, 2022).

2.5.2 Arquitetura Kappa

A arquitetura Kappa foi criada como uma simplificação da Lambda, eliminando a camada de *Batch* e contendo somente um fluxo principal em tempo real. Nessa abordagem, o mesmo *pipeline* de streaming é utilizado tanto para ingestão contínua e real-time como para reprocessamentos históricos, favorecendo sua aplicação em casos onde a baixa latência é essencial e os dados são produzidos em formato de log imutável, como em sistemas de eventos ou streaming contínuo. (REIS; HOUSLEY, 2022).

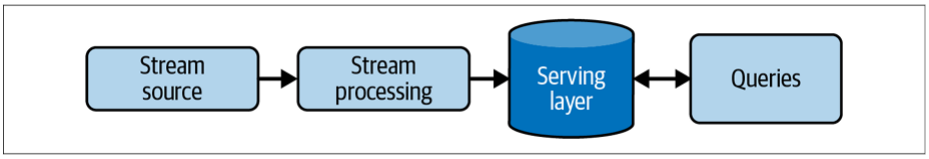

Figura 6 – Arquitetura Kappa. Fonte: Adaptado de (REIS; HOUSLEY, 2022).

2.5.3 Arquitetura *Medallion*

Diferente das arquiteturas anteriores, a *Medallion* propõe uma organização por camadas (Bronze, Silver e Gold) em um *Data Lake*, focando no tratamento incremental e progressivo dos dados. Essa abordagem é especialmente adequada para cenários orientados a dados históricos, como análises exploratórias, modelos preditivos e *business intelligence* (STRENGHOLT, 2025). A simplicidade de manutenção e a clareza do fluxo de dados tornam essa arquitetura atrativa em ambientes que priorizam transparência, governança e reprocessamento eficiente.

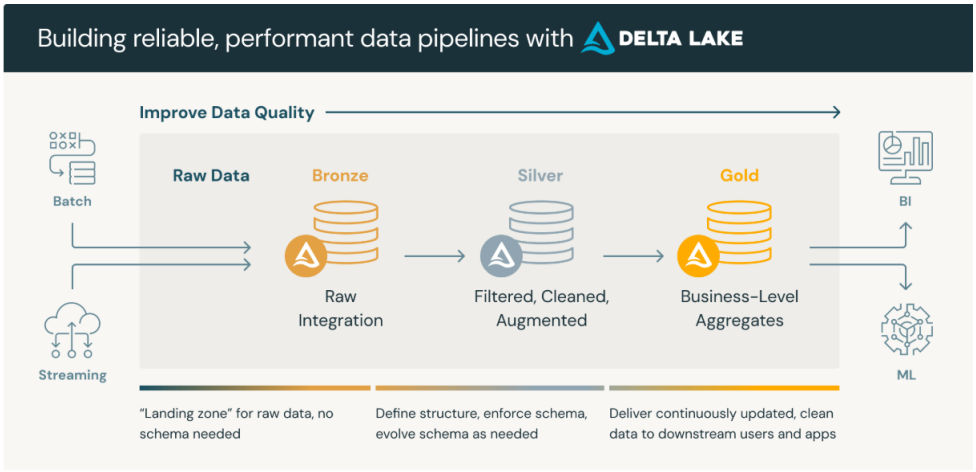

Figura 7 – Arquitetura *Medallion*. Fonte: Adaptado de (DATABRICKS, 2025a).

2.5.4 Justificativa da Escolha da Arquitetura *Medallion*

A escolha pela arquitetura *Medallion* neste projeto se deve à natureza *Batch* dos dados extraídos das APIs, bem como à facilidade de integração com os serviços da plataforma Azure, como o Data Lake Storage e o Synapse Analytics. Diferentemente das arquiteturas Lambda e Kappa, sendo mais indicadas para sistemas com alto volume de dados e requisitos de latência mínima, a *Medallion* oferece uma estrutura lógica mais adequada para o controle, rastreabilidade e evolução dos dados ao longo do tempo, sem a complexidade de manter pipelines redundantes ou sistemas de streaming em tempo real.

É importante destacar que essas abordagens não são excludentes. A arquitetura *Medallion* pode ser combinada com estratégias baseadas em Lambda ou Kappa, sobretudo em cenários que envolvem fluxos contínuos em tempo real e lote. No contexto deste estudo, a natureza dos dados e os requisitos de processamento favorecem a adoção isolada da abordagem *Medallion* como solução mais simples, eficiente e alinhada ao objetivo de organizar os dados em camadas de qualidade crescente.

2.6 Formatos de Arquivo

Em cenários modernos de engenharia e análise de dados, a escolha do formato de arquivo é um fator determinante para o desempenho, escalabilidade e custo das soluções. Formatos tradicionais, como CSV e JSON, embora amplamente utilizados, não são ideais para operações analíticas em larga escala, pois não oferecem compressão eficiente, leitura paralela otimizada, nem mecanismos internos de metadados que permitam o particionamento e o processamento seletivo de colunas. Por isso, surgiram formatos como Avro, ORC e Parquet, que adotam recursos relevantes para resolver diferentes problemas.

2.6.1 Avro

O Avro é um formato de arquivo projetado para armazenamento eficiente de dados estruturados e troca entre sistemas distribuídos. Baseado em esquemas definidos em JSON, o Avro permite a serialização compacta dos dados em binário, reduzindo a dimensão dos arquivos e acelera a leitura e escrita (REIS; HOUSLEY, 2022). É um formato criado pela Apache Software Foundation e possui suporte em diversos provedores de nuvem, é amplamente utilizado em ambientes Hadoop.

O formato Avro é muito utilizado em *pipelines* de streaming e mensageria, como os que envolvem Apache Kafka, devido à sua leveza e capacidade de transmitir registros estruturados com alta desempenho. Embora seja eficiente para o transporte e a ingestão de grandes volumes de dados, o Avro não é o formato mais indicado para consultas analíticas complexas, ao armazenar os dados em formato de linha, dificultando a leitura seletiva de colunas e impacta o desempenho em operações de agregação e varredura em larga escala.

2.6.2 ORC

Por armazenar os dados de forma colunar, o ORC (*Optimized Row Columnar*) permite que as ferramentas de análise leiam somente as colunas necessárias para uma consulta específica, reduzindo significativamente a quantidade de dados processados e o tempo de leitura em operações analíticas. Além disso, o ORC oferece recursos nativos de compressão, melhorando a eficiência de armazenamento (REIS; HOUSLEY, 2022).

No entanto, o formato ORC não é otimizado para o framework de processamento Apache Spark, devido aos seus mecanismos de particionamento e gerenciamento. Apesar de não ser muito utilizado em ambientes Spark, esse formato é bastante utilizado em ambientes Hive e Hadoop.

2.6.3 Parquet

O Apache Parquet é um formato colunar projetado para otimizar a leitura, a compressão e o processamento de grandes volumes de dados em ambientes distribuídos. Por adotar um modelo colunar, o Parquet permite que somente os dados relevantes para uma consulta sejam lidos, reduzindo significativamente o tempo de processamento, o uso de memória e o tráfego de I/O (ARMBRUST; DAS et al., 2020). Esse comportamento o torna altamente eficiente para *workloads* analíticas e aplicações em larga escala.

Um grande ponto de destaque é sua ampla compatibilidade com frameworks e motores de processamento distribuído, como Apache Spark, no qual se consolidou como o formato padrão de armazenamento otimizado em *Data Lakes* modernos.

2.7 Formatos de Armazenamento de Dados

Enquanto os formatos de arquivo tratam da estrutura física dos dados em disco, os formatos de armazenamento operam em um nível mais alto, adicionando camadas de metadados e logs transacionais que permitem controle de versão, consistência e governança.

Os formatos de armazenamento combinam a desempenho dos formatos colunares com propriedades ACID, possibilitando operações de atualização, exclusão e inserção seguramente. Entre os principais formatos utilizados estão o Delta Lake e o Apache Iceberg.

2.7.1 Delta Lake

É um formato de armazenamento transacional desenvolvido pela Databricks, que estende o formato Parquet com uma camada de logs de transação armazenada na estrutura de arquivos 8. Essa camada garante propriedades ACID, permitindo que operações de escrita e leitura ocorram de forma consistente e segura em ambientes distribuídos (DATABRICKS, 2025c).

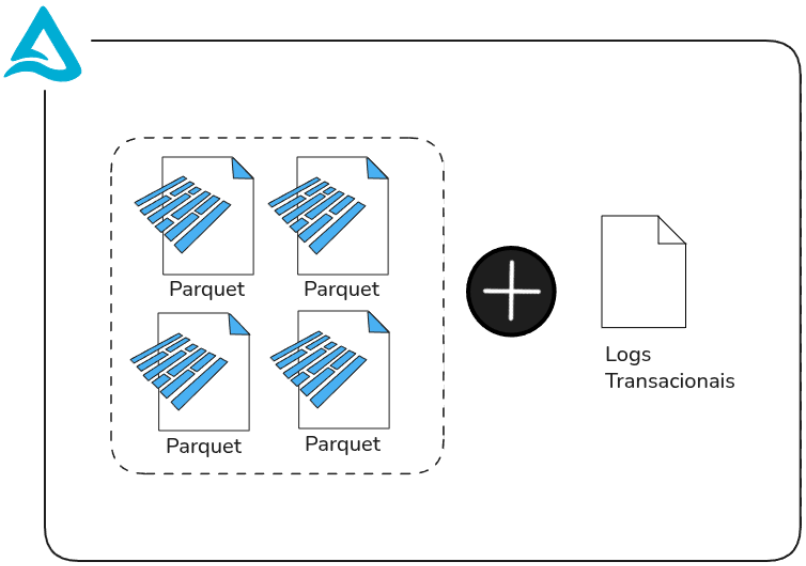

Figura 8 – Delta Lake. Fonte: Imagem gerada pelo próprio autor.

Além disso, o Delta Lake fornece recursos como *time travel* (acesso a versões anteriores dos dados), *schema enforcement* e *schema evolution* (ARMBRUST; DAS et al., 2020). Essas características tornam o Delta Lake uma das bases do conceito de *Lakehouse*, ao unir a escalabilidade dos *Data Lakes* com a governança e confiabilidade típicas dos *Data Warehouses*.

2.7.2 Apache Iceberg

É um formato de armazenamento aberto desenvolvido originalmente pela Netflix e posteriormente adotado pela Apache Foundation. Assim como o Delta Lake, o Iceberg adiciona uma camada de metadados e logs transacionais sobre arquivos em formatos colunares como Parquet e ORC, garantindo operações ACID em larga escala (REIS; HOUSLEY, 2022).

Além disso, o Iceberg oferece versionamento de dados, *schema evolution* sem recriação de tabelas e compatibilidade com múltiplos motores de processamento, como Apache Spark, Trino e Snowflake. Sua arquitetura neutra e orientada a metadados o torna uma alternativa robusta para organizações que buscam independência de fornecedor e interoperabilidade em seus *Data Lakes*.

2.8 Modelagem de Dados Analítica

Em ambientes analíticos, é fundamental adotar modelagens otimizadas para leitura e consulta, garantindo eficiência no acesso e na exploração das informações. Ao longo dos anos, diversos pesquisadores propuseram abordagens para solucionar esse desafio, entre os quais se destacam Bill Inmon e Ralph Kimball.

2.8.1 A Abordagem de Bill Inmon

O modelo de Bill Inmon adota uma abordagem normalizada, baseada no modelo entidade-relacionamento (ER), na qual os dados são estruturados em múltiplas tabelas interligadas por chaves primárias e estrangeiras. Essa forma de modelagem busca eliminar redundâncias, preservar a integridade referencial e garantir consistência lógica entre as entidades do sistema, aproximando-se de uma modelagem tradicional de banco de dados transacional, porém aplicada ao contexto analítico (INMON, 2002).

O foco dessa abordagem está na representação fiel e detalhada dos domínios de negócio, permitindo uma compreensão precisa das relações entre entidades e atributos. Apesar de favorecer a qualidade e coerência dos dados, a modelagem de Inmon tende a resultar em estruturas mais complexas para consulta, ao exigir múltiplas junções. Esse fator motivou o surgimento de novas alternativas, como a proposta por Ralph Kimball.

2.8.2 A Abordagem de Ralph Kimball

A modelagem apresentada por Ralph Kimball adota uma abordagem dimensional, na qual os dados são organizados em tabelas de fatos e dimensões. Diferente da estrutura normalizada de Inmon, o modelo de Kimball é desnormalizado de maneira proposital, priorizando o desempenho em consultas analíticas e a facilidade de interpretação pelos usuários de negócio. As tabelas fato armazenam medidas quantitativas dos processos

(como vendas e volumes), enquanto as dimensões descrevem os contextos dessas medidas (como tempo, produto ou localização) (KIMBALL; ROSS, 2013).

O foco dessa modelagem está em otimizar a leitura e a análise de grandes volumes de dados, simplificando a exploração e favorecendo a criação de relatórios e dashboards. Essa estrutura facilita o uso de ferramentas de BI e consultas analíticas, ao reduzir o número de junções e melhora o tempo de resposta. Embora sacrifique parte da normalização e possa introduzir redundância, a modelagem de Kimball é amplamente utilizada por oferecer alto desempenho, simplicidade e aderência às necessidades analíticas das organizações (KIMBALL; ROSS, 2013).

Enquanto a abordagem de Inmon privilegia a integridade e consistência, a de Kimball prioriza agilidade e acessibilidade, tornando-se a base para grande parte das soluções de *business intelligence* modernas.

2.8.3 A Abordagem *One Big Table* (OBT)

Além das abordagens clássicas de Inmon e Kimball, modelos analíticos modernos têm incorporado uma estratégia alternativa conhecida como *One Big Table* (OBT). Nesta abordagem, os dados são consolidados em uma tabela única, altamente desnormalizada, contendo tanto atributos dimensionais quanto métricas derivadas de múltiplas fontes. A OBT funciona como uma visão materializada e pronta para consumo analítico, reduzindo significativamente a necessidade de junções em tempo de consulta e simplificando a experiência de exploração dos usuários finais.

A principal motivação para o uso de OBTs está associada à performance e à simplicidade operacional em plataformas de dados centradas em processamento distribuído. Ao organizar os dados em uma estrutura unificada, sistemas de consulta conseguem explorar de forma mais eficiente estratégias de particionamento, *predicate pushdown*, compressão e armazenamento colunar, típicas de tecnologias como Delta Lake e outros formatos orientados a análise. Como consequência, consultas analíticas obtêm ganhos expressivos de latência e previsibilidade.

Apesar dessas vantagens, a abordagem OBT implica trade-offs relevantes. A desnormalização extensa introduz redundância de dados, aumentando o custo de armazenamento e exigindo mecanismos de atualização mais robustos para evitar inconsistências. Além disso, OBTs costumam ser construídas na camada Gold em arquiteturas Medallion, funcionando como produtos finais de dados e não como fontes de transformação subsequentes. Dessa forma, seu uso é recomendado quando a prioridade está em maximizar desempenho de leitura e entregar objetos analíticos prontos para consumo, especialmente em cenários de BI, dashboards executivos e modelos preditivos que exigem acesso rápido e consolidado às informações.

Em síntese, enquanto Inmon privilegia consistência, Kimball otimiza a exploração multidimensional e a OBT maximiza a performance de consumo direto. Essa complementaridade torna as três abordagens úteis em diferentes pontos de arquiteturas analíticas modernas, podendo inclusive coexistir dentro da mesma plataforma.

2.9 Evolução dos Repositórios Analíticos

A medida que a utilização de dados analíticos cresce, surge a necessidade de ter um ambiente centralizado, confiável, acessível e resiliente, um ambiente que possa ser tratado como uma única fonte da verdade (INMON, 2002). Esse ambiente emerge com o nome de *Data Warehouse*, um repositório centralizado e confiável que logo ganha tração no mercado.

Quando os primeiros *Data Warehouses* foram concebidos, os dados analíticos tinham origem predominantemente em sistemas transacionais. Por isso, essas estruturas foram implementadas em bancos de dados relacionais, como o SQL Server, que oferecem consistência e suporte a consultas estruturadas. A partir de modelagens analíticas, os bancos relacionais passaram a armazenar os dados de forma mais eficiente para leitura e consulta, otimizando o desempenho das análises (KIMBALL; ROSS, 2013).

No entanto, com o avanço da tecnologia e o aumento da variedade dos dados, especialmente os não estruturados, os bancos relacionais tornaram-se insuficientes como base tecnológica para serem utilizados como *Data Warehouse*. Diante dessa limitação, surge o *Data Lake*, uma estrutura projetada para armazenar dados em diversos formatos e níveis de estruturação, funcionando como um repositório centralizado e escalável.

Os *Data Lakes* possibilitam armazenar os mais diversos formatos de dados, como imagens, vídeos e fotos. Entretanto, por não possuir recursos de governança e ACID, não cumprem os requisitos necessários para ser um repositório analítico centralizado confiável. Enxergando a necessidade de ter uma flexibilidade no armazenamento de dados em um ambiente analítico, surge a arquitetura de *Lakehouse*.

Com a incorporação de propriedades ACID e mecanismos de governança aos ambientes de *Data Lake*, permitiu-se combinar a escalabilidade e flexibilidade dessas estruturas com a consistência e confiabilidade típicas dos *Data Warehouses*, originando o conceito de *Data Lakehouse*.

2.10 Governança, Qualidade e Linhagem dos Dados

A governança de dados consiste no conjunto de processos, políticas e controles responsáveis por garantir que os dados sejam gerenciados de forma eficiente, segura e conforme requisitos legais e regulatórios (ALHASSAN; SAMMON; AND, 2016)(CHEONG; CHANG, 2007).

A qualidade dos dados é um conceito muito importante quando falamos de governança, ela envolve acurácia, atualidade, consistência e confiabilidade. Dados de baixa qualidade são muito perigosos para qualquer organização, pois eles podem ser utilizados como recurso para tomar direcionamentos incorretos (BATINI; SCANNAPIECO, 2014). Por isso, nas camadas Silver e Gold é necessário garantir essa consistência de qualidade por meio de práticas como a padronização de formatos, remoção de duplicidades e tratamento de valores nulos.

Outro conceito fundamental é o de linhagem de dados, que permite rastrear os caminhos que os dados percorreram desde a sua origem até os sistemas analíticos (SYED; NAMPALLI, 2020). Isso é essencial para auditoria e resolução de erros, além de permitir que a plataforma possua uma transparência útil em ambientes regulados. Ferramentas como o Azure Purview, em conjunto com metadados bem definidos e catálogos de dados, permitem manter a rastreabilidade e a transparência necessárias para a governança e auditoria das informações.

2.11 Computação em Nuvem

Nos últimos anos, a computação em nuvem ganhou grande destaque ao possibilitar que empresas de todos os portes pudessem adotar arquiteturas escaláveis, resilientes e de rápida implementação. A computação em nuvem pode ser entendida como a disponibilização de recursos computacionais, como armazenamento, processamento e redes, de forma sob demanda, acessíveis pela internet e cobrados conforme a utilização (MICROSOFT, 2024).

No contexto de engenharia de dados, a computação em nuvem desempenha um papel crucial, ao possibilitar o processamento de grandes volumes de dados com escalabilidade e disponibilidade, além de permitir construir pipelines com maior agilidade e menor esforço operacional. Dentro desse ambiente, os serviços gerenciados se destacam por abstraírem tarefas como manutenção de hardware, balanceamento de carga e implementações básicas de segurança; isso faz com que o time de dados consiga dar mais atenção à modelagem e tratamento de dados.

Diversas empresas disponibilizam serviços de computação em nuvem, sendo as líderes em adoção atualmente a Amazon (AWS), a Google (GCP) e a Microsoft (Azure). Neste trabalho, optou-se pela utilização da Azure, por se tratar de uma plataforma amplamente consolidada no meio corporativo e com um portfólio robusto de serviços voltados à engenharia de dados, como o Azure Data Factory, Azure Data Lake e Azure Synapse. Essa combinação de serviços permite construir pipelines completos, escaláveis e de alto desempenho, desde a ingestão até a visualização dos dados, que se alinha com a solução proposta neste trabalho.

2.12 Comparativo entre Arquiteturas de Dados Clássicas e Modernas

As soluções de dados evoluíram significativamente nas últimas décadas. Enquanto abordagens tradicionais, como *Data Warehouses* e sistemas OLTP, focam em dados estruturados e relatórios analíticos básicos, plataformas modernas, como *Data Lakes* e *Lakehouses*, foram projetadas para lidar com grandes volumes de dados diversos, de forma escalável e integrada. A Tabela 2 apresenta um comparativo entre essas duas abordagens.

Tabela 2 – Arquitetura Tradicional vs Arquitetura Moderna

Característica	**Arquitetura Tradicional**	**Arquitetura Moderna**
Tipos de dados	Predominantemente estruturados e relacionais	Estruturados, semi-estruturados e não estruturados
Formato de armazenamento	Tabelas relacionais	Parquet + camadas ACID (Delta/Iceberg)
Padrão de processamento	ETL	ELT
Latência	Janela fixa em lote	Híbrido: lote + quase tempo real
Escalabilidade	Vertical e Rígida	Horizontal e Elástica
Custo operacional	Alto para expandir; capacidade ociosa	*Pay-as-you-go*; otimização por camadas e autoscale
Governança/Catálogo	Metadados no DW/ETL; catálogo limitado	Catálogo centralizado + *data lineage*
Evolução de schema	Difícil, com downtime	*Schema evolution* e *enforcement* nativo
Integração com ML/AI	Indireta	Nativa
Camadas lógicas	Única	*Medallion* (Bronze/Silver/Gold)

3 Trabalhos Relacionados

Em (MANCHANA, 2023), é apresentada uma análise aprofundada sobre a construção de plataformas modernas de dados em ambientes de nuvem. O trabalho discute o desafio crescente de lidar com grandes volumes de dados heterogêneos em arquiteturas escaláveis e resilientes, propondo a integração de *Data Lakes* e *Data Lakehouses* como alicerce para *workloads* analíticos de alta complexidade. Nessa perspectiva, o autor enfatiza o uso de serviços de nuvem capazes de oferecer armazenamento distribuído, mecanismos de processamento massivo e estratégias robustas de governança e metadados, essenciais para assegurar qualidade, consistência e operacionalidade ao longo de todo o ciclo de vida dos dados.

Em (ZHYRENKOV; DOROSHENKO, 2025), os autores propõem uma arquitetura de dados baseada em componentes modulares, que engloba quatro camadas: ingestão, processamento, armazenamento e disponibilização. A proposta visa superar limitações de arquiteturas monolíticas tradicionais. O modelo integra padrões como Lambda, Kappa e *Medallion Architecture*, conseguindo operar fluxos de ETL e ELT, com suporte a Delta Lake e Apache Iceberg. O estudo apresenta implementações reais com Apache Spark, Trino e Apache Kafka, demonstrando a aplicabilidade do modelo em cenários de dados em nuvem com múltiplas fontes.

Em (ABOUZAID et al., 2025),os autores apresentam a construção de uma plataforma moderna de dados baseada na arquitetura de *Data Lakehouse* na nuvem. O trabalho discute os limites das arquiteturas tradicionais (*Data Lake* e *Data Warehouse*), assim como a implementação completa, desde a orquestração até a disponibilização dos dados. O estudo relata como formatos abertos, como o Delta Lake, ampliam a interoperabilidade e reduzem dependência de plataformas pagas. Além disso, os resultados dos testes apresentados no artigo demonstram ganhos consistentes de desempenho em cenários de consultas analíticas, o que é relevante considerando a função da plataforma.

Em (BHUVA, 2025), é apresentada uma implementação prática de um *Lakehouse* corporativo construído integralmente no ecossistema Azure, utilizando Azure Databricks, Delta Lake e a arquitetura *Medallion* como estrutura central de organização dos dados. O artigo descreve, detalhadamente, as camadas Bronze, Silver e Gold, o uso de pipelines de ingestão via Azure Data Factory, além da incorporação de governança unificada por meio do Unity Catalog, que ilustra o fluxo governado entre ADLS Gen2, Delta Lake e Databricks. A contribuição do estudo reside na demonstração de ganhos concretos de desempenho e governança em ambiente real, com reduções expressivas no tempo de consulta, melhoria significativa na qualidade dos dados e otimizações de custo decorrentes de

processamento distribuído e estratégias de autoscaling. Cabe destacar que a solução dos autores se apoia no Azure Databricks, enquanto este trabalho utiliza o Synapse Spark nativo, oferecendo uma perspectiva distinta para organizações que buscam integração plena com serviços nativos do Azure e uma alternativa sem contratação adicional do Databricks. Essa abordagem conecta-se diretamente ao presente trabalho, ao evidenciar, em um contexto de produção, os mesmos princípios adotados neste estudo: organização baseada na arquitetura *Medallion*, processamento distribuído com Spark, armazenagem transacional com Delta Lake e integração nativa com serviços Azure para entregar uma plataforma unificada, escalável e governada.

Em (MOHNA et al., 2022), os autores realizam uma revisão sistemática abrangente sobre arquiteturas modernas de engenharia de dados, com ênfase especial na adoção da arquitetura *Medallion* e no papel de pipelines orientados a IA em ambientes de nuvem. O estudo analisa 106 publicações relevantes e identifica um movimento consistente na direção de plataformas que combinam armazenamento em *Data Lakes*, processamento distribuído e modelos ELT em grande escala. Entre as conclusões, o artigo destaca a prevalência crescente da arquitetura *Medallion* (Bronze–Silver–Gold) como padrão dominante para organização, governança e refinamento progressivo dos dados, apontando seus benefícios na redução de redundância, na reprodutibilidade de experimentos e na integração entre engenharia e ciência de dados. Essa discussão converge diretamente com o presente trabalho, que implementa uma plataforma moderna de dados em Azure Synapse utilizando Spark e camadas *Medallion* para demonstrar a eficiência de um ambiente unificado, escalável e orientado à inteligência analítica.

Para situar a contribuição deste trabalho frente ao estado da arte, a Tabela 3 apresenta um comparativo entre as abordagens analisadas. Observa-se que, embora a arquitetura de *Data Lakehouse* e o modelo *Medallion* sejam amplamente discutidos e implementados em soluções proprietárias (como Databricks), existe uma lacuna significativa na literatura quanto à disponibilidade de implementações reprodutíveis via Infraestrutura como Código (IaC) e, principalmente, quanto à análise quantitativa de custos operacionais (FinOps) por gigabyte processado, diferenciais centrais desta pesquisa.

Tabela 3 – Comparativo entre Trabalhos Relacionados e a Proposta Atual

Autor/Ano	**Abordagem**	**Plataforma**	**Arquitetura *Medallion*?**	**Usa IaC (*Terraform*)?**	**Análise Quantitativa de Custos?**
Manchana (2023)	Teórica/Conceitual	Genérica (Cloud)	Sim (*Lakehouse*)	Não	Não
Zhyrenkov; Doroshenko (2025)	Proposta Modular	Multi-Cloud	Sim	Não	Não
Abouzaid et al. (2025)	Implementação	Cloud-Native	Sim	Não	Não
Bhuva (2025)	Estudo de Caso	Azure (Databricks)	Sim	Não	Parcial (Qualitativa)
Mohna et al. (2022)	Revisão Sistemática	N/A	Sim	Não	Não
Este Trabalho (2025)	**Experimental**	**Azure (Synapse)**	**Sim**	**Sim**	**Sim (Detalhada)**

Fonte: Elaborado pelo autor.

Em síntese, os trabalhos analisados evidenciam uma convergência clara em torno de práticas consolidadas na engenharia de dados moderna, especialmente no uso de arquiteturas distribuídas, modelos em camadas como a Arquitetura *Medallion* e formatos

abertos capazes de integrar processamento e governança de maneira unificada. Apesar de abordarem contextos distintos, desde arquiteturas baseadas em componentes até implementações corporativas completas e revisões sistemáticas de padrões emergentes, todos reforçam a necessidade de plataformas escaláveis, governadas e orientadas a análises de alto desempenho. O presente trabalho se insere nesse panorama ao demonstrar, em um ambiente real construído com Azure Synapse e Apache Spark, a aplicação prática desses princípios, apresentando uma avaliação empírica dos impactos em desempenho, custo e escalabilidade. Dessa forma, este estudo contribui ao fornecer evidências quantitativas e um arcabouço replicável que reforça o papel das arquiteturas *Lakehouse* e *Medallion* na construção de plataformas modernas de dados em nuvem.

4 Tecnologias Envolvidas

4.1 Versionamento com Git

O versionamento de artefatos é um pilar fundamental para a governança e a manutenção de plataformas de dados modernas. O Git possibilita ter um sistema de controle de versões em múltiplos componentes na nuvem, como pipelines de orquestração, scripts de ingestão e códigos de infraestrutura. A partir da adoção do versionamento, é possível assegurar rastreabilidade, colaboração e reprodutibilidade.

No contexto do trabalho proposto, o versionamento foi aplicado a três dimensões principais da plataforma:

- **Pipelines de Orquestração (Azure Data Factory)**
- **Infraestrutura como Código (Terraform)**
- **Notebooks de Processamento de Dados com PySpark**

4.2 Infraestrutura com *Terraform*

A utilização de IaC (Infrastructure as Code) é fundamental no desenvolvimento de sistemas em nuvem, especialmente em arquiteturas modernas de dados. A Infraestrutura como Código permite definir, provisionar e gerenciar recursos de nuvem por meio de código declarativo. No contexto da plataforma de dados desenvolvida, adotou-se o Terraform, ferramenta *open source* amplamente utilizada no mercado que oferece suporte a múltiplos provedores de nuvem, incluindo a Microsoft Azure (HASHICORP, 2025).

O Terraform permite que todos os componentes da arquitetura, como Resource Groups, Storage Accounts, Data Factories, Synapse Workspaces e Key Vaults, sejam descritos em linguagem HCL. Essa abordagem garante consistência entre ambientes, facilita o versionamento via Git e proporciona controle total sobre o ciclo de vida da infraestrutura, desde a criação até a destruição.

No pipeline proposto, o Terraform desempenha papel fundamental em todas as etapas, uma vez que ele gerencia o provisionamento de todos os recursos utilizados na arquitetura, desde a ingestão até a disponibilização dos dados na camada Gold. Dessa forma, a infraestrutura torna-se modular, reprodutível e auditável, reduzindo erros humanos e acelerando os deployments.

Ademais, o uso de blocos modulares e variáveis parametrizadas utilizando HCL permite reaproveitar configurações e adaptações da arquitetura em diferentes ambientes (desenvolvimento, teste e produção). A execução do código se dá em três etapas principais (HASHICORP, 2025):

- `Terraform init`: inicializa o ambiente e baixa os provedores necessários;
- `Terraform plan`: gera um plano de execução mostrando as alterações que serão aplicadas;
- `Terraform apply`: aplica as mudanças definidas no plano de execução.

No Terraform, também é possível desprovisionar todos os ambientes criados, basta utilizar o comando `Terraform destroy` (HASHICORP, 2025).

4.3 Serviços Microsoft Azure na Engenharia de Dados

4.3.1 Resource Groups

No Microsoft Azure, os recursos provisionados são organizados logicamente em Resource Groups. Os Resource Groups são fundamentais para o controle e o gerenciamento dos recursos utilizados no projeto, além de facilitarem a visualização e o acompanhamento dos custos associados.

No projeto descrito, é criado um grupo de recursos que possui todos os recursos necessários para o funcionamento da arquitetura, sendo possível gerenciar os recursos de forma agrupada com facilidade.

4.3.2 Azure Key Vault

Quando trabalhamos com aplicações que serão colocadas em produção, é necessário armazenar as strings de conexão e segredos de maneira segura, pois essas informações não podem ficar visíveis.

O Azure Key Vault atua como um repositório seguro e centralizado para o armazenamento de segredos. Ele permite que aplicações e serviços na nuvem acessem esses recursos de forma controlada, utilizando políticas de acesso, autenticação baseada em identidade gerenciada (*Managed Identity*) e registro de auditoria. Além disso, o Key Vault elimina a necessidade de embutir credenciais diretamente na aplicação, reduzindo a superfície de ataque e garantindo maior conformidade com práticas de segurança corporativas.

No contexto desta arquitetura, o Key Vault é utilizado para armazenar strings de conexões sensíveis, como chaves de acesso ao Azure Data Lake Storage, ao Synapse SQL e demais segredos necessários ao funcionamento dos pipelines.

4.3.3 Azure Lake Storage (ADLS)

O Azure Data Lake Storage (ADLS) é um serviço de armazenamento otimizado para cargas analíticas e dados em larga escala. Trata-se de uma extensão do Azure Blob Storage, que adiciona funcionalidades voltadas a cenários analíticos em larga escala, como suporte nativo a hierarquias de diretórios, segurança granular e integração com ferramentas como Azure Databricks e Azure Synapse Analytics.

No trabalho proposto, o ADLS é utilizado como repositório central para as camadas Bronze, Silver e Gold da arquitetura *Medallion*. A camada Bronze armazena os dados brutos fornecidos pelas APIs. A camada Silver contém os dados limpos e padronizados, já a camada Gold disponibiliza os dados prontos para análise. O uso dessa solução possibilita armazenar grandes quantidades de dados com acesso controlado, sendo essencial para garantir a governança e a rastreabilidade.

4.3.4 Azure Data Factory (ADF)

O Azure Data Factory é um serviço de orquestração que permite criar pipelines de ingestão e movimentação de informações de forma automatizada. Ele oferece conectores nativos para diversas fontes de dados, como APIs, bancos de dados e arquivos na nuvem. Além da ampla conectividade com diversas fontes, o ADF também se destaca pela capacidade de agendar execuções, criar triggers e realizar tarefas em paralelo.

No contexto deste projeto, o ADF atua como orquestrador do pipeline de dados, responsável por coordenar e automatizar as etapas de processamento de forma periódica e controlada. Por meio dele, é possível agendar execuções, definir dependências entre atividades e acionar serviços complementares, como ingestão de dados, Notebooks Spark e outros componentes da arquitetura.

4.3.5 Azure Synapse

O Azure Synapse é uma plataforma totalmente gerenciada e integrada ao ecossistema Azure, incorporando o Apache Spark como motor de processamento distribuído. No Synapse, é possível criar notebooks interativos conectados a diferentes fontes de dados e executá-los utilizando pools de processamento configuráveis, como SQL e Spark.

De maneira geral, o Azure Synapse fornece um espaço colaborativo onde engenheiros e analistas de dados podem criar, testar e executar notebooks com suporte a várias

linguagens, como Python, SQL e Scala. Além disso, o ambiente possibilita a criação de clusters Spark sob demanda, otimizados para alto desempenho e gerenciados.

Na aplicação deste estudo de caso, o Azure Synapse será utilizado para a execução dos notebooks de transformação de dados entre as camadas Bronze, Silver e Gold. Sua integração nativa com o ADLS viabiliza a leitura e a escrita eficientes dos dados diretamente nas camadas organizadas pela arquitetura *Medallion*.

Parte III

Método de Pesquisa

5 Metodologia

Este trabalho caracteriza-se como uma pesquisa aplicada, de abordagem quantitativa e método de estudo de caso. O estudo de caso foi adotado conforme proposto por Yin (YIN, 2015), por ser apropriado quando se busca avaliar o comportamento de implementações em contextos reais e controlados.

Segundo Gil (GIL, 2008), a pesquisa aplicada visa produzir conhecimento voltado à solução de problemas práticos. Já a abordagem quantitativa é adequada quando se busca mensurar e analisar, por meio de métricas objetivas, o desempenho e os resultados de uma solução, especialmente em ambientes tecnológicos.

A abordagem quantitativa justifica-se pelo interesse em mensurar objetivamente o desempenho de uma plataforma de dados moderna implementada na nuvem Microsoft Azure, considerando métricas como latência, tempo de execução, capacidade de escalabilidade e custos operacionais associados ao processamento.

Para garantir controle experimental e consistência entre os cenários analisados, a coleta de dados baseou-se em conjuntos de dados sintéticos de IoT gerados especificamente para este estudo, permitindo variar a volumetria (10 GB, 40 GB e 100 GB) e observar o comportamento da arquitetura sob diferentes cargas de trabalho.

A metodologia desta pesquisa aplicada foi estruturada nas seguintes etapas:

1. **Levantamento bibliográfico**
   Nesta fase, foram coletados conceitos, modelos arquiteturais e boas práticas relacionadas à engenharia de dados moderna, com ênfase em arquiteturas de *Data Lakehouse*, processamento distribuído com Apache Spark e organização de dados segundo o modelo *Medallion*.

2. **Arquitetura e definição de ferramentas**
   A solução proposta foi planejada a partir da arquitetura *Medallion*, utilizando exclusivamente serviços da nuvem Microsoft Azure. Foram definidos Azure Data Lake Storage como repositório central, Azure Data Factory para orquestração dos pipelines em *Batch*, Azure Synapse Analytics com Spark Pool para processamento distribuído e Terraform como ferramenta de provisionamento da infraestrutura (IaC).

3. **Implementação da plataforma**
   Com a arquitetura definida, procedeu-se à construção do ambiente. Os recursos foram criados via Terraform, garantindo reprodutibilidade e padronização. Em seguida, configuraram-se os pipelines no Azure Data Factory, organizaram-se as cama-

das Bronze, Silver e Gold no *Data Lake* e desenvolveram-se os notebooks PySpark responsáveis pelas transformações e agregações necessárias nas etapas Silver e Gold.

4. **Execução de testes com métricas de desempenho**
   Nesta etapa, a plataforma foi submetida a experimentos controlados utilizando dados sintéticos de IoT gerados especificamente para o estudo. Três cenários foram avaliados: 10 GB, 40 GB e 100 GB. As métricas analisadas concentraram-se em tempo de execução, latência, escalabilidade do pipeline frente ao aumento de volumetria e custos operacionais associados ao processamento. As execuções foram repetidas para obtenção de médias consistentes e comparação entre os cenários.

5. **Avaliação e considerações**
   Por fim, realizou-se uma análise crítica dos resultados obtidos, observando o comportamento da arquitetura diante das diferentes volumetrias, sua eficiência em termos de execução e custo, bem como os impactos das decisões de arquitetura adotadas. A avaliação considerou os seguintes critérios:

   - **Clareza e robustez da arquitetura implementada**
   - **Conformidade com boas práticas de engenharia de dados em nuvem**
   - **Comportamento do processamento distribuído frente ao aumento de escala**
   - **Eficiência operacional e custo-benefício da solução**

A Figura 9 apresenta uma visão geral da arquitetura proposta, evidenciando as etapas de ingestão, armazenamento, processamento e consumo dos dados ao longo do pipeline estruturado segundo o modelo proposto.

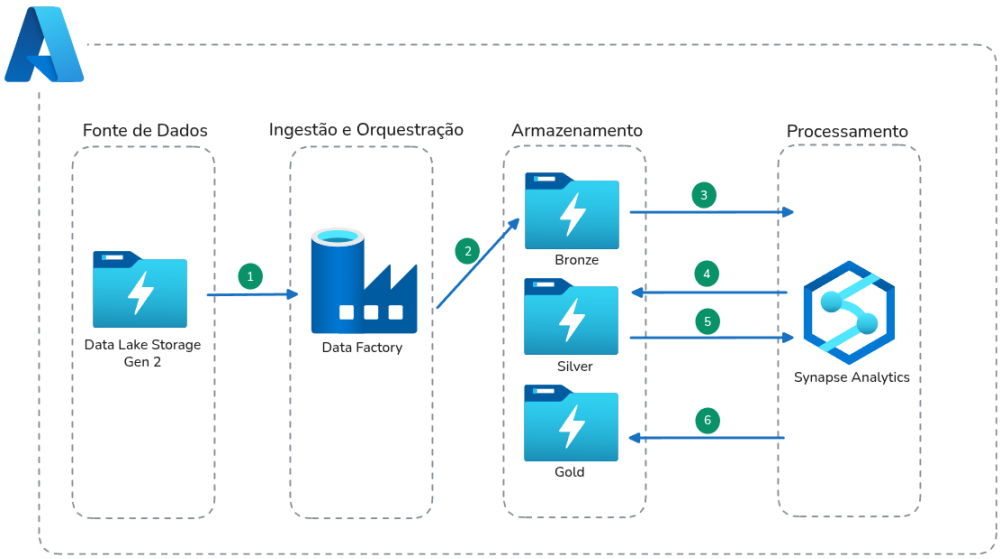

Figura 9 – Arquitetura geral da plataforma de dados proposta. A solução contempla a ingestão, armazenamento em camadas (*Medallion*) e Processamento com Spark

6 Configuração dos Experimentos

Este capítulo apresenta as configurações adotadas para a execução dos experimentos, detalhando os elementos essenciais que compõem o ambiente analítico utilizado. Além de descrever a infraestrutura em nuvem, são discutidos os mecanismos de geração dos dados sintéticos, a organização das camadas do *Data Lake* e as definições que orientaram o funcionamento dos pipelines de processamento.

O objetivo é estabelecer de forma transparente as condições sob as quais os testes foram conduzidos, permitindo compreensão precisa das características do ambiente e garantindo reprodutibilidade dos resultados. Para assegurar clareza e organização, a descrição foi estruturada em três partes principais, conforme detalhado abaixo:

- **Provisionamento da infraestrutura**: Detalha a criação automatizada dos recursos na Azure por meio do Terraform, contemplando a conta de armazenamento, o Azure Synapse Analytics, o cluster Spark utilizado nos experimentos e o Azure Data Factory responsável pela orquestração.
- **Criação dos dados sintéticos**: Descreve o processo de geração dos conjuntos de dados utilizados nos testes, incluindo a estratégia de calibração, definição de schema e controle de volumetria (10 GB, 40 GB e 100 GB).
- **Configuração dos recursos provisionados**: Abrange a implementação dos pipelines no Azure Data Factory, o desenvolvimento dos notebooks PySpark responsáveis pelo processamento das camadas Silver e Gold, e a configuração das permissões entre os serviços por meio de identidades gerenciadas. Essa etapa garante que o Azure Data Factory, o Azure Synapse e o Azure Data Lake Storage operem de forma integrada e com acesso seguro aos dados.

6.1 Provisionamento da Infraestrutura

Esta seção descreve como os recursos necessários para a execução dos experimentos foram provisionados na plataforma Microsoft Azure. Todo o ambiente foi criado por meio do Terraform, uma ferramenta de *Infrastructure as Code* (IaC) que permite definir, versionar e reproduzir recursos de forma declarativa.

6.1.1 Motivação da Adoção de IaC no Provisionamento de Recursos

A opção por utilizar *Infrastructure as Code* (IaC) no provisionamento dos recursos da plataforma fundamenta-se na necessidade de garantir reprodutibilidade, consistência e

automação ao ambiente experimental.

Como evidenciado em (LEKKALA, 2022), IaC representa uma mudança significativa em comparação aos métodos tradicionais baseados em configurações manuais, os quais tendem a ser mais lentos, propensos a erros e difíceis de escalar. Ao substituir procedimentos manuais por infraestrutura descrita em código, permite-se criar ambientes padronizados, auditáveis e facilmente replicáveis.

Assim, a adoção de Terraform não apenas simplifica o provisionamento, mas também aumenta a confiabilidade dos experimentos e reforça as boas práticas de engenharia de dados e computação em nuvem.

6.1.2 Execução do Provisionamento com Terraform

O provisionamento da infraestrutura foi realizado localmente por meio da ferramenta *Terraform*, utilizando o provedor `AzureRM` para interação com a plataforma Microsoft Azure.

As definições da infraestrutura foram organizadas em arquivos `.tf` e em um arquivo de variáveis, `variables.tf`, no qual foram parametrizados elementos como nomes lógicos de recursos, sufixos de hash, tamanho do pool do Spark e tags padrão do ambiente.

Antes da execução, foi realizada a autenticação na Azure por meio do comando `az login`, permitindo que o Terraform utilizasse o ambiente em nuvem autenticado para realizar ações.

Em seguida, o comando `Terraform init` foi utilizado para inicializar o diretório do trabalho, baixar o provedor `AzureRM` e preparar o ambiente para o provisionamento. A etapa seguinte consistiu na execução do comando `Terraform plan`, que gerou uma prévia das ações que seriam realizadas, possibilitando validar as alterações de recursos definidas.

Por fim, o ambiente completo foi criado por meio do comando `Terraform apply`, que materializou todos os recursos especificados nos arquivos `.tf`. Visando armazenar e versionar essa estrutura criada, foi realizada uma publicação do código final em um repositório do Github.

6.1.3 Recursos Provisionados

Todos os recursos foram provisionados na região **Brazil South**. Essa escolha foi motivada pela proximidade geográfica em relação ao meu local de utilização, garantindo menor latência nas comunicações e maior desempenho geral da infraestrutura.

Os recursos provisionados, assim como suas respectivas funções, são:

- **Resource Group**

  - **Nome**: `iotsyntheticdata202511`
  - **Função**: Agrupar logicamente todos os recursos da plataforma

- **Azure Data Factory**
  - **Nome**: `iot-synthetic-data-factory-202511-{hash}`
  - **Configurações**:
    * Identidade gerenciada: *SystemAssigned*
  - **Função**: Orquestração dos pipelines de ingestão e execução dos notebooks

- **Azure Data Lake Storage Gen2**
  - **Nome**: `iotsyntheticdatadl{hash}`
  - **Configurações**:
    * Tipo: Standard
    * Redundância: Local (LRS)
    * Hierarchical Namespace habilitado
    * Containers:
      · `source` — dados de origem
      · `bronze` — dados ingeridos sem modificação
      · `silver` — dados limpos e processados
      · `gold` — dados consolidados para consumo
      · `mysynapse` — diretório operacional do Synapse
  - **Função**: Armazenamento central da arquitetura *Medallion*

- **Azure Synapse Analytics**
  - **Nome**: `iot-synthetic-synapse-{hash}`
  - **Configurações**:
    * Identidade gerenciada: *SystemAssigned*
    * Acesso público habilitado
  - **Função**: Execução dos notebooks Spark e das consultas SQL

- **Spark Pool**
  - **Nome**: `SparkPool`
  - **Configurações**:
    * Tipo: *Memory Optimized*
    * Versão do Spark: 3.4

    - Tamanho dos nós: Small (4VCores e 32GB de RAM)
    - Número de nós: 3 (auto-scaling fixo 3–3)
    - Auto-pause: 15 minutos
  - **Função**: Processamento das transformações nas camadas Silver e Gold

Além dos recursos principais, foram aplicadas tags padrão a todos os serviços provisionados, incluindo `Environment=dev` e `Project=iotsyntheticdata`. Essas marcações têm papel importante na organização e governança do ambiente, facilitando a identificação dos recursos e permitindo um melhor controle administrativo, especialmente em cenários de múltiplos projetos ou equipes.

Outro aspecto relevante do provisionamento diz respeito à estratégia de nomes adotada. Para garantir nomes únicos globalmente, requisito para serviços como Storage Accounts, Data Factory e Synapse Workspace, foram utilizados sufixos baseados em hashes determinísticos derivados do identificador do Resource Group. Essa abordagem assegura previsibilidade, evita colisões de nomes e mantém um padrão uniforme em todos os recursos criados via Terraform.

6.2 Configuração dos recursos provisionados

Após realizar o provisionamento da infraestrutura, foi necessário realizar configurações pontuais nos recursos, assim como criar os notebooks no Synapse e os pipelines no Data Factory. Para facilitar o entendimento desta etapa, iremos dividi-la em três subseções:

1. **Configurações de Permissões e Acessos**
2. **Desenvolvimento dos Notebooks de Processamento (Azure Synapse)**
3. **Desenvolvimento dos Pipelines de Orquestração (Azure Data Factory)**

6.2.1 Configurações de Permissões e Acessos

Após o provisionamento da infraestrutura, foi necessário configurar os mecanismos de acesso entre os serviços envolvidos no pipeline - Azure Synapse, Azure Data Factory e Azure Data Lake Storage. Essa etapa assegura que os componentes da plataforma possam operar de forma integrada, respeitando o princípio do privilégio mínimo e garantindo comunicação segura entre os serviços.

Como a solução não utiliza redes privadas, o acesso público foi mantido como habilitado para o Synapse Workspace e para o Azure Data Factory. Entretanto, esse

acesso foi restrito por meio da restrição de IPs permitidos, garantindo que somente origens autorizadas pudessem acessar os recursos e serviços expostos. Essa abordagem reduz significativamente a superfície de ataque, e mantém os acessos necessários habilitados.

Em seguida, foram configuradas as permissões do ADLS por meio do *Acess Control*. As identidades gerenciadas do Synapse e do Data Factory receberam os papéis de *Storage Blob Data Contributor*, permitindo leitura e escrita nas camadas `source`, `bronze`, `silver` e `gold`, assim como do diretório operacional utilizado pelo workspace do Synapse. Essas configurações são necessárias para os notebooks executarem transformações, criem arquivos, Delta, apliquem reparticionamentos e gerenciem diretórios intermediários.

Essa segmentação de permissões foi projetada para garantir:

- que cada serviço tenha somente o acesso estritamente necessário para suas operações;
- que a comunicação entre ADF, Synapse e ADLS ocorra exclusivamente por identidades gerenciadas, eliminando chaves e segredos;
- que a plataforma implemente boas práticas de segurança, reduzindo riscos de exposição acidental;
- que o fluxo operacional da arquitetura *Medallion* seja preservado, permitindo ingestão, transformação e consolidação de forma contínua.

Com essa configuração de permissões e acessos, os recursos passam a operar de maneira coordenada e segura, estabelecendo a base necessária para a execução dos experimentos conduzidos neste estudo.

6.2.2 Desenvolvimento dos Notebooks de Processamento

Nesta etapa, as transformações são desenvolvidas em notebooks Jupyter, executados no ambiente do Synapse Analytics, é utilizado o PySpark na escrita dos códigos. Os notebooks criados foram associados ao Spark Pool criado na etapa de provisionamento de recursos, de modo que sua execução ocorra utilizando o cluster já provisionado. Foram criados dois notebooks distintos, um para cada etapa da arquitetura *Medallion*:

1. Notebook (Bronze → Silver) – responsável pela ingestão, limpeza inicial, enriquecimento e aplicação de regras de qualidade de dados.
2. Notebook (Silver → Gold) – responsável pela agregação analítica e geração de métricas consolidadas para consumo analítico.

Ambos os notebooks seguem princípios fundamentais em engenharia de dados: modularidade, particionamento eficiente e evolução de schema controlada.

6.2.2.1 Notebook (Bronze → Silver)

Este notebook pretende transformar os dados brutos armazenados em arquivos Parquet na camada Bronze em uma tabela analítica, versionada e confiável na camada Silver, em formato Delta Lake.

As principais etapas implementadas foram:

- Inicialização da `SparkSession` com suporte nativo ao Delta Lake e configurações otimizadas, como `shuffle.partitions = 128` e execução adaptativa ativada (AQE).
- Definição explícita do schema de leitura dos arquivos Parquet da camada Bronze, evitando inferência automática.
- Leitura unificada de todos os arquivos Parquet (`*.parquet`) com opção `mergeSchema=true` para suportar evolução de schema.
- Derivação de colunas temporais: `event_date`, `event_year`, `event_month` e `event_hour` a partir do campo `event_ts`.
- Aplicação de regras de qualidade de dados:
  - `dq_temp_out_of_range`: temperatura $< -50\,^{\circ}\mathrm{C}$ ou $> 100\,^{\circ}\mathrm{C}$
  - `dq_pressure_out_of_range`: pressão ≤ 0
  - `dq_battery_out_of_range`: bateria $< 0\%$ ou $> 100\%$
- Normalização do campo `tags` em array de strings únicas (`array_distinct(split(...))`).
- Extração de metadados do campo `payload` via `regexp_extract`:
  - `fw_version`
  - `meta_location_group`
  - `meta_device_group`
- Escrita na camada Silver em formato Delta Lake:
  - Modo `overwrite`
  - Particionamento físico por `event_year` e `event_month`
  - Opção `overwriteSchema=true`

6.2.2.2 Notebook (Silver → Gold)

Este notebook pretende consolidar os dados já tratados na camada Silver e transformá-los em uma estrutura analítica unificada na camada Gold. Nessa fase, os registros são agregados, enriquecidos e reorganizados para compor uma *One Big Table* (OBT) em formato Delta Lake, preparada para consultas analíticas de alto desempenho.

- Inicialização da `SparkSession` com suporte ao Delta Lake e mesmas otimizações do notebook anterior.
- Leitura resiliente da tabela Silver (Delta) com reconstrução automática de colunas de data caso ausentes.
- Definição de criticidade do evento (`is_critical`):
  - `status_code` $\neq$ ``OK'' ou
  - presença da tag `critical`
- Conversão das flags booleanas de *Data Quality* em inteiros para agregação por soma.
- Agregação diária por `event_date`, `device_id`, `location_id` e grupos de metadados, gerando:
  - `events_total`, `events_critical`, `pct_critical_events`
  - Temperatura: média, máxima e mínima
  - Pressão média, consumo médio de energia, nível médio de bateria
  - Contagem absoluta de violações *Data Quality* por tipo de sensor
- Escrita na camada Gold em formato Delta Lake:
  - Modo `overwrite`
  - Particionamento por `event_year` e `event_month`
  - Opção `overwriteSchema=true`

6.2.3 Desenvolvimento dos Pipelines de Orquestração

Os pipelines realizam a orquestração dos fluxos de ingestão e de processamento, sendo responsáveis por realizar a ativação de cada componente no momento correto. A criação dos pipelines é realizada no Azure Data Factory, e permite que todo o fluxo de dados ocorre de forma estruturada, controlada e reprodutível, além de possuir as integrações nativas com os recursos provisionados no ambiente.

6.2.4 Desenvolvimento dos Pipelines de Orquestração

A orquestração do fluxo de processamento foi implementada no Azure Data Factory (ADF), que desempenha o papel de coordenar a execução sequencial dos notebooks responsáveis pelo tratamento dos dados nas camadas Bronze, Silver e Gold. O uso do ADF garante que o pipeline opere de forma estruturada, controlada e reprodutível, mantendo a coerência entre etapas e permitindo que o processo seja monitorado e executado repetidamente durante os experimentos.

Os pipelines foram desenvolvidos utilizando atividades do tipo Synapse Notebook, que permitem acionar diretamente os notebooks hospedados no Azure Synapse utilizando a identidade gerenciada do próprio Data Factory. Essa abordagem elimina a necessidade de chaves ou credenciais explícitas, tornando a comunicação entre os serviços mais segura e alinhada às boas práticas de autenticação baseadas em identidade.

A estrutura do pipeline segue a lógica da arquitetura *Medallion*, composta pelas seguintes etapas:

- **Ingestão de Dados (Bronze)**
  A primeira etapa do pipeline consiste em uma atividade *Copy Data*, responsável por copiar os arquivos sintéticos da camada `source` para a camada `bronze`. Essa cópia é realizada com a opção *preserve hierarchy* habilitada, garantindo que a estrutura original de diretórios e arquivos seja mantida durante a movimentação.

- **Transformações e padronização (Silver)**
  A segunda etapa consiste na execução de um notebook PySpark que realiza o tratamento dos dados brutos. Nessa fase, são aplicadas correções de tipos, conversões explícitas de colunas, normalização de valores, remoção de registros inválidos e criação de atributos derivados.

- **Consolidação e geração da OBT (Gold)**
  A etapa final executa um notebook PySpark dedicado à consolidação analítica dos dados. Aqui, são realizadas agregações, cálculos estatísticos por dispositivo, métricas temporais, indicadores de falhas e sumarizações operacionais.

Para garantir robustez e rastreabilidade, os pipelines incluem tratamento de erros, logs de execução e dependências explícitas entre as atividades. Dessa forma, a transição entre notebooks ocorre somente após a conclusão bem-sucedida da etapa anterior, evitando inconsistências no fluxo de dados. Além disso, a estrutura modular facilita a reexecução parcial do pipeline, permitindo reiniciar somente as etapas necessárias em caso de falhas.

Essa abordagem de orquestração proporciona um fluxo de processamento previsível e controlado, assegurando que todas as execuções realizadas nos experimentos sigam a

mesma sequência de operações e condições de processamento. Com os pipelines implementados, a plataforma passa a operar de forma integrada, alinhando ingestão, transformação e consolidação em um único fluxo automatizado.

6.3 Criação dos Dados Sintéticos

Para realizar os experimentos descritos neste trabalho, foram sintetizados dados de IoT em diferentes volumetrias (10 GB, 40 GB e 100 GB). A escolha de utilizar dados sintéticos permite controlar rigorosamente a volumetria, schema e a distribuição dos atributos.

Com o controle realizado nos dados, é possível isolar a arquitetura e garantir que o comportamento observado decorra unicamente das características da solução implementada. A partir da execução com os dados gerados, será possível inferir o desempenho, latência, escalabilidade e custos da plataforma de dados implementada. Além disso, a geração artificial permite simular cenários comuns em ambientes de IoT sem incorrer em limitações de acesso, privacidade ou disponibilidade.

6.3.1 Ambiente de Execução

A geração dos dados foi realizada por meio de um notebook hospedado no Azure Synapse, utilizando o Spark Pool provisionado para o projeto. O cluster empregado possui três nós do tipo Small, cada um com 4vCores e 32GB de memória RAM. A escolha de um cluster com essa configuração foi motivada pelo equilíbrio entre custo operacional e capacidade de processamento distribuído necessária para a execução estável do processo de geração.

Foi habilitado o *Adaptive Query Execution* (AQE) para permitir que o Spark ajuste dinamicamente o plano de execução conforme as características observadas dos dados, melhorando a eficiência das operações de shuffle.

6.3.2 Esquema e Características dos Dados

O processo de geração fundamenta-se na função `generate`, que produz um *DataFrame* compatível com o schema adotado no pipeline de IoT da plataforma. Cada linha representa um evento gerado por um dispositivo, contendo atributos identificadores, medidas numéricas, marcadores de tempo e metadados operacionais. A Tabela 4 apresenta o conjunto de atributos sintetizados.

Tabela 4 – Atributos dos dados sintéticos gerados para os experimentos

Atributo	Tipo	Descrição
`event_id`	Long	Identificador sequencial único do evento.
`device_id`	String	Identificador do dispositivo, gerado via `uuid()`.
`device_type`	String	Categoria do dispositivo (`type-A`, `type-B`, `type-C`).
`location_id`	String	Identificador numérico da localização.
`event_ts`	Timestamp	Marcador de tempo aleatório em um intervalo de três anos.
`temperature`	Double	Temperatura simulada com distribuição normal.
`pressure`	Double	Pressão atmosférica simulada com distribuição normal.
`energy_consumption`	Double	Consumo energético do dispositivo.
`battery_level`	Double	Percentual de bateria (0 a 100).
`status_code`	String	Estado operacional (`OK` ou `ERROR`).
`tags`	String	Classificação textual do evento.
`payload`	String	Metadados codificados (firmware, grupos lógicos).

Os valores são gerados a partir de funções aleatórias uniformes e normais, garantindo variabilidade estatística e simulando condições realistas de comportamento de dispositivos IoT.

6.3.3 Calibração da Volumetria

A calibração da volumetria ocorreu para garantir que os conjuntos de dados sintéticos fossem gerados com precisão e de forma compatível com as limitações de memória do cluster Spark utilizado nos experimentos. Como o ambiente de execução dispõe de recursos reduzidos, a geração direta de grandes volumes poderia resultar em erros de estouro de memória. Para mitigar esse risco, o processo foi estruturado em duas fases: calibração inicial e geração incremental dos volumes finais.

Na fase de calibração, o notebook Spark gera uma amostra de 1.000.000 de linhas com o mesmo schema adotado na geração dos dados finais. Essa amostra é gravada em um único arquivo Parquet em um diretório temporário no *Data Lake*. A partir da consulta do tamanho total desse arquivo criado, é calculado o tamanho médio por linha, permitindo estimar de forma empírica quantas linhas são necessárias para compor aproximadamente 1 GB de dados.

Além de estimar a quantidade de linhas por gigabyte, a calibração também determina o número adequado de partições para os arquivos Parquet ficarem próximos a 128

MB, tamanho que equilibra paralelismo e eficiência de leitura. Essa definição é importante pois evita a criação de arquivos pequenos ou grandes demais, contribuindo para o bom desempenho do pipeline de processamento nas etapas Silver e Gold.

Com essas informações iniciais, o sistema calcula quantos ciclos de geração de dados são necessários para atingir cada volume alvo (10 GB, 40 GB, 100 GB). Esse mecanismo garante precisão nas volumetrias desejadas, ao ser calculado com exatidão o número de linhas que devem compor cada chunk.

Por fim, o diretório temporário utilizado é removido, e os parâmetros de tamanho médio por linha, linhas necessárias por gigabyte e número ideal de partições são passados para o gerador de dados. Com o recebimento dos parâmetros previamente calculados, o gerador de dados sintéticos realiza a produção dos dados resultando em conjuntos de dados compatíveis com o objetivo experimental desse estudo.

6.3.4 Geração Incremental dos Dados

Após a calibração da volumetria, iniciou-se a etapa de geração dos conjuntos finais de dados sintéticos, estruturada incrementalmente para garantir estabilidade operacional no cluster Spark. Ao invés de produzir todo o volume de uma só vez, o que poderia resultar em uso excessivo de memória, o processo foi dividido em ciclos sucessivos, cada um responsável pela criação de aproximadamente 1 GB de dados.

Para cada volumetria definida (10 GB, 40 GB e 100 GB), o sistema executa um número de iterações proporcional ao valor alvo. Em cada iteração, a função `generate` cria um conjunto de linhas com tamanho previamente calculado na etapa de calibração, assegurando que cada chunk corresponda a aproximadamente 1 GB. Após a geração, o DataFrame é reparticionado conforme o número ideal de partições estimado, garantindo que os arquivos Parquet obtidos permaneçam próximos a 128 MB, contribuindo para a eficiência do processamento.

Os chunks são gravados sequencialmente no *Data Lake*, no modo *append*, permitindo que o volume desejado seja alcançado gradualmente, sem sobrecarregar a memória do cluster. Como cada chunk é independente e utiliza reparticionamento explícito antes da escrita, o processo mantém um comportamento previsível e aproveita o paralelismo do cluster, ainda que sua memória seja limitada.

Ao final da geração de cada volumetria, o diretório correspondente contém todos os arquivos Parquet produzidos pelos ciclos, representando o volume total especificado. Essa estratégia de produção incremental permite que conjuntos de dados de até 100 GB sejam gerados de forma segura e reprodutível.

6.3.5 Resumo Quantitativo dos Dados

Após a geração incremental dos dados, foram coletadas informações sobre os conjuntos de dados produzidos. Para cada volumetria gerada (10 GB, 40 GB, 100 GB), foram observados o número total de eventos, a quantidade de arquivos Parquet resultantes, o tamanho médio dos arquivos e o tamanho efetivo armazenado no *Data Lake*. A tabela 5 demonstra as observações realizadas a partir dos dados gerados sinteticamente.

Tabela 5 – Resumo quantitativo dos dados sintéticos gerados

Tamanho Alvo	Eventos gerados	Arquivos Parquet	Tamanho Real
10 GB	126,414,490	80	10,10 GB
40 GB	505,657,960	320	40,43 GB
100 GB	1,264,144,900	800	101,09 GB

6.4 Plano Experimental: Fatores, Níveis e Métricas

Para conduzir os experimentos deste estudo de forma sistemática e controlada, foi necessário definir claramente os fatores manipulados e os níveis associados a cada um deles. A identificação desses elementos permite estruturar o desenho experimental, garantindo que os resultados obtidos reflitam as variações introduzidas intencionalmente e possibilite análises comparativas consistentes entre os cenários avaliados.

6.4.1 Fator 1: Volumetria dos Dados

O principal fator experimental considerado neste trabalho é a volumetria dos dados processados pela plataforma. Esse fator foi selecionado por representar diretamente a carga de trabalho aplicada ao pipeline e por influenciar de maneira significativa o comportamento de sistemas baseados em processamento distribuído. Os níveis definidos para este fator são:

- **10 GB**: Cenário de menor carga.
- **40 GB**: Cenário de carga intermediária.
- **100 GB**: Cenário de maior carga.

6.4.2 Fator 2: Etapas do *Pipeline* de Processamento

O segundo fator relevante para a análise diz respeito às etapas do pipeline executadas no Azure Synapse. Como cada fase do fluxo realiza operações distintas, desde movimentação e preparação dos dados até agregações complexas, é esperado que apresentem comportamentos diferentes em termos de desempenho. Os níveis considerados foram:

- **Ingestão**: Etapa de ingestão de dados da fonte e carregamento na camada bronze
- **Transformação da camada Bronze -> Silver**: Etapa de limpeza, normalização e padronização do schema.
- **Transformação da camada Silver -> Gold**: etapa de agregação, enriquecimento e consolidação analítica.

A escolha desses fatores se deu com base nos objetivos do experimento: entender como o volume de dados impacta o desempenho da arquitetura e avaliar a latência nas diferentes fases do pipeline.

Cada combinação entre os níveis dos fatores foi tratada como uma condição experimental, permitindo a observação cruzada do comportamento da plataforma.

6.4.3 Métricas

As métricas representam os indicadores quantitativos utilizados para avaliar o desempenho da plataforma em cada combinação de fatores e níveis definidos. Elas permitem mensurar o comportamento do pipeline sob diferentes cargas de trabalho e fornecem informações relevantes para a análise comparativa entre os cenários experimentais.

Para esse experimento, são consideradas as seguintes métricas:

6.4.3.1 Tempo de execução

Corresponde ao tempo total de cada atividade do pipeline, desde o disparo pelo Azure Data Factory até a conclusão da operação. Na ingestão, esse tempo representa a duração completa da *Copy Activity*. Já nas transformações (Bronze $\rightarrow$ Silver e Silver $\rightarrow$ Gold), corresponde ao tempo total de execução do notebook no *Spark Pool*.

6.4.3.2 Latência de Orquestração

Refere-se ao intervalo entre o acionamento da atividade pelo Azure Data Factory e o início efetivo do processamento associado. Assim, na ingestão, a latência é a diferença entre o tempo total e o tempo efetivo da *Copy Activity*; nas transformações, é a diferença entre o tempo total e a duração do job no Spark.

6.4.3.3 Custo operacional

O custo operacional do pipeline é composto por três elementos principais: processamento, orquestração e operações de armazenamento.

6.4.3.3.1 a) Custo de processamento (Spark Pool).

Todas as transformações foram executadas em um Spark Pool com preço de **US$ 2.30 por hora**, conforme as tarifas vigentes em **16 de novembro de 2025**. Assim, o custo de execução de cada notebook é dado por:

$$\text{Custo}_{\text{Spark}} = \text{Duração (h)} \times 2.30.$$

6.4.3.3.2 b) Custo de orquestração (Azure Data Factory).

O custo de orquestração apresenta dois comportamentos distintos ao longo dos cenários avaliados.

Nos cenários de ingestão (C1–C3), que utilizam exclusivamente a *Copy Activity*, adota-se o modelo tarifário oficial baseado em consumo de *DIU-hours*, expresso por:

$$\text{Custo}_{\text{ADF}} = 0.25 \times \text{DIU-hour} + 0.001.$$

Além do custo proporcional ao uso de DIUs, cada execução do pipeline gera um único *Activity Run*, cujo valor unitário é de aproximadamente 0.01 U$. Para representar esse componente tarifável, o custo final de cada execução inclui essa parcela fixa adicional.

A partir dos cenários C4–C9, o pipeline deixa de utilizar *Copy Activity* e opera somente com *External Activities*, responsáveis por acionar notebooks Spark. Embora essas atividades registrem tempos de execução variando entre 0.15 e 1.18 horas, o custo por hora associado a esse tipo de operação é extremamente reduzido, sempre inferior a 0.01 U$. Por esse motivo, o valor é tratado como desprezível e representado como um custo fixo de 0.01 U$ por execução, refletindo somente o custo mínimo do *Activity Run* associado ao acionamento externo.

6.4.3.3.3 c) Custo de operações do armazenamento (ADLS Gen2).

As etapas do pipeline envolvem diversas interações com o Azure Data Lake Storage Gen2 (ADLS), todas executadas na **Hot Tier**. Essas interações são tarifadas com base no número de operações realizadas, e não no volume de dados processado. As chamadas identificadas durante as execuções foram agrupadas em dois grandes conjuntos:

- **Operações de leitura (*Read operations*)**: incluem chamadas como `ReadFile`, `GetPathStatus`, `GetFileProperties`, `ListBlobs`, `GetBlobServiceProperties`, `GetContainerProperties`, entre outras operações de metadados associadas à consulta do estado ou estrutura dos arquivos.

- **Operações de escrita (*Write operations*)**: incluem `AppendFile`, `FlushFile`, `CreatePathFile`, `DeleteFile`, `CreatePathDirectory`, `RenamePath`, além de outras operações que modificam conteúdo ou estrutura do sistema de arquivos.

Esses dois grupos possuem tarifas distintas na **Hot Tier**. De forma geral, operações de escrita são mais caras que operações de leitura, devido ao custo adicional associado à garantia de durabilidade e consistência do armazenamento distribuído.

Assim, o custo total decorrente das operações no ADLS é calculado por:

$$\text{Custo}_{ADLS} = \left(\sum_{i \in Read} \text{Chamadas}_i \times \text{Preço}_i \right) + \left(\sum_{j \in Write} \text{Chamadas}_j \times \text{Preço}_j \right).$$

Cada execução do pipeline gerou milhares de operações em ambos os grupos, variando conforme a volumetria (10 GB, 40 GB e 100 GB) e conforme a etapa (Ingestão, Bronze → Silver e Silver → Gold). Esses registros foram extraídos diretamente do Azure Insights, permitindo estimar o custo transacional real associado a cada cenário experimental.

6.4.3.3.4 Considerações sobre o custo mensal de armazenamento.

O ADLS também possui tarifas de armazenamento mensais (GB/mês), porém esse custo **não foi incluído** na análise. O objetivo do estudo é isolar o custo operacional das execuções, e não o custo de retenção de dados. Além disso, o custo mensal é constante e não interfere na comparação entre cenários, volumetrias e etapas de processamento.

6.4.3.4 Escalabilidade

A escalabilidade é avaliada com base na variação proporcional do tempo de execução e do custo total à medida que a volumetria cresce entre os níveis de 10 GB, 40 GB e 100 GB. Essa análise permite identificar padrões de desempenho, gargalos e a capacidade do pipeline de lidar com cargas crescentes de forma eficiente.

6.5 Cenários do Experimento

A partir da combinação desses fatores, foram definidos nove cenários experimentais, descritos a seguir:

- **C1**: Ingestão (Source → Bronze) com volumetria de 10 GB.
- **C2**: Ingestão (Source → Bronze) com volumetria de 40 GB.

- **C3**: Ingestão (Source → Bronze) com volumetria de 100 GB.
- **C4**: Transformação (Bronze → Silver) com volumetria de 10 GB.
- **C5**: Transformação (Bronze → Silver) com volumetria de 40 GB.
- **C6**: Transformação (Bronze → Silver) com volumetria de 100 GB.
- **C7**: Transformação (Silver → Gold) com volumetria de 10 GB.
- **C8**: Transformação (Silver → Gold) com volumetria de 40 GB.
- **C9**: Transformação (Silver → Gold) com volumetria de 100 GB.

6.6 Justificativa da Abordagem Experimental

Esta seção pretende apresentar os fundamentos utilizados para a definição dos fatores, níveis, métricas e cenários experimentais adotados neste estudo. As escolhas aqui descritas asseguram que a avaliação conduzida seja coerente com a literatura e reflita adequadamente os objetivos de mensurar desempenho, escalabilidade e eficiência da arquitetura proposta.

6.6.1 Fatores

6.6.1.1 Volumetria

Visando mensurar o desempenho da arquitetura implementada, este trabalho adota a volumetria de dados como um dos fatores analisados. Em plataformas de dados, a capacidade de lidar com diferentes volumetrias de dados é um elemento determinante para avaliar a escalabilidade, eficiência operacional e previsibilidade de custo.

Desse modo, a avaliação baseada em volumetria permite observar, de maneira prática, como a infraestrutura se comporta sob cargas crescentes, desde cenários de volumes menores até a volumes mais expressivos.

A relevância desse fator também se sustenta em trabalhos relacionados a processamento distribuído de dados. Estudos que analisam arquiteturas distribuídas frequentemente utilizam faixas de volume como variável experimental, ao evidenciarem possíveis gargalos e limites do sistema. Essa abordagem é adotada, por exemplo, em trabalhos como (CHENG; ZHANG; YE, 2019), onde diferentes tamanhos de conjuntos de dados são utilizados para avaliar a capacidade de processamento e estabilidade da plataforma.

6.6.1.2 Etapas do Pipeline

As etapas do pipeline de dados foram tratadas como fatores, uma vez que cada uma representa um ponto diferente exigência no fluxo de dados. Trabalhos como (SSERUNJOGI et al., 2025) mostram que pipelines reais costumam separar o processamento em fases claras, isto é: ingestão, processamento e modelagem. Além disso, é mostrado como cada fase afeta distintamente o tempo de execução e o consumo de recursos.

6.6.2 Níveis

6.6.2.1 Volumetrias

Para estabelecer faixas representativas, recorreu-se à classificação proposta por (SCHMIDT et al., 2025), que apresenta o conceito de *Not-Small–Not-Big Data* (NoS–NoB Data). Segundo os autores, essa categoria abrange conjuntos de dados que excedem a capacidade de ferramentas tradicionais, mas que ainda não exigem infraestruturas massivas. Na literatura, esses volumes situam-se tipicamente entre dezenas e centenas de gigabytes, posicionando-se em um intervalo intermediário no espectro de dados.

Com base nessa abordagem, os níveis definidos para este estudo, 10 GB, 40 GB e 100 GB, representam faixas distintas dentro dessa zona intermediária. O volume de 10 GB corresponde ao limite inferior da categoria, onde ferramentas convencionais começam a apresentar limitações técnicas e operacionais. O nível de 40 GB situa-se no núcleo da faixa NoS–NoB, refletindo um cenário realista de cargas moderadas que já demandam tecnologias de processamento distribuído. Por fim, o nível de 100 GB aproxima-se do limite superior descrito pelos autores, caracterizando um cenário de carga elevada que permite avaliar a escalabilidade do pipeline em condições mais exigentes.

Assim, a escolha desses três níveis possibilita observar comportamentos distintos do pipeline à medida que a volumetria cresce, garantindo que os experimentos reflitam situações coerentes com o espectro de dados intermediários discutido na literatura e adequados ao objetivo de avaliar a capacidade de escalabilidade da arquitetura implementada.

Por fim, é importante ressaltar que não foram cogitadas faixas massivas de dados, como as discutidas por (GAO; HAN; WAN, 2020), que mencionam escalas operando na ordem de terabytes a exabytes em aplicações reais. Esses cenários, embora característicos de ambientes de *Big Data* em larga escala, são impraticáveis em infraestruturas com limites de recursos, como no caso deste estudo.

6.6.2.2 Etapas do Pipeline de Processamento

As etapas do pipeline também foram tratadas como níveis. Assim, cada experimento foi executado em três estágios distintos: ingestão → Bronze, Bronze → Silver e Silver → Gold.

Cada nível captura um tipo de demanda computacional diferente, permitindo observar efeitos de I/O, normalização, agregação e operações analíticas.

6.6.3 Métricas

As métricas adotadas neste estudo têm o objetivo de quantificar o desempenho, a responsividade, o custo e a escalabilidade do pipeline sob diferentes combinações de fatores e níveis experimentais. As definições a seguir refletem o comportamento real das execuções na plataforma Azure.

- **Tempo de execução**: corresponde ao tempo total de cada atividade do pipeline, incluindo tanto o período de orquestração quanto o período de processamento efetivo.
- **Latência de Orquestração**: representa o intervalo entre o disparo da atividade pelo Azure Data Factory e o início efetivo do processamento associado. Assim, a latência é obtida pela diferença entre o tempo total de execução e o tempo efetivamente dedicado à operação principal.
- **Custo operacional**: corresponde ao consumo efetivo dos recursos da plataforma, abrangendo orquestração, processamento e armazenamento envolvidos em cada etapa do pipeline.
- **Escalabilidade**: avaliada por meio da variação do tempo de execução à medida que a volumetria aumenta, permitindo identificar como o pipeline responde ao crescimento da carga em cada etapa (ingestão, Bronze $\rightarrow$ Silver e Silver $\rightarrow$ Gold).

6.6.4 Cenários

Os cenários experimentais foram definidos pela combinação entre fatores e níveis. Consideram-se:

- Três volumetrias: 10 GB, 40 GB e 100 GB;
- Três etapas do pipeline: ingestão $\rightarrow$ Bronze, Bronze $\rightarrow$ Silver e Silver $\rightarrow$ Gold.

Para cada um dos nove cenários definidos, resultantes da combinação de fatores, foram realizadas três execuções independentes do pipeline. Essa repetição de execução visa obter valores mais consistentes, a partir das médias obtidas nas três execuções.

6.7 Organização dos experimentos

Os experimentos foram estruturados para refletir o funcionamento real de uma plataforma moderna de dados baseada na arquitetura *Medallion*. Cada execução corresponde a um estágio específico do pipeline e utiliza uma combinação controlada dos serviços envolvidos no fluxo operacional.

A seguir, descreve-se como cada conjunto de experimentos foi conduzido e quais serviços são utilizados em cada fase.

6.7.1 Etapa 1: Ingestão (Source → Bronze)

Nesta etapa, avalia-se o fluxo de ingestão dos dados brutos para o *Data Lake*.

A operação é composta por:

- *Azure Data Factory* (Orquestração)
- *Copy Data* (Componente de Movimentação de Dados)

Não há transformação ou padronização, o foco é medir o as métricas no processo de ingestão.

Foram executados três cenários: 10 GB, 40 GB e 100 GB.

6.7.2 Etapa 2: Transformação Bronze → Silver

Esta etapa mensura o impacto das operações de limpeza, normalização e *schema enforcement*.

Os serviços utilizados são:

- *Azure Data Factory* (Orquestração)
- Synapse Spark Notebook (Execução das transformações)

As operações incluem correção de tipos, eliminação de inconsistências e padronização do schema.

Novamente, três cenários foram avaliados (10 GB, 40 GB, 100 GB).

6.7.3 Etapa 3: Transformação Silver → Gold

Na terceira etapa, concentram-se as transformações responsáveis por disponibilizar os dados para uso analítico: agregações, joins, enriquecimentos e derivação de indicadores.

A composição segue a mesma lógica da etapa anterior:

- *Azure Data Factory* (Orquestração)
- Synapse Spark Notebook (Execução das transformações)

Também foram avaliados três cenários, cada um com uma volumetria diferente.

6.7.4 Execuções Repetidas e Tamanho de Recursos

Para assegurar maior consistência na comparação entre os cenários, cada experimento foi executado três vezes sob as mesmas condições operacionais. Essa abordagem contribui para resultados mais representativos, reduzindo flutuações momentâneas do ambiente de nuvem e permitindo uma análise mais confiável do comportamento da plataforma.

A quantidade de repetições por cenário foi definida considerando o equilíbrio entre rigor experimental e as limitações práticas inerentes à execução de *workloads* em ambientes de computação em nuvem. Como cada execução envolve custos financeiros diretamente relacionados ao tempo de uso dos recursos, especialmente do Spark Pool, optou-se por realizar três execuções independentes por cenário. Esse número é suficiente para reduzir variações operacionais típicas de ambientes elásticos, ao mesmo tempo, em que respeita o orçamento disponível para o projeto.

Pelas mesmas razões, o dimensionamento do cluster utilizado e as volumetrias escolhidas também foram consideradas a partir de restrições de custo e capacidade. O cluster foi configurado com um número reduzido de nós e memória, refletindo não somente a realidade de equipes que operam com recursos limitados, mas também cenários comuns de ambientes corporativos que adotam práticas de controle financeiro em nuvem.

Parte IV

Resultados

7 Resultados

Este capítulo apresenta os resultados obtidos a partir da execução dos nove cenários experimentais definidos na metodologia. O objetivo é descrever, de forma quantitativa e organizada, o comportamento da plataforma de dados diante das diferentes volumetrias e etapas do pipeline avaliadas.

As métricas apresentadas incluem o tempo total de execução, a latência introduzida pela orquestração e o custo operacional estimado para cada etapa. Cada cenário foi executado três vezes, de modo a reduzir a variabilidade típica de ambientes de computação em nuvem e fornecer estimativas mais consistentes.

É importante esclarecer que os resultados exibidos neste capítulo permitem observar tendências relacionadas à escalabilidade da solução. Entretanto, a escalabilidade não é uma métrica diretamente fornecida pelo sistema. Sua avaliação depende da interpretação conjunta dos valores coletados para tempo, latência e custo sob diferentes volumetrias. Desse modo, qualquer análise sobre o comportamento escalável do pipeline, bem como considerações sobre eficiência, estabilidade e eventuais gargalos, será desenvolvida somente no capítulo seguinte, intitulado *Discussão e Análise dos Resultados*.

7.1 Estrutura da Avaliação

A avaliação experimental foi estruturada para isolar os efeitos da volumetria e das diferentes etapas do pipeline sobre o desempenho e os custos da arquitetura. Para isso, dois fatores principais foram definidos:

- **Volumetria dos dados**: 10 GB, 40 GB e 100 GB.
- **Etapa do pipeline**: ingestão, transformação Bronze → Silver e transformação Silver → Gold.

A combinação entre esses fatores originou nove cenários experimentais (C1–C9), cada um representando uma configuração específica de carga e complexidade de processamento. Essa organização permite avaliar separadamente os comportamentos de cada etapa, bem como identificar tendências gerais à medida que a volumetria aumenta.

7.2 Cenários do Experimento

Os nove cenários experimentais adotados neste estudo derivam da combinação entre os fatores metodológicos previamente definidos: as três volumetrias de dados (10 GB, 40 GB e 100 GB) e as três etapas do pipeline baseadas na arquitetura *Medallion* (ingestão, Bronze → Silver e Silver → Gold). A descrição detalhada de cada cenário encontra-se na Seção 6.5; nesta seção, apresenta-se apenas uma síntese.

Em linhas gerais, cada cenário representa uma configuração específica de carga e estágio de processamento, permitindo isolar e analisar os efeitos da volumetria e da etapa do fluxo de dados sobre o comportamento da plataforma. Para mitigar a variabilidade inerente ao ambiente de nuvem e aumentar a confiabilidade das métricas obtidas, todos os cenários foram executados de forma independente em três repetições.

7.3 Resultados dos Cenários

Esta seção apresenta os resultados obtidos nos nove cenários experimentais, organizados por métricas. Com o objetivo de proporcionar uma visão mais clara e objetiva, são apresentados os valores médios consolidados referentes às execuções realizadas em cada cenário.

As tabelas completas que apresentam o detalhamento individual de cada execução, assim como todas as operações realizadas no Azure Data Lake Storage, foram reunidas no Apêndice A. Essa organização visa centralizar no texto principal os valores representativos, mantendo a integridade dos registros técnicos acessíveis para consulta.

Ademais, a interpretação e discussão serão desenvolvidas no capítulo subsequente, dedicado exclusivamente à análise crítica dos dados obtidos.

7.3.1 Tempo de Execução, Tempo de Processamento e Latência de Orquestração

Os resultados consolidados desta subseção sintetizam três métricas centrais do comportamento temporal da plataforma:

- Tempo Total de Execução
- Tempo Efetivo de Processamento
- Latência de Orquestração

Cada valor representa à média simples das três execuções realizadas, segmentada por cenário.

Tabela 6 – Tempo de Execução e Latência de Orquestração Consolidado

Cenário	Tempo Exec. (min)	Tempo Proc. (min)	Latência Orq. (s)
C1 — Ingestão (10 GB)	0.83	0.69	8.67
C2 — Ingestão (40 GB)	1.55	1.40	9.33
C3 — Ingestão (100 GB)	2.51	2.35	9.66
C4 — Bronze → Silver (10 GB)	8.65	7.37	77.33
C5 — Bronze → Silver (40 GB)	25.79	24.30	89.33
C6 — Bronze → Silver (100 GB)	61.77	60.83	96.66
C7 — Silver → Gold (10 GB)	7.91	6.41	90.33
C8 — Silver → Gold (40 GB)	25.72	24.21	91.00
C9 — Silver → Gold (100 GB)	67.38	65.95	86.33

7.3.2 Custo Operacional Consolidado

Os custos operacionais são apresentados de forma consolidada e segmentados em três componentes:

- Orquestração no Azure Data Factory
- Processamento no Azure Synapse
- Armazenamento no Azure Data Lake Storage

O componente de orquestração reflete o custo associado à execução das atividades do pipeline. Nos cenários de ingestão (C1–C3), esse valor resulta principalmente do consumo de DIU-h da *Copy Activity*. Nos cenários que acionam notebooks Spark (C4–C9), o custo corresponde às *External Activities*, cujo valor unitário permanece baixo, sendo registrado para garantir maior precisão na composição do custo total.

O componente de processamento aplica-se exclusivamente aos cenários que executam cargas utilizando Apache Spark (C4–C9). O valor consolidado representa o custo proporcional ao tempo de processamento observado em cada etapa, calculado a partir da tarifa horária do Spark Pool utilizada.

Por fim, o componente de armazenamento reflete o custo derivado do número de operações de leitura e escrita executadas no ADLS em cada cenário.

A Tabela 7 apresenta o consolidado final de custos para os nove cenários.

Tabela 7 – Custo operacional médio por componente e cenário (US$)

Cenário	**Orquestração**	**Processamento**	**Armazenamento**	**Total**
C1 — Ingestão (10 GB)	0.14	—	0.02	0.16
C2 — Ingestão (40 GB)	0.15	—	0.08	0.28
C3 — Ingestão (100 GB)	0.28	—	0.21	0.49
C4 — Bronze → Silver (10 GB)	0.01	0.28	0.03	0.32
C5 — Bronze → Silver (40 GB)	0.01	0.93	0.05	0.99
C6 — Bronze → Silver (100 GB)	0.01	2.30	0.13	2.44
C7 — Silver → Gold (10 GB)	0.01	0.24	0.01	0.26
C8 — Silver → Gold (40 GB)	0.01	0.92	0.04	0.97
C9 — Silver → Gold (100 GB)	0.01	2.52	0.13	2.66

8 Discussão e Análise dos Resultados

Este capítulo visa interpretar os resultados experimentais obtidos a partir da plataforma implementada, relacionando-os com a hipótese de trabalho e com os fundamentos teóricos apresentados nos capítulos anteriores. A partir das métricas consolidadas de tempo, custo e operações de armazenamento, busca-se avaliar como a arquitetura de *Data Lakehouse* construída se comporta em relação a escalabilidade e previsibilidade de custos em cenários variados.

A discussão está estruturada em quatro pilares principais. Inicialmente, apresenta-se uma visão geral dos resultados, destacando os padrões globais que foram observados nas diferentes etapas do pipeline. Em sequência, é analisada a escalabilidade temporal da arquitetura, comparando o comportamento da ingestão com as transformações executadas no Azure Synapse. Em seguida, analisa-se a composição dos custos por componente (orquestração, processamento e armazenamento), dando enfoque na distribuição relativa desses custos por volumetria. Por fim, são discutidos aspectos de estabilidade e variabilidade das execuções.

8.1 Escalabilidade da Arquitetura

8.1.1 Escalabilidade da Ingestão (Source → Bronze)

A etapa de ingestão apresenta um comportamento altamente eficiente em relação à escalabilidade. Ao comparar os cenários C1, C2 e C3, é notório que o aumento de 10 vezes na volumetria de dados (10 GB → 100 GB) resulta em um aumento de aproximadamente 3 vezes no tempo médio de execução (0,83 min → 2,51 min) 10. Este comportamento sub-linear indica que a *Copy Activity* do Azure Data Factory explora paralelismo interno e otimizações de I/O, impedindo que o tempo cresça de forma proporcional ao volume processado.

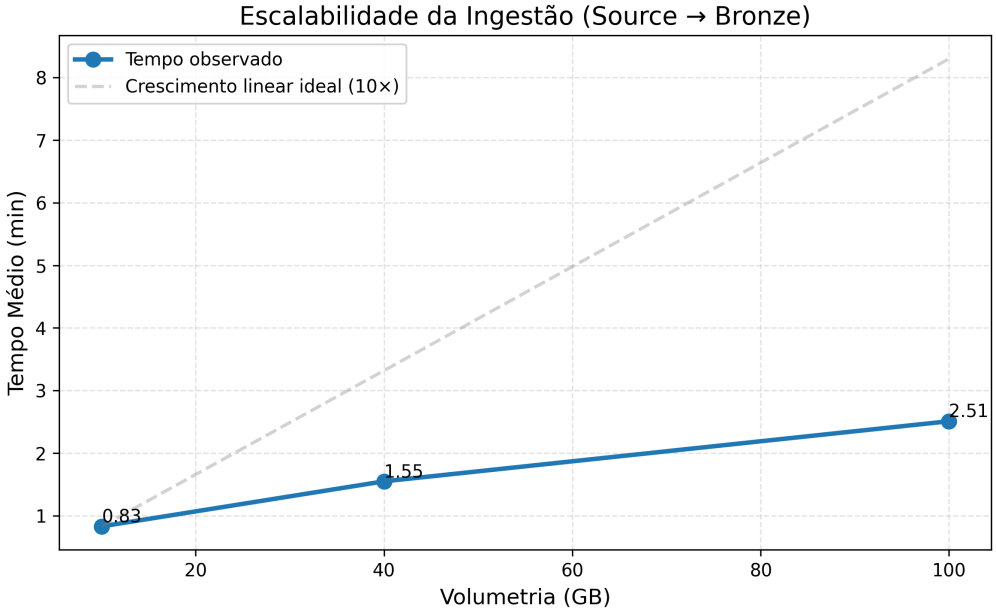

Figura 10 – Escalabilidade sub-linear Ingestão de Dados.

Além disso, a latência de orquestração permanece praticamente constante (entre 8,67 s e 9,66 s), evidenciando que a sobrecarga associada ao Azure Data Factory é independente do volume de dados. Em conjunto, esses fatores demonstram que a ingestão não constitui gargalo mesmo em cenários de expansão significativa da volumetria.

8.1.2 Escalabilidade do Processamento (Bronze → Silver e Silver → Gold)

Nas etapas de transformação, o comportamento muda drasticamente em relação à ingestão. Tanto em Bronze → Silver quanto em Silver → Gold, o crescimento de tempo, ainda que sub-linear, se aproxima de uma tendência linear conforme a volumetria cresce 11 e 12. No primeiro caso, elevar o volume de 10 para 100 GB aumenta o tempo de 8,65 min para 61,77 min (cerca de 7,1 vezes). No segundo, o tempo cresce de 7,91 min para 67,38 min (aproximadamente 8,5 vezes).

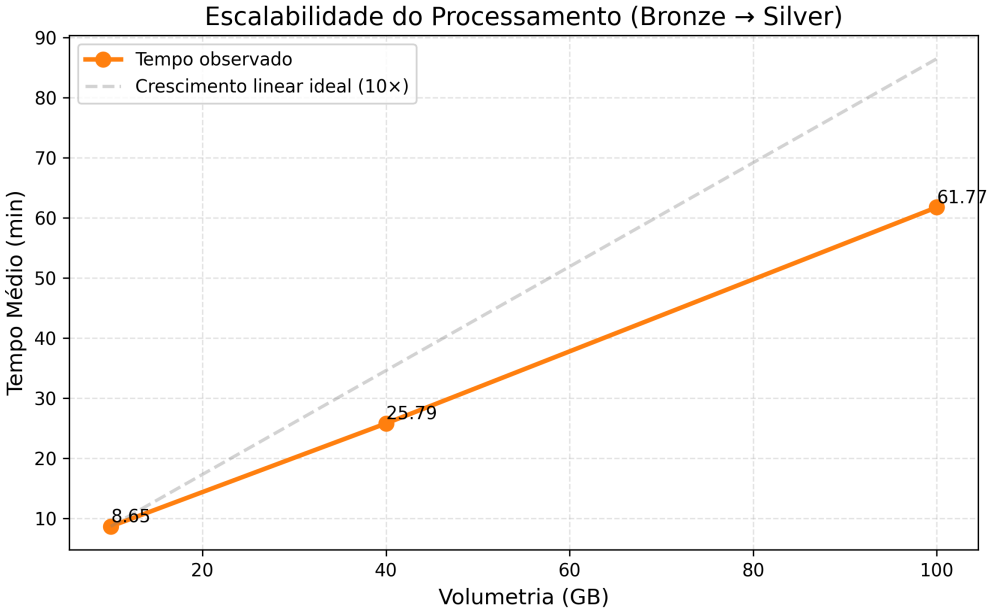

Figura 11 – Escalabilidade sub-linear Bronze → Silver.

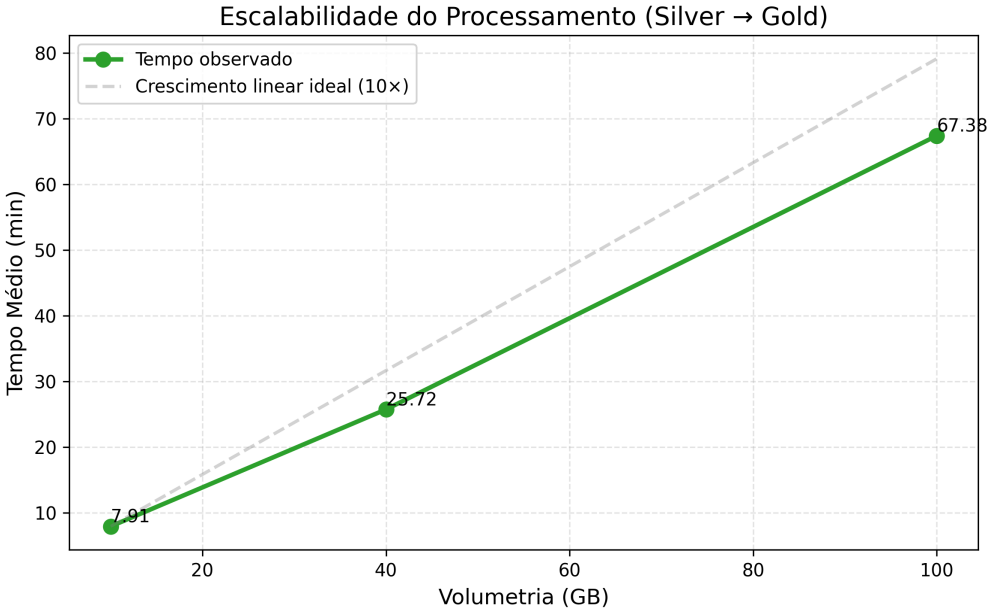

Figura 12 – Escalabilidade sub-linear Silver → Gold.

Esse padrão decorre das características intrínsecas do Apache Spark. Operações como shuffle, serialização e redistribuição de partições, além da escrita transacional do Delta Lake, introduzem sobrecargas que crescem proporcionalmente ao tamanho dos dados. Mesmo com paralelismo, o pipeline executa transformações computacionalmente intensivas (*CPU-bound*), cujo tempo aumenta de forma proporcional ao volume processado.

Outro aspecto fundamental na interpretação dos resultados é a configuração do cluster utilizada. Todas as execuções foram conduzidas em um Spark Pool de tamanho *Small*, composto por apenas três nós e sem escalabilidade automática. Essa configuração fixa impõe um limite claro à paralelização e restringe a capacidade de acomodação de cargas maiores. Ainda assim, o comportamento observado foi melhor do que o esperado para um cluster rigidamente limitado: mesmo com a infraestrutura travada, o tempo de processamento apresentou crescimento sub-linear em relação ao volume total de dados, indicando que o Spark conseguiu explorar de forma eficiente o paralelismo disponível dentro das restrições impostas pelo ambiente experimental.

O contraste com a etapa de ingestão evidencia ainda mais essas diferenças estruturais. Enquanto o cluster de tamanho fixo do Spark Pool opera com paralelismo controlado, o *Copy Activity* do Azure Data Factory faz uso automático de estratégias de transferência paralela, particionamento de arquivos e múltiplos fluxos de leitura/escrita. Como resultado, seu comportamento é marcadamente sub-linear, mesmo quando o volume de dados cresce em uma ordem de magnitude.

Apesar das limitações do cluster, as transformações demonstraram um comportamento altamente previsível: tempos crescem de forma proporcional ao volume e sem instabilidades operacionais. Essa consistência é essencial para planejamento de capacidade e estimativa de custos em ambientes corporativos.

8.2 Composição de Custos por Componente

A composição dos custos revela padrões distintos entre os componentes tarifáveis da arquitetura. A **orquestração** apresenta participação proporcionalmente maior nos cenários de menor volumetria, já que seu custo tende a se manter constante por execução. Assim, em 10 GB ela representa mais de um quinto do custo total, mas sua relevância cai rapidamente à medida que o volume aumenta, tornando-se praticamente marginal em 100 GB.

O **armazenamento** mantém comportamento estável em todas as volumetrias, com participação relativa próxima de 8% a 9%. Esse padrão reflete a natureza proporcional das operações de leitura e escrita no ADLS, que crescem de forma previsível conforme o número de arquivos manipulados aumenta.

Já o **processamento** emerge como o componente dominante do custo total. Sua participação cresce de forma consistente com o volume de dados, passando de cerca de 70% em 10 GB para mais de 86% em 100 GB. Essa predominância evidencia que a dimensão computacional do pipeline, especialmente nas etapas de transformação, é o principal determinante econômico da arquitetura.

A Figura 13 sintetiza esse comportamento, mostrando como os custos estruturais se diluem com o aumento de volume, enquanto os custos computacionais assumem papel central na formação do custo final.

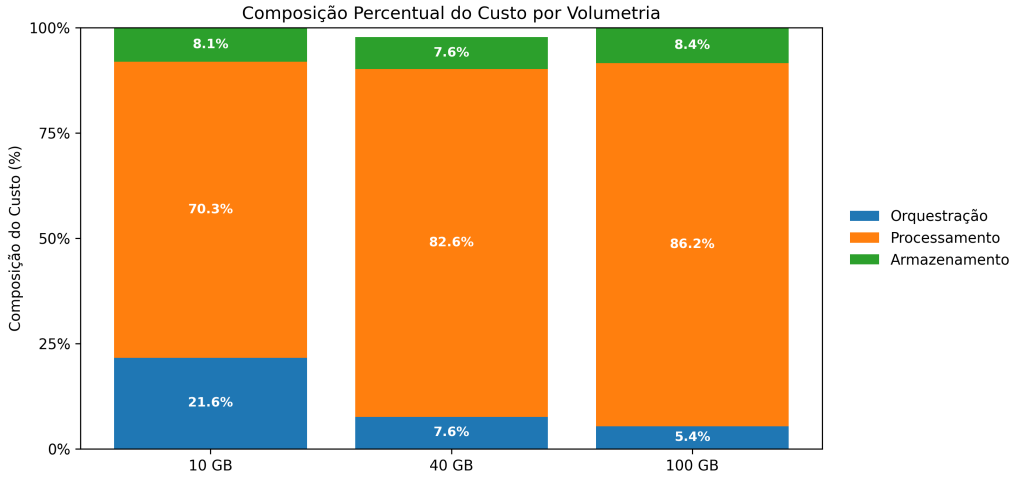

Figura 13 – Composição percentual dos custos por componente em diferentes volumetrias.

8.3 Estabilidade das Execuções e Variabilidade das Métricas

A análise conjunta dos gráficos de dispersão de custo total, latência de orquestração e tempo de execução mostra que o pipeline manteve um comportamento altamente estável em todas as volumetrias avaliadas. Cada ponto representa uma execução individual e a proximidade entre eles evidencia a baixa variabilidade observada nas três repetições realizadas para cada cenário.

No gráfico de custo 14, os valores de ingestão, processamento e armazenamento permanecem praticamente idênticos entre as execuções. Mesmo nos cenários de maior volume, a dispersão é mínima, o que indica que o custo final é previsível e diretamente relacionado ao volume de dados, sem oscilações operacionais relevantes.

O gráfico de latência de orquestração 15 apresenta o mesmo padrão de estabilidade. A latência da ingestão permanece concentrada na faixa de poucos segundos, enquanto a latência do processamento se mantém entre 70 e 120 segundos, variando pouco dentro de cada volumetria. Isso confirma que os mecanismos de acionamento dos pipelines e inicialização dos recursos funcionam de forma consistente.

O gráfico de tempo 16 de execução também reforça essa estabilidade. A etapa de ingestão apresenta tempos quase idênticos entre as execuções e cresce de forma proporcional ao volume. A etapa de processamento, apesar de naturalmente mais longa, mantém dispersão reduzida e comportamento regular, indicando que a alocação de recursos e o paralelismo interno do Spark permaneceram uniformes ao longo dos testes.

A leitura integrada desses três gráficos mostra que o sistema opera com alto nível de previsibilidade. Não foram observadas flutuações significativas que comprometessem a consistência das medições, e todas as 27 execuções ocorreram sem falhas, o que confirma a confiabilidade prática dos serviços Azure utilizados e a robustez da arquitetura implementada.

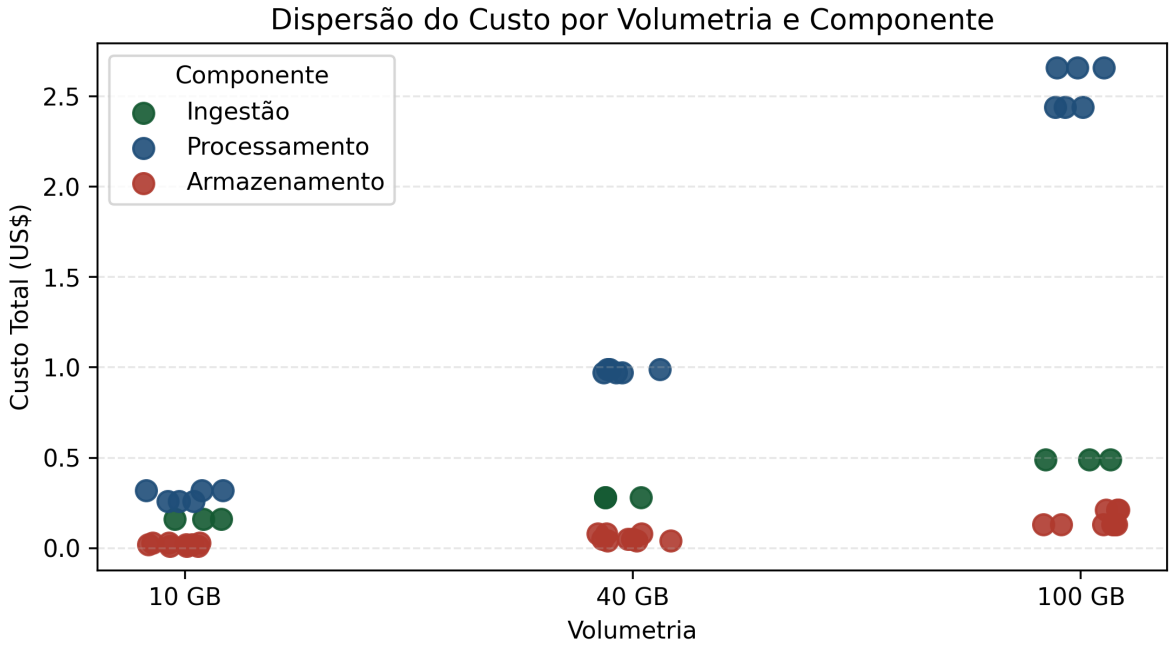

Figura 14 – Dispersão do custo total por volumetria e componente.

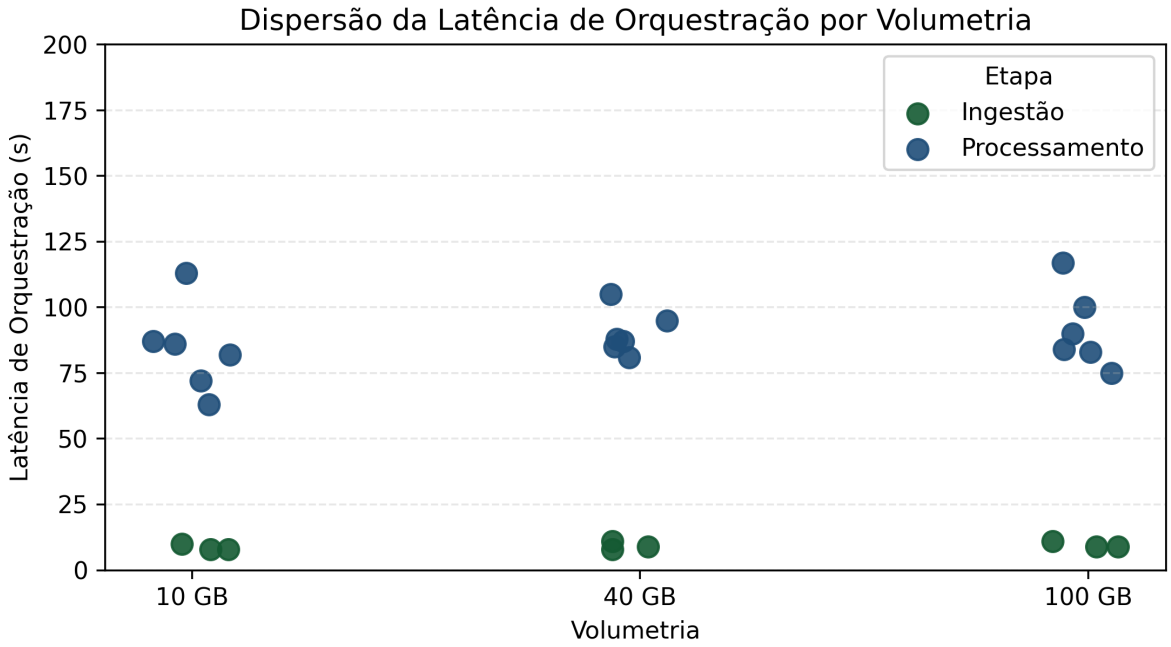

Figura 15 – Dispersão da latência de orquestração para ingestão e processamento.

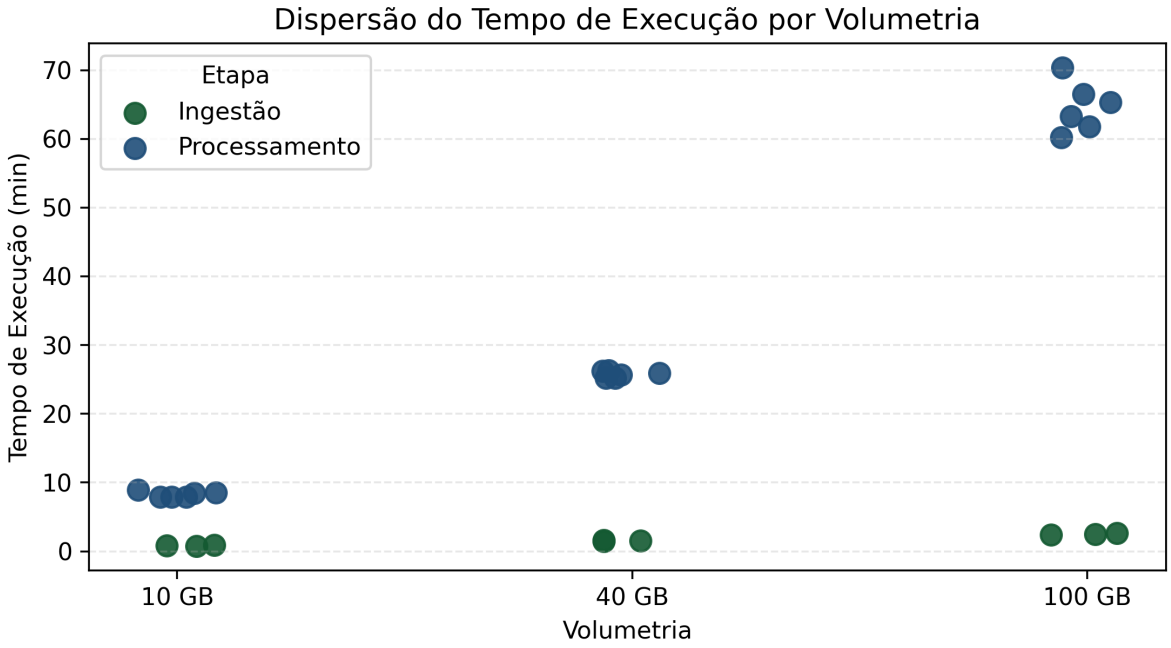

Figura 16 – Dispersão do tempo total de execução por volumetria para ingestão e processamento.

Parte V

Conclusões

Este trabalho atingiu seu objetivo principal ao projetar, implementar e validar uma Plataforma Moderna de Dados na nuvem Microsoft Azure, fundamentada na arquitetura *Medallion* e no paradigma de *Data Lakehouse*. A infraestrutura provisionada via Terraform (IaC) provou-se robusta, garantindo a reprodutibilidade integral dos ambientes experimentais e a consistência das configurações de segurança e acesso.

Os experimentos demonstraram que a arquitetura consegue escalar eficientemente. A separação entre computação (Azure Synapse - Spark) e armazenamento (ADLS) permitiu que o processamento de 100 GB de dados ocorresse sem degradação de desempenho, mantendo o custo por gigabyte constante. A implementação das camadas Bronze, Silver e Gold, suportada pelo formato Delta Lake, assegurou não somente a organização lógica, mas também a integridade transacional e a evolução de esquema (*schema evolution*) necessárias para ambientes corporativos.

Os resultados obtidos permitiram confirmar as hipóteses norteadoras deste estudo. A Hipótese 1 (H1) foi validada pela observação de um custo marginal constante (aproximadamente US$ 0,026 por Gigabyte) em todos os cenários de processamento, comprovando que o desacoplamento entre computação e armazenamento em arquiteturas de *Data Lakehouse* na nuvem viabiliza uma escalabilidade financeira linear. Simultaneamente, a Hipótese 2 (H2) foi corroborada pela estabilidade da latência de orquestração, que se manteve independente da volumetria dos dados processados. Isso demonstra que o overhead de inicialização dos serviços gerenciados é fixo e previsível, ratificando a adequação da arquitetura para *workloads* em lote (*Batch*), ao mesmo tempo que evidencia suas limitações naturais para cenários de tempo real estrito.

Conclui-se que a adoção de serviços gerenciados (PaaS) combinada com boas práticas de engenharia de dados (IaC, CI/CD, *Medallion*) reduz drasticamente a complexidade operacional. A plataforma consegue transformar dados brutos em ativos analíticos de valor com previsibilidade de custo e desempenho.

9 Principais Resultados

A realização deste estudo resultou na entrega de uma plataforma de dados funcional, escalável e economicamente viável, validada por experimentos quantitativos. As principais contribuições deste trabalho podem ser categorizadas em três pilares fundamentais:

- **Arquitetura de Referência Reprodutível (IaC)**: Um dos resultados mais importante foi a consolidação de um código de infraestrutura (Terraform) capaz de provisionar, em minutos, um ambiente analítico completo na nuvem Microsoft Azure. A modularização dos recursos (Synapse, Data Factory, ADLS Gen2 e Key Vault) demonstrou ser possível eliminar a configuração manual sujeita a erros, garantindo que ambientes de desenvolvimento, teste e produção mantenham consistência idêntica. O artefato de código produzido serve como um marcador para implementações corporativas de *Data Lakehouse*.
- **Validação de Escalabilidade e Custo Linear:** Os experimentos comprovaram que a arquitetura proposta suporta o crescimento exponencial de dados sem degradação exponencial de custos. A identificação de um custo de processamento estável de aproximadamente US$ 0,026 por Gigabyte (nos cenários de 10 GB a 100 GB) oferece um modelo de previsibilidade financeira para gestores de TI. Demonstrou-se que o desacoplamento entre armazenamento (ADLS) e computação (Spark Pools) permite escalar a volumetria mantendo a eficiência operacional.
- **Implementação Eficiente da Arquitetura *Medallion*:** O estudo entregou um pipeline de dados operando em três camadas lógicas (Bronze, Silver, Gold). A utilização do formato Delta Lake provou-se eficaz para garantir propriedades ACID e evolução de esquema (*schema evolution*) em um ambiente de *Data Lake*, resolvendo problemas históricos de inconsistência de dados em sistemas de arquivos distribuídos. A transformação de dados brutos de IoT em insights agregados na camada Gold foi realizada com tempos de execução aceitáveis para janelas de processamento *Batch*, validando a escolha tecnológica do Apache Spark.

10 Trabalhos Futuros

Embora a arquitetura implementada tenha atendido aos objetivos de processamento em lote (*Batch*) com alta eficiência, a evolução contínua da engenharia de dados aponta para novas oportunidades de pesquisa e aprimoramento. Sugerem-se as seguintes linhas de investigação para trabalhos futuros:

- **Comparativo com Outras Engines de Processamento:** Realizar um estudo comparativo de desempenho e custo substituindo o Azure Synapse Spark pelo Azure Databricks. Tal análise permitiria verificar se as otimizações proprietárias do Databricks (como o motor Photon) justificam a diferença de custo em relação ao Synapse para volumetrias da ordem de Terabytes.
- **Automação de Deploy (CI/CD) e DevOps:** A execução do Terraform foi realizada localmente. Para um cenário produtivo, sugere-se a integração dos *scripts* de infraestrutura e dos notebooks Spark em esteiras de CI/CD (*Continuous Integration/Continuous Delivery*) utilizando Azure DevOps ou GitHub Actions. Isso permitiria testar a agilidade na entrega de novas features e a governança das alterações de código.
- **Otimizações Avançadas do Delta Lake:** O estudo utilizou as funcionalidades básicas do Delta Lake, porém sem empregar recursos avançados como `OPTIMIZE`, `ZORDER`, compactação automática de pequenos arquivos e regras de *retention*. Investigar essas estratégias permitiria avaliar o impacto real na redução de latência, nos custos de leitura/escrita e na eficiência das consultas analíticas em camadas Silver e Gold.
- **Escalabilidade Vertical e Horizontal do Spark Pool:** A arquitetura analisada fixou o cluster em uma configuração Small (3 nós). Trabalhos futuros podem explorar diferentes tamanhos de nó, diferentes quantidades de executores e o uso de autoscaling. Tal investigação permitiria identificar o ponto ótimo de custo-performance para diferentes volumetrias e avaliar se gargalos observados nas transformações são consequência direta das limitações de recursos.

Referências

ABOUZAID, Ahmed et al. Building a modern data platform based on the data lakehouse architecture and cloud-native ecosystem. *Discover Applied Sciences*, v. 7, n. 166, 2025. DOI: 10.1007/s42452-025-06545-w. Disponível em: https://doi.org/10.1007/s42452-025-06545-w. Citado 2 vezes nas páginas 37, 51.

AGARWAL, Giriraj. Robust Data Pipelines for AI Workloads: Architectures, Challenges, and Future Directions. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, v. 5, n. 2, p. 622–629, fev. 2025. ISSN (Online): 2581-9429, Impact Factor: 7.687. DOI: 10.48175/IJARSCT-23391. Disponível em: https://ijarsct.co.in/Paper23391.pdf. Citado 2 vezes na página 29.

AHMED, Nasim et al. A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. *Journal of Big Data*, Springer, v. 7, n. 1, p. 110, 2020. DOI: 10.1186/s40537-020-00388-5. Disponível em: https://doi.org/10.1186/s40537-020-00388-5. Citado 1 vez na página 37.

ALHASSAN, Ibrahim; SAMMON, David; AND, Mary Daly. Data governance activities: an analysis of the literature. *Journal of Decision Systems*, Taylor & Francis, v. 25, sup1, p. 64–75, 2016. DOI: 10.1080/12460125.2016.1187397. eprint: https://doi.org/10.1080/12460125.2016.1187397. Disponível em: https://doi.org/10.1080/12460125.2016.1187397. Citado 1 vez na página 47.

ARMBRUST, Michael; DAS, Tathagata et al. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 13, n. 12, p. 3411–3424, 2020. Presented at the 46th International Conference on Very Large Data Bases (VLDB 2020). DOI: 10.14778/3415478.3415560. Disponível em: https://doi.org/10.14778/3415478.3415560. Citado 2 vezes nas páginas 43, 44.

ARMBRUST, Michael; GHODSI, Ali et al. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In: 11TH Annual Conference on Innovative Data Systems Research (CIDR'21). Online: CIDR, jan. 2021. P. 1–15. Disponível em: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf. Citado 1 vez na página 28.

AZEROUAL, Otmane; FABRE, Renaud. Processing Big Data with Apache Hadoop in the Current Challenging Era of COVID-19. *Big Data and Cognitive Computing*, v. 5, n. 1, p. 12, 2021. DOI: 10.3390/bdcc5010012. Disponível em: https://doi.org/10.3390/bdcc5010012. Citado 3 vezes na página 39.

BATINI, Carlo; SCANNAPIECO, Monica. Data quality: The other face of Big Data. *2014 IEEE International Conference on Big Data (Big Data)*, IEEE, p. 1–6, 2014. DOI: 10.1109/BigData.2014.7004474. Disponível em: https://ieeexplore.ieee.org/document/6816764. Citado 1 vez na página 48.

BHUVA, Mehul K. Implementing Enterprise-Wide Lakehouse using Microsoft Azure Databricks and Delta Lake. *International Journal of Computer Trends and Technology*, v. 73, n. 4, p. 135–139, abr. 2025. ISSN 2231-2803. DOI: 10.14445/22312803/IJCTT-V73I4P119. Disponível em: https://doi.org/10.14445/22312803/IJCTT-V73I4P119. Citado 1 vez na página 51.

CHAMBERS, Bill; ZAHARIA, Matei. *Spark: The Definitive Guide*. 1. ed. Sebastopol, CA: O'Reilly Media, 2018. Citado 2 vezes nas páginas 39, 40.

CHENG, Yan; ZHANG, Qiang; YE, Ziming. Research on the Application of Agricultural Big Data Processing with Hadoop and Spark. In: 2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA). IEEE, 2019. P. 274–278. DOI: 10.1109/ICAICA.2019.8877790. Citado 1 vez na página 80.

CHEONG, Lai Kuan; CHANG, Vanessa. The Need for Data Governance: A Case Study. In: PROCEEDINGS of the 18th Australasian Conference on Information Systems (ACIS 2007). Toowoomba, Australia, 2007. P. 1–10. Disponível em: https://aisel.aisnet.org/acis2007/100. Citado 1 vez na página 47.

DATABRICKS. *Medallion Architecture*. Accessed on March 24, 2025. 2025. Disponível em: https://www.databricks.com/glossary/medallion-architecture. Citado 1 vez nas páginas 40, 42.

DATABRICKS. *What is a Data Lakehouse?* Accessed on: 1 November 2025. 2025. Disponível em: https://www.databricks.com/glossary/data-lakehouse. Citado 1 vez na página 27.

DATABRICKS. *What is Delta Lake?* Accessed on: 20 October 2025. 2025. Disponível em: https://docs.databricks.com/aws/en/delta/. Citado 1 vez na página 44.

DELOITTE. *Data as a Strategic Asset*. Accessed on May 21, 2025. 2021. Disponível em: https://www2.deloitte.com/us/en/pages/consulting/articles/data-strategic-asset.html. Citado 1 vez na página 27.

FEICK, Martin; KLEER, Niko; KOHN, Marek. Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa. In: SKILL 2018 - Studierendenkonferenz Informatik. Bonn: Gesellschaft für Informatik e.V., 2018. P. 55–66. ISBN 978-3-88579-448-6. Citado 1 vez na página 40.

FOUNDATION, Apache Software. *Cluster Mode Overview - Spark 4.0.1 Documentation*. 2025. https://spark.apache.org/docs/latest/cluster-overview.html. Accessed: 2025-12-11. Citado 0 vez na página 40.

FUTUREIOT. *IDC Forecasts Connected Iomuch IoT Devices to Generate 79.4ZB of Data in 2025*. Accessed on May 22, 2025. 2020. Disponível em: https://futureiot.tech/idc-forecasts-connected-iot-devices-to-generate-79-4zb-of-data-in-2025/. Citado 1 vez na página 27.

GAO, Penglin; HAN, Zhaoming; WAN, Fucheng. Big Data Processing and Application Research. In: 2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM). Beijing, China: IEEE, 2020. P. 125–128. DOI: 10.1109/AIAM50918.2020.00031. ISBN 978-1-7281-9986-3. DOI: 10.1109/AIAM50918.2020.00031. Citado 1 vez na página 81.

GIL, Antonio Carlos. *Métodos e Técnicas de Pesquisa Social*. 6. ed. São Paulo: Atlas, 2008. Citado 1 vez na página 61.

HARVARD BUSINESS REVIEW. *Big on Data: Study Shows Why Data-Driven Companies Are More Profitable than Their Peers*. Mar. 2023. Sponsored by Google Cloud. Available at: https://cloud.google.com/transform/data-leaders-more-profitable-innovative-hbr-data. Disponível em: https://cloud.google.com/transform/data-leaders-more-profitable-innovative-hbr-data. Acesso em: 16 nov. 2025. Citado 1 vez na página 27.

HASHICORP. *Terraform Documentation*. 2025. Accessed: 2025-12-08. Disponível em: https://developer.hashicorp.com/terraform/docs. Citado 3 vezes nas páginas 55, 56.

INMON, W. H. *Building the Data Warehouse*. 3. ed. New York, NY: John Wiley & Sons, 2002. ISBN 978-0-471-08130-9. Citado 2 vezes nas páginas 45, 47.

KARUNAKARAN, Shilesh; AGARWAL, Raghav. Building Robust and Scalable Data Pipelines Using Apache Kafka, Spark, AWS Lambda and AWS Glue. *International Research Journal of Modernization in Engineering Technology and Science (IRJMETS)*, v. 7, n. 4, p. 7610–7619, abr. 2025. e-ISSN: 2582-5208, Impact Factor: 8.187. DOI: 10.56726/IRJMETS74238. Disponível em: https://www.irjmets.com/uploadedfiles/paper/issue_4_april_2025/74238/final/fin_irjmets1745755846.pdf. Citado 3 vezes nas páginas 28, 29.

KHANRA, Sayantan; DHIR, Amandeep; MÄNTYMÄKI, Matti. Big Data Analytics and Enterprises: A Bibliometric Synthesis of the Literature. *Enterprise Information Systems*, v. 14, n. 6, p. 737–768, 2020. Citado 1 vez na página 27.

KIMBALL, Ralph; ROSS, Margy. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. 3rd. Indianapolis: Wiley, 2013. Citado 4 vezes nas páginas 35, 46, 47.

KIRAN, Mariam et al. Lambda architecture for cost-effective batch and speed big data processing. In: 2015 IEEE International Conference on Big Data (Big Data). Santa Clara, CA, USA: IEEE, 2015. P. 2785–2792. DOI: 10.1109/BigData.2015.7364082. Disponível em: https://ieeexplore.ieee.org/document/7364082. Citado 2 vezes nas páginas 40, 41.

LEKKALA, Chandrakanth. Automating Infrastructure Management with Terraform: Strategies and Impact on Business Efficiency. *European Journal of Advances in Engineering and Technology*, v. 9, n. 11, p. 82–88, nov. 2022. ISSN 2394-658X. Citado 1 vez na página 66.

LIN, Jimmy. The Lambda and the Kappa. *IEEE Internet Computing*, IEEE, v. 21, n. 5, p. 60–66, 2017. Big Data Bites column. DOI: 10.1109/MIC.2017.3575471. Disponível em: https://doi.org/10.1109/MIC.2017.3575471. Citado 1 vez na página 40.

MANCHANA, Ramakrishna. Building a Modern Data Foundation in the Cloud: Data Lakes and Data Lakehouses as Key Enablers. *Journal of Artificial Intelligence, Machine Learning and Data Science*, URF Publishers, v. 1, n. 1, p. 1098–1108, fev. 2023. Received: 02 February 2023; Accepted: 18 February 2023; Published: 20 February 2023. ISSN 2583-9888. DOI: 10.51219/JAIMLD/Ramakrishna-manchana/260. Disponível em: https://urfpublishers.com/journal-article-download/artificial-intelligence/260. Citado 1 vez na página 51.

MEEHAN, John et al. Data Ingestion for the Connected World. In: 9TH Biennial Conference on Innovative Data Systems Research (CIDR 2017). Chaminade, CA, USA, jan. 2017. Accessed on: 16 November 2025. Disponível em: http://cidrdb.org/cidr2017/papers/p28-meehan-cidr17.pdf. Citado 2 vezes na página 38.

MICROSOFT. *ETL vs. ELT for Analytical Processes*. Azure Architecture Center (versão em português). 2025. Disponível em: https://learn.microsoft.com/pt-br/azure/architecture/data-guide/relational-data/etl. Acesso em: 16 nov. 2025. Citado 0 vez nas páginas 35, 36.

MICROSOFT. *What is Cloud Computing?* Acesso em: 25 maio 2025. 2024. Disponível em: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing. Citado 1 vez na página 48.

MOHAMED, Norshidah. The impact model of business intelligence on decision support and organizational benefits. *Journal of Enterprise Information Management*, v. 29, n. 1, p. 19–50, 2016. DOI: 10.1108/JEIM-12-2014-0126. Disponível em: https://www.emerald.com/insight/content/doi/10.1108/JEIM-12-2014-0126/full/html. Citado 1 vez na página 27.

MOHNA, Hosne Ara et al. AI-READY DATA ENGINEERING PIPELINES: A REVIEW OF MEDALLION ARCHITECTURE AND CLOUD-BASED INTEGRATION MODELS.

American Journal of Scholarly Research and Innovation, v. 1, n. 1, p. 319–350, 2022. ISSN 3067-2163. DOI: 10.63125/51kxtf08. Citado 1 vez na página 52.

MOSTAFAEIPOUR, Ali et al. Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. *The Journal of Supercomputing*, Springer, v. 77, n. 2, p. 1273–1300, 2021. DOI: 10.1007/s11227-020-03328-5. Disponível em: https://doi.org/10.1007/s11227-020-03328-5. Citado 1 vez na página 40.

NARAYANAN, Pavan Kumar. *Engineering Data Pipelines Using Microsoft Azure*. Berkeley, CA: Apress, 2024. P. 571–616. Citado 1 vez na página 36.

REIS, Joe; HOUSLEY, Matt. *Fundamentals of Data Engineering: Plan and Build Robust Data Systems*. Sebastopol: O'Reilly Media, 2022. Citado 6 vezes nas páginas 35, 36, 41, 43, 45.

SCHMIDT, Peter et al. Not Small - Not Big Data: the Missing Size in the Data Spectrum. *Acta Polytechnica Hungarica*, v. 22, n. 5, p. 167–184, 2025. ISSN 1785-8860. Citado 1 vez na página 81.

SEAGATE. *Rethink Data Report Reveals That 68 Percent of Data Available to Businesses Goes Unleveraged*. Accessed on May 11, 2025. 2020. Disponível em: https://www.seagate.com/news/news-archive/seagates-rethink-data-report-reveals-that-68-percent-of-data-available-to-businesses-goes-unleveraged-pr-master/. Citado 1 vez na página 27.

SEENIVASAN, Dhamotharan. ETL vs ELT: Choosing the right approach for your data warehouse. *International Journal for Research Trends and Innovation*, v. 7, n. 2, p. 110–122, 2022. Paper ID IJRTI2202018. DOI: 10.2139/ssrn.5148194. Citado 1 vez na página 35.

SHARMA, Anshul; SHARMA, Saurabh. Growth of Digitization and its Impact on Big Data Analytics. *International Journal of Innovative Technology and Exploring Engineering*, v. 10, n. 3, p. 71–76, 2021. Disponível em: https://www.researchgate.net/publication/348605216_Growth_of_Digitization_and_its_Impact_on_Big_Data_Analytics. Citado 1 vez na página 27.

SNOWFLAKE. *Data Strategies for AI Leaders*. Accessed on May 22, 2025. 2024. Disponível em: https://www.snowflake.com/resource/data-strategies-for-ai-leaders. Citado 1 vez na página 27.

SSERUNJOGI, Richard et al. Design and Evaluation of a Scalable Data Pipeline for AI-Driven Air Quality Monitoring in Low-Resource Settings. *arXiv preprint arXiv:2508.14451*, ago. 2025. Preprint, submetido em 20 de agosto de 2025. arXiv: 2508.14451 [cs.SE]. Disponível em: https://arxiv.org/abs/2508.14451. Citado 1 vez na página 81.

STRENGHOLT, Piethein. *Building Medallion Architectures: Designing with Delta Lake and Spark*. 1. ed. Sebastopol, CA: O'Reilly Media, 2025. ISBN 978-1-098-17883-3. Citado 1 vez na página 41.

SYED, Shakir; NAMPALLI, Rama Chandra Rao. Data Lineage Strategies – A Modernized View. *Educational Administration: Theory and Practice*, v. 26, n. 4, p. 965–973, 2020. Research Article. ISSN 2148-2403. DOI: 10.53555/kuey.v26i4.8104. Disponível em: https://kuey.net/. Citado 1 vez na página 48.

YIN, Robert K. *Estudo de Caso: Planejamento e Métodos*. 5. ed. Porto Alegre: Bookman, 2015. Citado 1 vez na página 61.

ZAHARIA, Matei et al. Apache Spark: A Unified Engine for Big Data Processing. *Communications of the ACM*, v. 59, n. 11, p. 56–65, 2016. DOI: 10.1145/2934664. Disponível em: https://doi.org/10.1145/2934664. Citado 1 vez na página 37.

ZHYRENKOV, Oleksii; DOROSHENKO, Anatolii. A Proposal of Integrated Component-Based Big Data Architecture. In: PROCEEDINGS of the Workshop "Software Engineering and Semantic Technologies" (SEST), co-located with the 15th International Scientific and Practical Programming Conference UkrPROG'2025. Kyiv, Ukraine: CEUR Workshop Proceedings, mai. 2025. (CEUR-WS.org), p. 1–14. Disponível em: http://ceur-ws.org/Vol-XXXX/. Citado 1 vez na página 51.

ANEXO A – Resultados Detalhados dos Cenários

Este anexo apresenta os resultados detalhados dos nove cenários experimentais (C1–C9), incluindo as métricas temporais por execução, os custos de orquestração e processamento por cenário e o volume de operações realizadas no ADLS, bem como o custo de armazenamento associado. Esses dados complementam as tabelas consolidadas apresentadas no Capítulo de Resultados.

A.1 Métricas Temporais por Execução e Cenário

Tabela 8 – Resultados do cenário C1 — Ingestão (10 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	0.85	0.68	10
Exec. 2	0.92	0.78	8
Exec. 3	0.73	0.60	8
Média	**0.83**	**0.69**	**8.67**

Tabela 9 – Resultados do cenário C2 — Ingestão (40 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	1.55	1.40	9
Exec. 2	1.45	1.32	8
Exec. 3	1.65	1.47	11
Média	**1.55**	**1.40**	**9.33**

Tabela 10 – Resultados do cenário C3 — Ingestão (100 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	2.43	2.25	11
Exec. 2	2.60	2.45	9
Exec. 3	2.51	2.36	9
Média	**2.51**	**2.35**	**9.66**

Tabela 11 – Resultados do cenário C4 — Bronze → Silver (10 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	8.45	7.40	63
Exec. 2	8.95	7.50	87
Exec. 3	8.57	7.20	82
Média	**8.65**	**7.37**	**77.33**

Tabela 12 – Resultados do cenário C5 — Bronze → Silver (40 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	25.92	24.34	95
Exec. 2	26.28	24.81	88
Exec. 3	25.18	23.76	85
Média	**25.79**	**24.30**	**89.33**

Tabela 13 – Resultados do cenário C6 — Bronze → Silver (100 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	60.23	58.28	117
Exec. 2	63.28	61.78	90
Exec. 3	61.80	60.42	83
Média	**61.77**	**60.83**	**96.66**

Tabela 14 – Resultados do cenário C7 — Silver → Gold (10 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	7.93	6.05	113
Exec. 2	7.88	6.45	86
Exec. 3	7.93	6.73	72
Média	**7.91**	**6.41**	**90.33**

Tabela 15 – Resultados do cenário C8 — Silver → Gold (40 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	26.22	24.47	105
Exec. 2	25.23	23.78	87
Exec. 3	25.72	24.37	81
Média	**25.72**	**24.21**	**91**

Tabela 16 – Resultados do cenário C9 — Silver → Gold (100 GB)

Execução	Tempo Execução (min)	Tempo Processamento (min)	Latência Orq. (s)
Exec. 1	66.48	64.81	100
Exec. 2	65.30	64.05	75
Exec. 3	70.38	68.98	84
Média	**67.38**	**65.95**	**86.33**

A.2 Custos de Orquestração (Azure Data Factory)

Tabela 17 – Custo de Orquestração — C1

Exec	Ext. Act.	DIU-h	Custo (US$)
1	0	0.53	0.14
2	0	0.53	0.14
3	0	0.53	0.14
Média	**1**	**0.53**	**0.14**

Tabela 18 – Custo de Orquestração — C2

Exec	Ext. Act.	DIU-h	Custo (US$)
1	0	0.53	0.14
2	0	0.53	0.14
3	0	0.60	0.16
Média	**1**	**0.55**	**0.15**

Tabela 19 – Custo de Orquestração — C3

Exec	Ext. Act.	DIU-h	Custo (US$)
1	0	1.06	0.27
2	0	1.13	0.29
3	0	1.06	0.27
Média	**1**	**1.08**	**0.28**

Tabela 20 – Custo de Orquestração — C4

Exec	Ext. Act.	DIU-h	Custo (US$)
1	0.15	0	0.01
2	0.15	0	0.01
3	0.15	0	0.01
Média	**0.15**	–	**0.01**

Tabela 21 – Custo de Orquestração — C5

Exec	Ext. Act.	DIU-h	Custo (US$)
1	0.43	0	0.01
2	0.45	0	0.01
3	0.42	0	0.01
Média	**0.43**	–	**0.01**

Tabela 22 – Custo de Orquestração — C6

Exec	Ext. Act.	DIU-h	Custo (US$)
1	1.06	0	0.01
2	1.03	0	0.01
3	1.01	0	0.01
Média	**1.03**	–	**0.01**

Tabela 23 – Custo de Orquestração — C7

Exec	Ext. Act.	DIU-h	Custo (US$)
1	0.13	0	0.01
2	0.13	0	0.01
3	0.13	0	0.01
Média	**0.13**	–	**0.01**

Tabela 24 – Custo de Orquestração — C8

Exec	Ext. Act.	DIU-h	Custo (US$)
1	0.43	0	0.01
2	0.42	0	0.01
3	0.43	0	0.01
Média	**0.43**	–	**0.01**

Tabela 25 – Custo de Orquestração — C9

Exec	Ext. Act.	DIU-h	Custo (US$)
1	1.10	0	0.01
2	1.18	0	0.01
3	1.11	0	0.01
Média	**1.13**	–	**0.01**

A.3 Custos de Processamento (Azure Synapse)

Tabela 26 – C4 — Custo de Processamento

Execução	Tempo Proc. (min)	Custo (US$)
Exec. 1	7.40	0.28
Exec. 2	7.50	0.28
Exec. 3	7.20	0.27
Média	**7.37**	**0.28**

Tabela 27 – C5 — Custo de Processamento

Execução	Tempo Proc. (min)	Custo (US$)
Exec. 1	24.34	0.93
Exec. 2	24.81	0.95
Exec. 3	23.76	0.91
Média	**24.30**	**0.93**

Tabela 28 – C6 — Custo de Processamento

Execução	Tempo Proc. (min)	Custo (US$)
Exec. 1	58.28	2.23
Exec. 2	61.78	2.36
Exec. 3	60.42	2.31
Média	**60.16**	**2.30**

Tabela 29 – C7 — Custo de Processamento

Execução	Tempo Proc. (min)	Custo (US$)
Exec. 1	6.05	0.23
Exec. 2	6.45	0.24
Exec. 3	6.73	0.25
Média	**6.41**	**0.24**

Tabela 30 – C8 — Custo de Processamento

Execução	Tempo Proc. (min)	Custo (US$)
Exec. 1	24.47	0.93
Exec. 2	23.78	0.91
Exec. 3	24.37	0.93
Média	**24.21**	**0.92**

Tabela 31 – C9 — Custo de Processamento

Execução	Tempo Proc. (min)	Custo (US$)
Exec. 1	64.81	2.48
Exec. 2	64.05	2.45
Exec. 3	68.98	2.64
Média	**65.95**	**2.52**

A.4 Operações ADLS e Custos de Armazenamento

Tabela 32 – C1 — Operações ADLS e custo de armazenamento (10GB Ingestion)

Operação	Exec. 1	Exec. 2	Exec. 3	Média
ReadFile	2,640	2,640	2,640	2,640
AppendFile	2,640	2,640	2,640	2,640
LeaseFile	160	160	160	160
CreatePathFile	80	80	80	80
FlushFile	80	80	80	80
DeleteFile	0	80	80	53
BlobPreflightRequest	0	92	92	61
GetFileProperties	80	80	80	80
GetPathStatus	0	0	0	0
GetFilesystemProperties	32	32	32	32
ListBlobs	0	11	10	7
GetBlobServiceProperties	0	0	0	0
GetContainerProperties	0	0	0	0
GetContainerServiceMetadata	0	0	0	0
ListFilesystemDir	0	0	0	0
RenamePath	0	0	0	0
CreatePathDirectory	0	0	0	0
TOTAL READ	**2,753**	**2,880**	**2,873**	**2,835**
TOTAL WRITE	**2,960**	**3,042**	**3,042**	**3,015**
Custo ADLS (US$)	**0.02**	**0.02**	**0.02**	**0.02**

Tabela 33 – C2 — Operações ADLS e custo de armazenamento (40GB Ingestion)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	10,560	10,880	10,560	10,667
AppendFile	10,560	10,560	10,560	10,560
LeaseFile	640	640	640	640
CreatePathFile	320	320	320	320
FlushFile	320	320	320	320
DeleteFile	80	320	320	240
BlobPreflightRequest	94	329	330	251
GetFileProperties	320	957	320	532
GetPathStatus	0	0	0	0
GetFilesystemProperties	32	33	32	32
ListBlobs	13	8	9	10
GetBlobServiceProperties	12	6	8	9
GetContainerProperties	6	3	4	4
GetContainerServiceMetadata	6	3	4	4
ListFilesystemDir	0	0	0	0
RenamePath	0	0	0	0
CreatePathDirectory	2	0	2	1
TOTAL READ	**11,043**	**12,221**	**11,267**	**11,510**
TOTAL WRITE	**11,922**	**12,160**	**12,162**	**12,081**
Custo ADLS (US$)	**0.08**	**0.09**	**0.08**	**0.08**

Tabela 34 – C3 — Operações ADLS e custo de armazenamento (100GB Ingestion)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	26,400	26,400	26,460	26,420
AppendFile	26,400	26,050	26,950	26,467
LeaseFile	1,600	1,580	1,600	1,593
CreatePathFile	800	791	822	804
FlushFile	800	787	829	805
DeleteFile	0	0	800	267
BlobPreflightRequest	2	0	813	272
GetFileProperties	800	800	800	800
GetPathStatus	0	0	118	39
GetFilesystemProperties	32	32	32	32
ListBlobs	2	0	14	5
GetBlobServiceProperties	2	0	9	4
GetContainerProperties	1	0	5	2
GetContainerServiceMetadata	1	0	3	1
ListFilesystemDir	0	0	0	0
RenamePath	0	0	0	0
CreatePathDirectory	0	0	0	0
TOTAL READ	**27,241**	**27,233**	**28,254**	**27,576**
TOTAL WRITE	**29,600**	**29,208**	**31,001**	**29,936**
Custo ADLS (US$)	**0.21**	**0.21**	**0.22**	**0.21**

Tabela 35 – C4 — Operações ADLS e custo de armazenamento (10GB Silver)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	6,100	6,100	6,100	6,100
AppendFile	3,970	3,970	3,970	3,970
LeaseFile	160	160	160	160
CreatePathFile	169	169	169	169
FlushFile	169	169	169	169
DeleteFile	80	80	80	80
BlobPreflightRequest	122	122	122	122
GetFileProperties	80	80	80	80
GetPathStatus	962	962	962	962
GetFilesystemProperties	32	32	32	32
ListBlobs	13	13	13	13
GetBlobServiceProperties	5	5	5	5
GetContainerProperties	3	3	3	3
GetContainerServiceMetadata	0	0	0	0
ListFilesystemDir	0	0	0	0
RenamePath	0	0	0	0
CreatePathDirectory	0	0	0	0
TOTAL READ	**7,258**	**7,258**	**7,258**	**7,257**
TOTAL WRITE	**4,670**	**4,670**	**4,670**	**4,670**
Custo ADLS (US$)	**0.03**	**0.03**	**0.03**	**0.03**

Tabela 36 – C5 — Operações ADLS e custo de armazenamento (40GB Silver)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	13,870	13,880	13,900	13,883
AppendFile	5,040	5,310	5,310	5,220
LeaseFile	0	0	0	0
CreatePathFile	384	394	394	391
FlushFile	376	394	394	388
DeleteFile	0	0	0	0
BlobPreflightRequest	0	0	0	0
GetFileProperties	0	0	0	0
GetPathStatus	0	0	0	0
GetFilesystemProperties	3,900	3,960	3,980	3,947
ListBlobs	0	0	0	0
GetBlobServiceProperties	0	0	0	0
GetContainerProperties	0	0	0	0
GetContainerServiceMetadata	0	0	0	0
ListFilesystemDir	321	1	322	215
RenamePath	0	1	1	1
CreatePathDirectory	1	0	1	1
TOTAL READ	**18,110**	**17,846**	**18,213**	**18,056**
TOTAL WRITE	**5,800**	**6,100**	**6,100**	**6,000**
Custo ADLS (US$)	**0.05**	**0.05**	**0.05**	**0.05**

Tabela 37 – C6 — Operações ADLS e custo de armazenamento (100GB Silver)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	34,730	34,700	34,670	34,700
AppendFile	13,260	13,260	13,020	13,180
LeaseFile	0	0	0	0
CreatePathFile	970	970	962	967
FlushFile	970	970	954	965
DeleteFile	0	0	0	0
BlobPreflightRequest	0	0	0	0
GetFileProperties	0	0	0	0
GetPathStatus	0	0	0	0
GetFilesystemProperties	9,880	9,870	9,800	9,850
ListBlobs	0	0	0	0
GetBlobServiceProperties	6	0	0	2
GetContainerProperties	0	0	0	0
GetContainerServiceMetadata	0	0	0	0
ListFilesystemDir	802	802	801	802
RenamePath	1	1	0	1
CreatePathDirectory	1	1	1	1
TOTAL READ	**45,436**	**45,426**	**45,293**	**45,385**
TOTAL WRITE	**15,202**	**15,202**	**14,978**	**15,127**
Custo ADLS (US$)	**0.13**	**0.13**	**0.13**	**0.13**

Tabela 38 – C7 — Operações ADLS e custo de armazenamento (10GB Gold)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	3,080	3,080	3,080	3,080
AppendFile	1,350	1,350	1,350	1,350
LeaseFile	0	0	0	0
CreatePathFile	83	83	83	83
FlushFile	83	83	83	83
DeleteFile	0	0	0	0
BlobPreflightRequest	0	0	0	0
GetFileProperties	0	0	0	0
GetPathStatus	0	0	0	0
GetFilesystemProperties	801	800	801	801
ListBlobs	0	0	0	0
GetBlobServiceProperties	0	0	0	0
GetContainerProperties	0	0	0	0
GetContainerServiceMetadata	0	0	0	0
ListFilesystemDir	0	0	0	0
RenamePath	0	0	0	0
CreatePathDirectory	0	0	0	0
TOTAL READ	**3,881**	**3,880**	**3,881**	**3,881**
TOTAL WRITE	**1,516**	**1,516**	**1,516**	**1,516**
Custo ADLS (US$)	**0.01**	**0.01**	**0.01**	**0.01**

Tabela 39 – C8 — Operações ADLS e custo de armazenamento (40GB Gold)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	12,260	12,230	12,260	12,250
AppendFile	5,320	5,050	5,320	5,230
LeaseFile	0	0	0	0
CreatePathFile	311	304	311	309
FlushFile	311	296	311	306
DeleteFile	0	0	0	0
BlobPreflightRequest	0	0	0	0
GetFileProperties	0	0	0	0
GetPathStatus	0	0	0	0
GetFilesystemProperties	2,850	2,790	2,850	2,830
ListBlobs	0	0	0	0
GetBlobServiceProperties	0	0	0	0
GetContainerProperties	0	0	0	0
GetContainerServiceMetadata	0	0	0	0
ListFilesystemDir	0	0	0	0
RenamePath	0	0	0	0
CreatePathDirectory	0	0	0	0
TOTAL READ	**15,110**	**15,020**	**15,110**	**15,080**
TOTAL WRITE	**5,942**	**5,650**	**5,942**	**5,845**
Custo ADLS (US$)	**0.04**	**0.04**	**0.04**	**0.04**

Tabela 40 – C9 — Operações ADLS e custo de armazenamento (100GB Gold)

Operação	**Exec. 1**	**Exec. 2**	**Exec. 3**	**Média**
ReadFile	30,740	30,750	30,760	30,750
AppendFile	13,530	13,520	13,530	13,527
LeaseFile	0	0	0	0
CreatePathFile	830	830	831	830
FlushFile	829	829	831	830
DeleteFile	800	800	800	800
BlobPreflightRequest	804	804	805	804
GetFileProperties	0	0	0	0
GetPathStatus	0	0	0	0
GetFilesystemProperties	7,250	7,250	7,260	7,253
ListBlobs	8	10	8	9
GetBlobServiceProperties	5	5	5	5
GetContainerProperties	3	3	3	3
GetContainerServiceMetadata	1	1	1	1
ListFilesystemDir	0	0	0	0
RenamePath	0	0	0	0
CreatePathDirectory	0	0	0	0
TOTAL READ	**37,999**	**38,009**	**38,029**	**38,012**
TOTAL WRITE	**16,793**	**16,783**	**16,797**	**16,791**
Custo ADLS (US$)	**0.13**	**0.13**	**0.13**	**0.13**