UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Gustavo Batista

Migração de uma arquitetura de dados baseada em sistemas de *Business Intelligence* para uma arquitetura de dados baseada em nuvem.

Florianópolis - SC
2025

Gustavo Batista

Migração de uma arquitetura de dados baseada em sistemas de *Business Intelligence* para uma arquitetura de dados baseada em nuvem.

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof(a). Ricardo José Rabelo, Dr(a).

Florianópolis - SC
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Batista, Gustavo
Migração de uma arquitetura de dados baseada em sistemas de business intelligence para uma arquitetura de dados baseada em nuvem / Gustavo Batista ; orientador, Ricardo José Rabelo, 2025.
104 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2025.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Migração. 3. Dados. 4. Infraestrutura. 5. Nuvem. I. Rabelo, Ricardo José. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Gustavo Batista

Migração de uma arquitetura de dados baseada em sistemas de *Business Intelligence* para uma arquitetura de dados baseada em nuvem.

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 17 de Fevereiro de 2025.

Prof. Marcelo de Lellis Costa de Oliveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof(a). Ricardo José Rabelo, Dr.
Orientador(a)
UFSC/CTC/EAS

Documento assinado digitalmente
Ricardo Jose Rabelo
Data: 09/03/2025 17:43:28-0300
CPF: ***.802.119-**
Verifique as assinaturas em https://v.ufsc.br

Matheus Kjellin, Eng.
Supervisor(a)
Bix Tecnologia

Documento assinado digitalmente
MATHEUS KJELLIN
Data: 12/03/2025 14:04:16-0300
Verifique em https://validar.iti.gov.br

Prof(a). João Gabriel Zago, Dr.
Avaliador(a)
UFSC/CTC/EAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/EAS

Este trabalho é dedicado aos meus pais, meus irmãos,
minha companheira, meus colegas de curso e a todos
que, de alguma forma, fizeram parte dessa jornada.

AGRADECIMENTOS

Inicialmente, gostaria de agradecer aos meus pais, Ilção e Gisela, por toda a estrutura e suporte sob os quais pude trilhar minha vida acadêmica desde 2018. Foi uma jornada longa, desde a busca pela aprovação até a continuidade na graduação, e eles sempre estiveram ao meu lado, fornecendo tudo o que precisei para chegar até este momento. Agradeço também especialmente aos meus irmãos, Juliano e Felipe, que me acompanharam na decisão de cursar engenharia e foram inspirações constantes em minha jornada acadêmica, compartilhando frustrações, conquistas e a convivência que me foi de grande suporte ao longo do percurso. Agradeço à minha namorada, Bianca, que esteve ao meu lado ao longo de todo o meu percurso universitário, compartilhando felicidades e desafios e que, por incontáveis momentos, foi minha fonte de esperança de que as coisas ficariam bem, de que eu era capaz de cruzar todos os obstáculos encontrados e que tinha muito para entregar. Agradeço também aos meus colegas de curso, em especial Leonardo, Maurício, Laura, Juliana, Vinicius, Monique e Paula, que compartilharam comigo esta jornada inesperada, desde os desafios dos trabalhos e provas nas longas madrugadas até as festas e shows, também nas longas madrugadas. Agradeço também aos meus companheiros de banda, Pedro, André, Bernardo e Prof. Ubirajara, esta que surgiu em meio ao ambiente acadêmico e que me despertou novamente a felicidade em fazer música para as pessoas. Agradeço à Bix Tecnologia, em especial ao Felipe Ferraz Leonardo, pela oportunidade de fazer parte de uma empresa referência em sua área e desempenhar um papel importante nas tratativas com clientes e projetos de alta complexidade e desafiadores que me ajudaram a crescer como profissional. Aos mestres que foram os responsáveis pelas disciplinas que tive ao longo da graduação, meu mais profundo obrigado, pois foram vocês os responsáveis por transformar os desafios em conhecimento e o conhecimento em experiência que me levaram à pessoa e ao profissional que sou hoje. Agradeço à Universidade Federal de Santa Catarina, que, infelizmente, vi perder sua estrutura ao longo dos anos em que aqui estive, mas que, mesmo com adversidades orçamentárias, seus colaboradores e alunos nunca deixaram de lutar pelo ensino de qualidade e pela pesquisa que impulsiona nossa região e o país. Por fim, agradeço a mim mesmo por, mesmo duvidando em diversos momentos sobre a escolha que fiz, não ter desistido e chegado até aqui. Inúmeros foram os momentos em que duvidei que seria capaz de finalizar este curso, porém, aqui estou e, portanto, agradeço a mim mesmo por ter cuidado de mim, buscado ajuda quando necessário, ajudado quem precisava e, sobretudo, ter seguido em frente todos os dias.

“Ninguém educa ninguém,
ninguém educa a si mesmo,
os homens se educam entre si,
mediatizados pelo mundo.”
(Freire, 1987)

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 17 de Fevereiro de 2025.

Na condição de representante da BIX TECNOLOGIA na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

gov.br
Documento assinado digitalmente
MATHEUS KJELLIN
Data: 11/03/2025 10:20:05-0300
Verifique em https://validar.iti.gov.br

Matheus Kjellin
BIX TECNOLOGIA

RESUMO

O avanço da tecnologia da informação, aliada à digitalização dos processos industriais e empresariais e ao desenvolvimento da indústria 4.0 gerou uma realidade em que os processos de uma empresa tornaram-se geradores de informações. Essas informações são manipuladas e registradas pelos sistemas em forma de dados que, quando bem tratados e utilizados, podem resultar em vantagens para as empresas, podendo ser utilizadas em análises de operação com informações úteis para direcionar as decisões tomadas, como matéria-prima para desenvolver produtos voltados a dados e ainda como fontes para alimentar modelos de Inteligência Artificial (IA) capazes de realizar análises prescritivas, entre outras tantas possibilidades. Para que os dados sejam de fato utilizados de maneira eficiente e confiável, é imprescindível que os negócios tenham equipes e processos voltados para a gestão e desenvolvimento de infraestruturas de dados. É comum que se encontre em empresas com forte presença da tecnologia da informação equipes especializadas para desenvolver serviços com dados, compostas por analistas, engenheiros e cientistas de dados. A prática mais comum encontrada nas companhias é uma equipe formada apenas por analistas de dados utilizando ferramentas de *Business Intelligence (BI)* para desenvolver análises simples e com limitações quanto ao volume de dados processados por conta do poder de processamento de tais ferramentas. Neste contexto, o presente projeto foi desenvolvido pela empresa de consultoria Bix Tecnologia para uma empresa cliente a fim de expandir a infraestrutura de dados da mesma. O projeto consiste em uma migração de uma arquitetura de dados que utiliza somente sistemas de *BI* do ecossistema Microsoft para uma nova arquitetura voltada para serviços de nuvem. A utilização de serviços de nuvem apresenta vantagens do ponto de vista de poder de processamento, confiabilidade, segurança em todas as camadas de uma arquitetura de dados. Utilizando-se das mais modernas técnicas de processamento de dados, uma migração completa do conjunto de relatórios de *BI* já utilizados pela empresa cliente foi realizada, implementando novamente todas as tabelas utilizadas pela empresa em uma arquitetura utilizando os serviços de nuvem fornecidos pela Microsoft Azure. Foram aplicadas as melhores técnicas de Engenharia de dados, utilizando serviços de nuvem para armazenamento, processamento, governança e segurança de dados, a fim de fornecer para a empresa cliente uma infraestrutura capaz de suprir suas demandas analíticas crescentes de maneira mais ágil e robusta, além de desenvolver um ecossistema capaz de habilitar o uso de tecnologias modernas voltadas ao uso de IA. Como resultados, uma arquitetura completa de dados foi implementada contemplando as necessidades da empresa e apresentando ganhos relacionados à performance, segurança e maturidade de dados.

Palavras-chave: Migração. Dados. Arquitetura. Infraestrutura. Nuvem.

ABSTRACT

The advancement of information technology, combined with the digitization of industrial and business processes and the development of Industry 4.0, has created a reality where a company's processes have become generators of information. This information is handled and recorded by systems in the form of data which, when properly processed and utilized, can result in advantages for companies. These advantages include supporting operational analyses with useful information to guide decision-making, serving as raw material to develop data-driven products, and acting as sources to feed Artificial Intelligence (AI) models capable of performing prescriptive analyses, among many other possibilities. For data to be effectively and reliably utilized, it is essential that businesses have teams and processes dedicated to the management and development of data infrastructures. It is common for companies with a strong presence in information technology to have specialized teams for developing data services, composed of analysts, engineers, and data scientists. However, the most common practice in companies is to have a team made up solely of data analysts using Business Intelligence (BI) tools to develop simple analyses, often limited by the data volume that can be processed due to the computational power of such tools. In this context, the present project was developed by the consultancy firm Bix Tecnologia for a client company to expand its data infrastructure. The project involves the migration from a data architecture that exclusively uses BI systems from the Microsoft ecosystem to a new architecture focused on cloud services. The use of cloud services offers advantages in terms of processing power, reliability, and security across all layers of a data architecture. By employing the latest data processing techniques, a complete migration of the set of BI reports already used by the client company was carried out, implementing again all the tables utilized by the company within an architecture leveraging cloud services provided by Microsoft Azure. The best data engineering techniques were applied, using cloud services for data storage, processing, governance, and security. The goal was to provide the client company with an infrastructure capable of meeting its growing analytical demands in a faster and more robust manner, while also developing an ecosystem that enables the use of modern AI-focused technologies. As a result, a complete data architecture was implemented, addressing the company's needs and delivering improvements in performance, security, and data maturity.

Keywords: Migration. Data. Architecture. Infrastructure. Cloud.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANSI	American National Standards Institute
API	Application Programing Interface
ASF	Apache Software Foundation
BI	Business Intelligence
CSP	Cloud Service Provider
DW	Data Warehouse
ETL	Extract, Transform, Load
GFS	Google File System
HD	Hard Drive
HDFS	Hadoop Distribuited File System
IA	Inteligência Artificial
IOT	Internet of Things
IR	Integration Runtimes
ISO	International Standards Organization
JVM	Java Virtual Machine
ML	Machine Learning
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PAYG	Pay as You Go
RDD	Resilient Distribuited Datasets
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SCD	Slowly Changing Dimensions
SOAP	Simple Object Access Protocol
TI	Tecnologia da Informação

SUMÁRIO

1 INTRODUÇÃO

Este relatório tem como objetivo apresentar o desenvolvimento do Projeto de Fim de Curso do graduando Gustavo Batista no contexto da disciplina DAS5511 - Projeto de Fim de Curso. O projeto foi desenvolvido no setor de Engenharia de Dados da empresa BIX TECNOLOGIA durante o primeiro e segundo semestres do ano de 2024, sob a supervisão local de Matheus Kjelin e supervisão acadêmica de Ricardo José Rabelo, com o objetivo de desenvolver uma migração completa de uma infraestrutura de dados voltada a sistemas de *Business Intelligence (BI)* para uma nova arquitetura baseada em sistemas de nuvem e *Data Lakehouse*.

A BIX TECNOLOGIA é uma empresa fundada na cidade de Florianópolis - SC no ano de 2014, com o objeto de fornecer serviços de consultoria nas áreas de tecnologia voltadas a dados e desenvolvimento de software. Através de uma estrutura organizacional dividida em times com especialidades distintas, a empresa tem como objetivo fornecer serviços de consultoria e desenvolvimento tecnológico a fim de auxiliar seus clientes, sendo eles nacionais ou internacionais. A empresa conta com times de BI focados no desenvolvimento de sistemas de relatórios e análises descritivas e prescritivas, time de Ciência de Dados focado em desenvolver soluções utilizando Inteligência Artificial (IA) nos mais diversos contextos, além do time de Engenharia de Software, subdividido em Desenvolvimento de Software e Engenharia de Dados, sendo estes focados em desenvolver infraestruturas para lidar com volumes de dados diversos e de origens distintas, trabalhando no desenvolvimento de *Data Warehouses* ou *Data Lakehouses* e aqueles em desenvolvimento de softwares capazes de suprir as necessidades das empresas clientes conforme a realidade da mesma.

Ao longo dos seus anos de operação, a empresa atraiu clientes por todo o país e internacionalmente através da qualidade dos projetos entregues e dos times comerciais dedicados a apresentar e divulgar os serviços oferecidos pela companhia. Em sua carta de clientes, constam grandes varejistas do ramo da moda, indústrias metalmecânicas, indústrias agrícolas, mineradoras, empresas de tecnologia, entre outros. No contexto deste projeto de fim de curso, o cliente atendido foi uma grande salina nacional, com expressiva presença em todo o país e internacionalmente, com suas operações centralizadas no Sudeste e Nordeste do país.

1.1 CONTEXTUALIZAÇÃO DO PROJETO

A Empresa Cliente atendida pela BIX TECNOLOGIA neste projeto possui suas operações com forte presença de tecnologia, utilizando softwares específicos em cada uma das etapas dos processos fabris e administrativos. A fim de encontrar maior clareza sobre suas operações e buscar ideias que pudessem direcionar a tomada de decisões, a empresa possui uma estrutura analítica baseada em sistemas de BI com

relatórios visuais para cada um dos setores, auxiliando os operadores no acompanhamento dos trabalhos, os gestores no acompanhamento da produção e os diretores nas tomadas de decisão.

Sistemas de BI são muito úteis quando o volume de dados não é grande, podendo atender análises descritivas com considerável eficiência. Porém, quando o volume de dados cresce, sendo uma tendência natural de processos com a forte presença de elementos de Tecnologia da Informação que guardam as informações das operações realizadas, cresce a demanda por capacidade computacional para processar a crescente quantidade de informações. Nesta realidade, a Empresa Cliente apresenta um volume crescente de dados utilizados em suas análises sem que os sistemas de BI atuais sejam capazes de lidar com o novo volume, o que exige alternativas para lidar com o novo cenário. Uma das possibilidades seria o investimento em licenças de maior valor financeiro do *software* de BI utilizado atualmente, elevando a capacidade de processamento da ferramenta, atendendo à demanda da Empresa Cliente. Outra possibilidade seria o desenvolvimento de uma infraestrutura de dados nova para a empresa, não mais utilizando ferramentas de *BI*, mas baseada em serviços de computação em nuvem. Ao estudar as possibilidades, concluiu-se que a alternativa de uma nova arquitetura seria a com maiores benefícios para a Empresa Cliente, uma vez que a mesma almeja dar um novo passo em sua estrutura de dados que apresente não somente a robustez e confiabilidade que uma arquitetura em nuvem possui, como também possibilite o uso de modelos de Machine Learning (ML) com seus dados a fim de expandir as possibilidades de negócio.

Portanto, um projeto de migração completa da infraestrutura de dados atual da Empresa Cliente foi realizado. Foram migradas todas as estruturas de tabelas que alimentam os relatórios atuais da empresa, a fim de manter a estrutura analítica atual da empresa. Porém, a nova arquitetura apresenta um conjunto de ferramentas capazes de fornecer à Empresa Cliente uma nova experiência com seus dados, tanto do ponto de vista de robustez e segurança quanto da expansão das possibilidades de desenvolver novas análises a partir da infraestrutura fornecida.

1.2 OBJETIVOS GERAIS

Como objetivo geral, espera-se que, ao final do projeto, a mesma estrutura analítica de relatórios atual da empresa seja mantida, contudo utilizando os dados alocados na nova arquitetura. A nova arquitetura deve ser capaz de buscar os dados em diferentes origens, sendo elas bancos de dados relacionais, planilhas externas, *API's*, salvando-os em um serviço de armazenamento de dados em nuvem. Também deve-se utilizar uma ferramenta para realizar o processamento dos dados de acordo com a metodologia proposta, realizando-se todas as transformações já presentes na arquitetura atual de tabelas utilizadas pela Empresa Cliente, replicando fielmente todos

os tratamentos e alterações nos dados para o uso final. Para realizar toda a ingestão dos dados brutos das origens e orquestrar os fluxos de dados na arquitetura, deve-se utilizar um serviço de nuvem disponível para essas tarefas, que apresente segurança e robustez, capaz de garantir a atualização diária dos dados na nova infraestrutura.

Com a nova arquitetura implementada, objetiva-se oferecer ao cliente uma nova filosofia de tecnologias utilizadas em seus processos, mais robusta e confiável, com etapas de segurança e possibilidades além do oferecido pelos softwares de BI já utilizados. Além disso, espera-se que, ao fim do projeto, a Empresa Cliente possua um conjunto de ferramentas, métodos e processos capazes de fornecer novas aplicações para os dados processados, ampliando suas capacidades analíticas para além de análises descritivas. Espera-se que a nova infraestrutura apoie as iniciativas futuras da Empresa Cliente no uso dos dados para alimentar modelos de Inteligência Artificial capazes de prever demandas, tempos de processos e outras variáveis importantes para a companhia.

A fim de cumprir com os objetivos do projeto, os passos a seguir foram seguidos dentro do cronograma do projeto:

- **Ambientação e Acessos** - Gestão dos acessos ao serviço de nuvem a ser implementado com contas específicas da Empresa Cliente;
- **Estruturação da Arquiterura no Serviço de Nuvem** - Configuração dos serviços de nuvem utilizados no projeto, bem como a interconexão entre os mesmos;
- **Implementação dos *Pipelines*** - Implementação dos *pipelines* de dados, desde a criação da ingestão dos dados até o processamento e disponibilização;
- **Ajustes** - Ajustes nas tabelas implementadas e entregues;
- **Otimizadores** - Implementação dos otimizadores para melhor uso dos recursos;
- **Ajustes Finais** - Ajustes finais na nova arquitetura.

1.3 EQUIPE ENVOLVIDA

Para o desenvolvimento do projeto, foram necessários alguns elementos tanto da parte da BIX TECNOLOGIA quanto da parte da Empresa Cliente, a fim de realizar o gerenciamento e a execução. Da parte da BIX TECNOLOGIA, o time foi composto por:

- **Líder Técnico** - Responsável pelo gerenciamento das tarefas e relacionamento com a Empresa Cliente;
- **Engenheiro de Dados (O autor deste documento)** - Responsável pela execução da parte técnica referente a nova infraestrutura de dados;

- **Analista de *Business Intelligence*** - Responsável pela execução da parte técnica referente a migração dos relatórios atuais para a nova infraestrutura de dados.

Já da parte da Empresa Cliente, o time foi composto por:

- **Gerente de TI** - Responsável pelo departamento de tecnologia da informação da Empresa Cliente e principal cliente interno do projeto;
- **Analista de TI Sênior** - Responsável pelo auxílio a equipe da BIX TECNOLOGIA a fim de contextualizar a infraestrutura atual e demais necessidades.

1.4 ESTRUTURA DO DOCUMENTO

O presente documento será dividido em capítulos para apresentar os contextos e motivações em que o projeto foi desenvolvido, bem como as premissas e justificativas de uso das tecnologias utilizadas e, por fim, o desenvolvimento do projeto. Os capítulos serão:

- **Capítulo 1 - Introdução**
  Apresentados o contexto geral do projeto, as motivações e o objetivo de desenvolvimento do mesmo.
- **Capítulo 2 - Empresas envolvidas no projeto**
  Apresentados o contexto das empresas envolvidas no projeto, as estruturas que as compõem e os detalhes de sigilo do projeto
- **Capítulo 3 - Fundamentação teórica**
  Apresentados os principais conceitos teóricos que apoiaram o desenvolvimento do projeto, bem como técnicas e tecnologias exploradas.
- **Capítulo 4 -Descrição do problema e Requisitos do projeto**
  Apresentados detalhes do problema atacado, bem como os requisitos funcionais e não funcionais do projeto, a fim de elucidar como a proposta de solução resolveu o problema.
- **Capítulo 5 - Solução Proposta e Metodologia utilizada**
  Apresentadas a solução proposta para resolver o problema descrito na seção 4, bem como a metodologia utilizada para execução do projeto.
- **Capítulo 6 - Desenvolvimento e Análise dos resultados obtidos**
  Apresentado o desenvolvimento dos requisitos do capítulo 4, passando pelas etapas e estruturas implementadas para resolução do problema, bem como a análise dos resultados obtidos com a nova estrutura.

- **Capítulo 7 - Conclusão**
  Apresentada a conclusão do projeto desenvolvido resumindo todo o processo de desenvolvimento desde a elucidação do problema até a implementação da solução e indicações para trabalhos futuros..

2 EMPRESAS ENVOLVIDAS

2.1 BIX TECNOLOGIA

A BIX TECNOLOGIA é uma empresa fundada no ano de 2014 na cidade de Florianópolis-SC, um dos polos tecnológicos do país. Voltada inicialmente para prestar consultorias em tecnologia, especialmente na área de dados, a empresa foi capaz de encontrar um nicho de mercado de análise de dados ainda não explorado com tanta ênfase à época. Ao longo de seus mais de 10 anos de operação, a empresa foi capaz de chegar a empresas de diversos tamanhos, evoluindo de indústrias e companhias catarinenses até clientes com expressiva operação nacional e, futuramente, chegando a clientes a nível internacional. A internacionalização da empresa levou à fundação do seu braço internacional, a BIX TECH CORP., localizada na Flórida-EUA e atendendo clientes do mercado americano, europeu e asiático.

Em seus 10 anos de constante evolução, a empresa foi capaz de estabelecer sua postura no mercado como referência em desenvolvimento de produtos e soluções voltadas a dados. Inicialmente, a companhia teve seu foco voltado para os sistemas de BI, fornecendo a seus clientes uma consultoria completa desde o entendimento da natureza do negócio, das áreas, processos e demandas, aliando esses conhecimentos à tecnologia para trazer mais clareza, direcionamento e assertividade nas tomadas de decisão, auxiliando na otimização dos negócios. Em seu crescimento, a BIX TECNOLOGIA passou a atender clientes com nível de operação nacional com um patamar de volume de dados elevado, sendo necessária a evolução dos projetos para atender as demandas voltadas a *Big Data* (grandes volumes de dados). Com esse movimento, a empresa criou uma área específica para lidar com o processamento dos dados, a Engenharia de Dados, sendo responsável por garantir o fluxo correto de grandes volumes de dados, utilizando ferramentas de mercado e tecnologias voltadas para este fim e oferecendo projetos com magnitudes cada vez maiores. A partir do avanço do uso da IA em processos empresariais, surgiu também a demanda dos clientes da BIX TECNOLOGIA por soluções voltadas às novas tecnologias. Portanto, a área de Ciência de Dados foi criada, voltada especificamente para projetos envolvendo a aplicação de modelos de Aprendizado de Máquina e Redes Neurais, desde projetos de previsão de vendas ou de resultados de operações até a aplicação de Visão Computacional para identificação de objetos em uma linha de produção. A empresa conta também com uma área de Desenvolvimento de Software, oferecendo o serviço de criação de softwares específicos para diferentes áreas de negócio e, em diversos projetos, integrando-se com as áreas de BI, Engenharia ou Ciência de Dados a fim de entregar uma solução completa e robusta a seus clientes.

A BIX TECNOLOGIA conta hoje com uma expressiva presença no mercado conquistada pelas lideranças e pelas pessoas que fazem ou fizeram parte de seu

quadro ao longo de seus mais de 10 anos de existência, sendo vencedora em prêmios voltados à inovação em nível estadual e nacional. Atendendo clientes de porte nacional e internacional, a empresa apresenta maturidade e propriedade no desenvolvimento de diferentes soluções no ramo da tecnologia, especialmente em soluções voltadas para Análise, Engenharia e Ciência de Dados, bem como no Desenvolvimento de Software. A seguir, será apresentada a estrutura interna da empresa.

2.1.1 Estrutura Interna

Como já apresentado, a BIX TECNOLOGIA conta com áreas específicas com o objetivo de estratificar suas especialidades a fim de entregar soluções completas para seus clientes. De maneira geral, a empresa se divide em 3 grandes áreas técnicas, sendo elas:

- **BI** - Área responsável pelo desenvolvimento de projetos voltados à análise de dados em relatórios visuais interativos. Utilizando ferramentas de *BI* como Power BI, Qlik ou Tableau o time é capaz de desenvolver relatórios de análises descritivas com o objetivo de trazer mais clareza aos clientes e explorar a natureza de seu negócio em diferentes níveis com o apoio de dados reais para embasar as decisões.
- ***Software Engineering*** - Área subdividida em outras duas sendo: *Software Development* e *Data Engineering*. O time de *Software Development* é o responsável pelo desenvolvimento de softwares específicos para os clientes de acordo com a necessidade, utilizando o conhecimento interno em diferentes etapas do desenvolvimento como *Back-End*, *Front-End* e Experiência de Usuário para desenvolver soluções que atendam as demandas. Já o time de *Data Engineering* é o responsável pelo desenvolvimento de soluções para migração, ingestão, tratamento e processamento e dados em volumes diversos. O time atua desenvolvendo **pipelines** de ETL completos, soluções de processamento de *Big Data* entre outras soluções para lidar com o tratamento seguro e gerenciado dos dados.
- ***Data Science*** - Área responsável pelo desenvolvimento de projetos utilizando diferentes níveis de Inteligência Artificial. É o time especializado em aplicar diferentes modelos de acordo com a necessidade do cliente, podendo desenvolver projetos de previsão de vendas a partir de dados históricos com modelos de regressão, estimativa de produção a partir de séries temporais até aplicação de visão computacional para analisar linhas de produção de maneira automatizada.

As áreas são divididas nos chamados *Chapters* (Capítulos em português), que possuem uma base de conhecimento compartilhada entre seus membros para desenvolvimento dos projetos da empresa. Também são produzidos *Squads* (Esquadrão em

português) quando são necessárias diferentes expertises para trabalhar em um projeto, como, por exemplo, um *Squad* com membros dos times de *Software Engineering* e *Data Sciente* para entregar uma solução que atenda o cliente de maneira mais abrangente. Esta estrutura baseia-se nos modelos de desenvolvimento ágil do *Spotify*, onde cada equipe define a melhor forma de gerir o trabalho com as metodologias ágeis, gerando autonomia e acelerando o processo de desenvolvimento (CRUTH, 2022).

Além dos *Chapters* técnicos, a BIX TECNOLOGIA conta também com áreas focadas em atividades comerciais e relacionamento com clientes. São os *Chapters* de Comercial, Marketing, Administrativo-Financeiro e Pessoas, desempenhando papéis de contato com clientes para prospecção e venda de projetos, divulgação dos trabalhos, gestão financeira e relacionamento pessoal, bem como novas contratações.

2.2 EMPRESA CONTRATANTE

A BIX TECNOLOGIA em seus anos de operação foi capaz de atrair médias e grandes empresas para a sua carta de clientes. Grandes indústrias já foram atendidas pela BIX TECNOLOGIA, como grandes companhias do setor de telecomunicações, setor metalúrgico, agronegócio, entre outros. No presente Projeto de Fim de Curso (PFC), o cliente atendido foi uma grande produtora de sal brasileira, com expressiva operação em todo o território nacional, contando com unidades logísticas distribuídas pelas regiões do país. Além do mercado nacional, a empresa conta com clientes no exterior, com produtos sendo comercializados em países da América do Sul, América do Norte, África e Europa. A empresa possui uma infraestrutura industrial robusta e de grandes proporções para produzir e atender a todos os clientes com produtos voltados a diferentes finalidades, desde o uso industrial até o consumo humano. A Empresa Cliente também possui uma estrutura de logística respeitável, sendo capaz de distribuir a produção por terra a nível nacional e por mar a nível internacional.

2.2.1 Acordo de Não-Divulgação

Por questões de segurança institucional, todas as informações relacionadas a detalhes da empresa atendida neste PFC foram suprimidas para evitar possíveis transtornos e problemas de vazamento de informações importantes. Para fins de documentação, as informações relacionadas à empresa contidas neste relatório serão anonimizadas ou alteradas para não prejudicar a informação real que é de completo domínio da própria, sendo esta ao longo deste documento referida como a "Empresa Cliente". Um **acordo de não divulgação** foi assinado pelo autor deste documento a fim de garantir que as informações a respeito do projeto sejam guardadas somente pelos membros do mesmo e pela Empresa Cliente.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar os principais métodos, conceitos e técnicas utilizados na implementação do projeto, especialmente utilizando como base o conhecimento disponível na literatura para aplicação em um projeto real. Serão explicitados os principais conceitos de bancos de dados, técnicas de processamento distribuído, computação em nuvem, infraestrutura de dados e *API's*, todas as técnicas ou conceitos utilizados para a execução do projeto.

3.0.1 Dado

Ao longo de um dia, diversos fatos acontecem no mundo real e têm impacto direto em diferentes esferas da sociedade. Seja na área de saúde, negócios, indústria, justiça, relações interpessoais, todas essas interações geram fatos e estes fatos direcionam o rumo de nossa realidade. Esses fatos podem ser ou não enumerados ou categorizados em estruturas de informação que são registradas, formando assim o que se conhece como Dado.

De acordo com o Dicionário de Cambridge, um dado é uma informação, especificamente um fato ou número, coletado e utilizado para auxiliar a tomada de decisões, ou então uma informação em formato eletrônico, armazenada e utilizada por sistemas computadorizados (CAMBRIGE, 2024).

Portanto, as informações geradas em um processo do mundo real podem ser registradas ou armazenadas para que sejam utilizadas nas mais diversas aplicações possíveis. Seja em sistemas de manufatura para registrar a produção realizada em um dia, registros de interações em redes sociais, na coleta de informações de processos jurídicos, entre outros, informações do mundo real, quando convertidas para dados e computadorizadas, cumprem um papel fundamental no suporte eficiente a esses processos.

Para que a informação registrada em forma de Dado seja utilizada de maneira efetiva, faz-se necessário o desenvolvimento de estruturas que permitam o seu uso de maneira otimizada. Na próxima seção, serão abordados os fundamentos e aplicações dos chamados Bancos de Dados, estruturas que permitiram o uso efetivo dos Dados.

3.1 BANCOS DE DADOS

A digitalização dos processos em diferentes setores possui em seu cerne tecnologias e métodos desenvolvidos no decorrer das últimas décadas com o objetivo de cada vez mais tornar os processos eficientes e confiáveis, utilizando computadores e sistemas de tecnologia da informação como ferramenta central das atividades realizadas. Entre as tecnologias desenvolvidas e que seguem sendo um pilar para os

sistemas utilizados no processo de digitalização dos processos estão os Bancos de Dados.

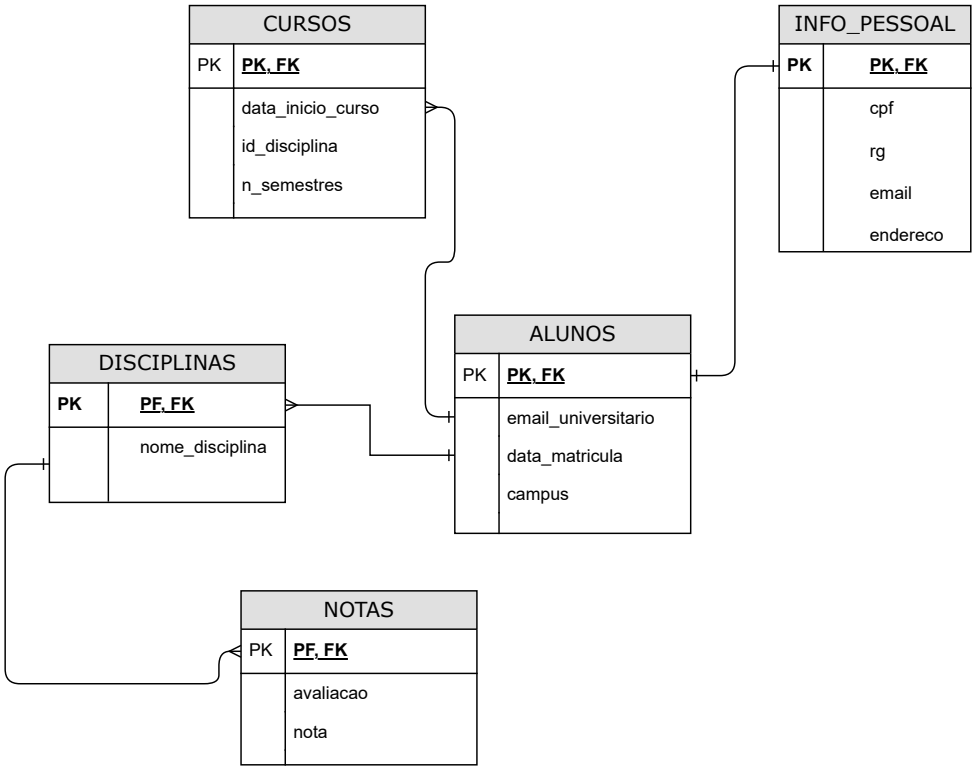
Bancos de Dados podem ser descritos como uma coleção de informações relacionadas, como, por exemplo, nomes, números de telefone ou endereços. Os dados podem ser persistidos em diferentes sistemas, de acordo com a natureza da aplicação do banco, como celulares, computadores pessoais ou servidores, resultando em um Banco de Dados (ELMASRI; NAVATHE, 2015). De forma geral, qualquer conjunto de informações poderia recair sob a definição de Banco de Dados, porém, algumas propriedades são utilizadas para descrevê-los como:

- Bancos de Dados Representam alguma característica do mundo real, onde mudanças nestas características acarretam mudanças no Banco de Dados;
- Um Banco de Dados é uma coleção de dados logicamente coerente com um significado definido. Um conjunto de informações aleatórias não pode ser caracterizada como um Banco de Dados.
- Bancos de Dados são projetados e populados com um objetivo específico. O mesmo possui um grupo definido de usuários e são aplicados a sistemas préestabelecidos os quais os usuários fazem uso.

Bancos de Dados possuem uma origem ou um conjunto de origens para a informação gerada, uma interação com o mundo real que altera as informações e um destino final para estas informações (ELMASRI; NAVATHE, 2015), podendo ser aplicados em diferentes contextos e em sistemas diversos para registrar as informações inseridas, alteradas ou deletadas nos mesmos.

Usualmente, Bancos de Dados são associados a tabelas ou conjuntos de tabelas. Os dados são organizados no que chama-se de estrutura tabular que é alocada em discos rígidos nos espaços de memória de forma a associar os registros entre si e formar a tabela final. Um exemplo simples de aplicação de Banco de Dados são os registros dos estudantes de uma universidade, onde ficam alocados dados pessoais do estudante como Nome, CPF, e-mail, telefone, endereço, assim como dados do curso em que o estudante está alocado, departamento, disciplinas cursadas, avaliações realizadas, resultados obtidos, entre outras informações pertinentes ao universo em que os dados aplicam-se. As informações podem ser alocadas nos bancos de dados conforme o diagrama apresentado na Figura 1 e alteradas por sistemas externos conforme a necessidade.

Figura 1 – Diagrama Entidade Relacionamento de um Banco de Dados Relacional.



Fonte: Autor.

O exemplo apresentado é somente uma das possíveis estruturas de Bancos de Dados utilizadas nas aplicações atuais. Porém, existem diferentes formatos desenvolvidos para as diferentes aplicações e necessidades que serão apresentados na seção seguinte.

3.1.1 Bancos de Dados Relacionais VS Não Somente Relacionais

Diferentes abordagens para o desenvolvimento de Bancos de Dados foram implementadas ao longo da evolução destes sistemas. Inicialmente, os Bancos de Dados foram desenvolvidos para o armazenamento de dados em formatos puramente textuais ou numéricos, o que se entende como os Bancos de Dados Tradicionais (ELMASRI; NAVATHE, 2015). Porém, com as novas demandas dos sistemas mais modernos, como processamento de imagens e vídeos, novas abordagens foram utilizadas para armazenar esses formatos de dados, levando ao surgimento de uma forma mais moderna de armazenamento.

Essencialmente, a divisão entre as diferentes formas de Bancos de Dados pode ser dividida entre os Bancos de Dados Relacionais (SQL) e Não Somente Relacionais (NoSQL). Desses dois formatos, entende-se os Bancos de Dados Relacionais como

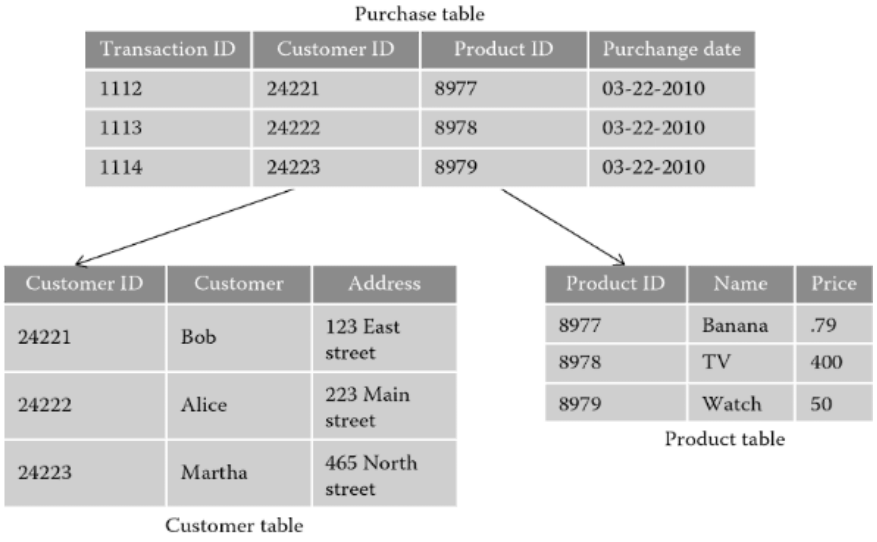
a forma tradicional e os Bancos de Dados Não Somente Relacionais (do inglês, Not Only SQL) como a forma moderna.

Bancos de Dados Relacionais

Os Bancos de Dados Relacionais foram introduzidos pela *IBM Research* por volta da década de 1970, com o artigo publicado pelo matemático e cientista da computação Edgar Frank Codd (CODD, 1970). Codd apoiou-se nos conceitos de matemática relacional, que assemelha-se de certa forma a uma estrutura tabular, para desenvolver um sistema lógico de organização de informações conectadas entre si (ELMASRI; NAVATHE, 2015). Por conta da natureza tabular da matemática relacional, os Bancos de Dados Relacionais podem ser entendidos como uma tabela ou conjunto de tabelas onde as linhas correspondem a fatos que acontecem no mundo real, como os registros dos estudantes na tabela ALUNOS na Figura 1. Já as colunas correspondem às características desses fatos, como as informações de data de matrícula ou campus em que o aluno está alocado.

Os Bancos de Dados Relacionais possuem estruturas bem definidas para cada uma das entidades (Geralmente as tabelas) e o relacionamento entre estas entidades. Conforme observa-se no exemplo da Figura 2, três tabelas fazem parte do Banco de Dados, cada uma com um fato do processo de vendas exemplificado (Compra, Cliente e Produto).

Figura 2 – Tabelas de um Banco Relacional de Vendas.



Fonte: (SAMSON; LU; XU, 2016).

A fim de que os Bancos de Dados Relacionais apresentem uma eficiente forma de lidar com as operações realizadas com seus dados, é necessário o uso de supervisores que garantam as operações corretas para evitar erros. Esses supervisores

são os chamados Sistemas de Gerenciamento de Bancos de Dados, ou SGBDs. Os SGBDs são sistemas computacionais que auxiliam na criação e gerenciamento de Bancos de Dados, auxiliando nas tarefas de definição das tabelas, manipulação dos dados e compartilhamento entre diferentes sistemas (ELMASRI; NAVATHE, 2015).

Além do controle da criação e manutenção do Banco de Dados, os SGBDs apresentam uma característica essencial para a longa aplicação dos Bancos de Dados Relacionais até os dias de hoje: as chamadas Propriedades ACID. ACID é o acrônimo para as palavras Atomicidade, Consistência, Isolamento e Durabilidade, todas as propriedades das operações realizadas nos bancos que garantem a efetividade de sua aplicação. Cada propriedade é descrita como:

- **Atomicidade** - Uma transação dos dados é uma unidade atomica de processamento, deve ser realizada de maneira integral ou então não realizada de maneira alguma;
- **Consistencia** - Uma transação dos dados deve ser realizada de maneira consistente sem a interferência de outras transações, levando o Banco de Dados de um estado a outro de maneira completa;
- **Isolamento** - Uma transação deve ocorrer de maneira isolada de outras transações, não sofrendo com as alterações realizadas por diferentes operações;
- **Durabilidade** - Uma transação realizada por completo deve persistir no Banco de Dados, não podendo ser perdida por nenhuma falha no banco.

Bancos de Dados Não Somente Relacionais

Como visto anteriormente, os Bancos de Dados Relacionais possuem um conjunto de características que tornam o seu uso extremamente confiável em muitos sistemas. Porém, em se tratando da evolução dos tipos de dados armazenados pelos Bancos de Dados, os Bancos de Dados Relacionais acabaram perdendo parte de sua efetividade quando as aplicações deixaram de lidar com dados estruturados (formato de tabela com números e textos) e passaram a estar em forma de dados não estruturados (imagens, áudios, vídeos) (ELMASRI; NAVATHE, 2015).

Além dos novos formatos de dados, o volume de informações registradas cresceu de maneira significativa ao longo do processo de digitalização das operações em diferentes setores, o que exigiu a criação de novos métodos para lidar com dados em larga escala. Nesse contexto, surgiram os Bancos de Dados Não Relacionais, capazes de realizar o processamento de grandes volumes de dados de maneira eficiente e otimizada, além de garantir o armazenamento de dados em formatos utilizados pelos sistemas mais modernos.

Os Bancos de Dados Não Somente Relacionais surgiram para atender necessidades advindas das evoluções dos dados armazenados pelos sistemas e pela grande proporção que estes sistemas têm tomado. Alguns casos podem ser elencados como exemplos de novas necessidades dos Bancos de Dados modernos, como o armazenamento de dados de redes sociais como Facebook ou Twitter, ou ainda serviços de *e-mail*. Em suas especificações, os dados registrados nessa categoria de sistemas não necessitam de uma estrutura bem definida, além de não necessitarem de toda a camada de controle de concorrência e garantia de transações, como é o caso dos Bancos de Dados Relacionais.

Algumas características fazem dos Bancos de Dados Não Somente Relacionais ferramentas poderosas para atender às necessidades dos processos atuais. Entre as características mais importantes, pode-se elencar:

- **Escalabilidade** - Os Bancos de Dados Não Somente Relacionais geralmente possuem escalabilidade horizontal, onde quando maior o número de dados, em mais nós os dados são replicados ao invés de aumentar o poder de processamento e de armazenamento;
- **Replicação** - Os dados são replicados em múltiplos nós para em caso de uma falha, o dado esteja disponível independentemente do erro;
- **Fragmentação de Arquivos** - Os dados são fragmentados em diferentes arquivos e em diferentes nós, distribuindo o balanço de carga de trabalho computacional;
- **Não Necessitam de Esquema** - Os Bancos de Dados Não Somente Relacionais podem apresentar estruturas sem uma estrutura de dados definida;
- **Versionamento** - Bancos de Dados Não Somente Relacionais podem armazenar múltiplas versões dos dados com datas para definir o prazo de validade das informações armazenadas, garantindo maior segurança para possíveis perdas de dados.

Conceitos de Bancos de Dados Não Somente Relacionais podem ser explorados em infraestruturas analíticas modernas, permitindo o processamento de grandes volumes de dados analíticos de forma eficiente. Adiante, nesta seção, serão apresentados estes conceitos.

3.1.1.1 SQL

Como apresentado anteriormente, os Bancos de Dados Relacionais possuem em sua implementação uma camada de controle robusta, com o objetivo de garantir a efetividade das transações realizadas no banco. Também atuam como gerenciadores

das estruturas de tabelas definidas em um banco de dados, sendo a camada de criação, alteração, deleção, entre outras operações com as tabelas e os campos (colunas) que as compõem. A fim de que estas operações sejam realizadas pelos usuários dos SGBDs, é importante que exista um meio através do qual os administradores dos Bancos de Dados realizem essas ações. Para este fim, uma linguagem foi criada a fim de que as abstrações escritas pelos usuários sejam traduzidas pelos SGBDs em atividades práticas no banco; esta linguagem é a Structured Query Language, ou SQL.

O SQL surgiu como uma interface para um Banco de Dados Relacional experimental desenvolvido pela *IBM Research* chamado *System R*. Por conta de sua facilidade de uso e padronização para gerenciar diferentes implementações de Bancos de Dados Relacionais, o SQL teve um papel fundamental na migração de antigos sistemas de bancos de dados para os bancos relacionais (ELMASRI; NAVATHE, 2015). Um padrão para a linguagem SQL foi desenvolvido pela *American National Standards Institute (ANSI)* e a *International Standards Organization (ISO)* a fim de garantir que a linguagem seja desenvolvida seguindo uma estrutura bem definida e padronizada (ISO, 2023).

Através do SQL, é possível realizar uma infinidade de operações com os Bancos de Dados Relacionais, desde a criação de tabelas especificando colunas, tipos e outras características, como permissão de acesso, até realizar manipulações e buscas nos dados que podem oferecer informações importantes a respeito da natureza de aplicação dos mesmos. A linguagem pode ser dividida em subconjuntos de operações, de acordo com a natureza da ação realizada no banco. Os subconjuntos mais utilizados podem ser divididos da seguinte forma:

- **DML (Linguagem de Manipulação de Dados)** - Utilizada para realizar manipulações nos dados de um banco de dados. Exemplos de comandos:
  - INSERT - Inserção de dados
  - UPDATE - Atualização de dados
  - DELETE - Deleção de dados
- **DDL (Linguagem de Definição de Dados)** - Utilizada para criação e alteração de tabelas de um banco de dados. Exemplos de comandos:
  - CREATE - Criação de tabela
  - ALTER - Alteração de tabela
  - DROP - Deleção de tabela
- **DQL (Linguagem de Consulta de Dados)** - Utilizada para realizar consultas
  - SELECT - Seleciona conjunto de dados

Os comandos apresentados acima são utilizados em conjunto com outras expressões específicas do SQL, como:

- FROM – Especifica a tabela a ser buscada
- WHERE – Especifica condições para os dados buscados.
- GROUP BY – Agrupa os dados com base em certas colunas (utilizada em conjunto de operações de agregação).
- ORDER BY – Ordena os dados buscados com base em uma coluna ou colunas.

Também podem ser utilizados operadores lógicos como AND, OR e NOT, assim como operadores relacionais como Maior (>), Menor (<), Maior ou Igual (>=), Menor ou Igual (<=), Igual (=) ou Diferente (<>). Também é possível realizar operações matemáticas e estatísticas com os dados em um banco através dos comandos:

- AVG – Média
- COUNT – Contagem
- SUM – Soma
- MAX – Máximo
- MIN – Mínimo
- STDDEV - Desvio Padrão
- VARIANCE - Variância

Exemplos de aplicação da linguagem na prática podem ser observados abaixo, onde duas operações são realizadas utilizando o SQL para o comando.

```
SELECT
    ID_COMPRA,
    NOME_CLIENTE,
    VALOR_COMPRA
FROM
    VENDAS
WHERE
    VALOR_COMPRA >= 1000
```

```
SELECT
    NOME_CLIENTE,
    AVG(VALOR_COMPRA)
FROM
    VENDAS
GROUP BY
    NOME_CLIENTE
```

Em vista do exposto, percebe-se a vasta possibilidade de comandos e operações possíveis utilizando o SQL e o motivo de ter-se tornado uma linguagem utilizada para diversos fins. Através da mesma, é possível realizar a criação de tabelas com a DDL, realizar atualizações nos dados com a DML e consultar de diversas formas os dados com a DQL. Por conta dessa abrangência de possibilidades, o SQL é a linguagem padrão utilizada nas principais soluções de Bancos de Dados Relacionais disponíveis no mercado, sejam ferramentas pagas ou de código aberto. Além dos Bancos de Dados Relacionais, o SQL também pode ser utilizado em Bancos de Dados Não Somente Relacionais, utilizando-se abstrações do SQL que muito assemelham-se à sintaxe da linguagem, tornando-a extremamente democrática e necessária no mundo dos dados.

3.2 COMPUTAÇÃO DISTRIBUÍDA

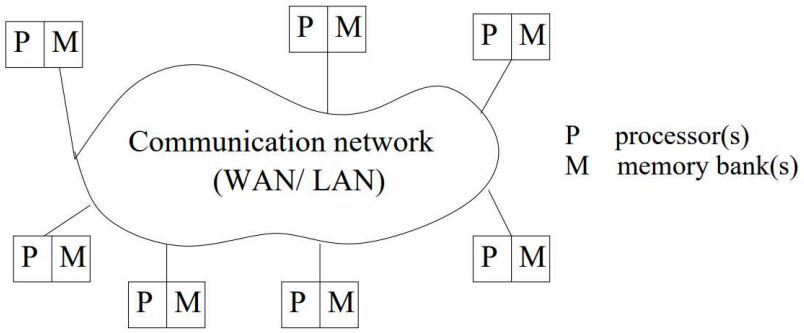
Com uma resposta natural ao avanço tecnológico observado ao longo das últimas décadas, cada vez mais empresas e outros órgãos migram suas operações para meios digitais. Naturalmente, observou-se uma necessidade crescente de maiores capacidades de processamento e armazenamento, uma vez que cada negócio possui em sua operação múltiplos sistemas, cada um com suas especificidades de bancos de dados. Além disso, o surgimento de novas tecnologias como dispositivos de Internet das Coisas (*Internet of Things (IOT)*), *smartphones*, veículos inteligentes, entre outros sistemas, contribuiu significativamente para a expansão do volume de dados processados. Esse aumento significativo do volume de dados gerados pelos sistemas e dispositivos é o que se conhece hoje por *Big Data*. A partir daí, novas tecnologias precisaram ser desenvolvidas a fim de que o avanço no volume de informações processadas pelos sistemas atendesse à demanda sem que os processos perdessem eficiência.

Um conceito muito importante a ser abordado no processo de crescimento do processamento é o de escalabilidade. Escalabilidade refere-se à capacidade de um sistema de responder a um aumento de processamento de dados. Duas possibilidades de escalabilidade para os sistemas são observadas, sendo elas a Escalabilidade Vertical e a outra a Escalabilidade Horizontal. A Escalabilidade Vertical é o aumento dos recursos computacionais em um servidor para o processamento de novos volumes de dados, ou seja, quanto maior o volume, maior a necessidade de *hardware*. Já a Escalabilidade Horizontal é a distribuição do processamento ao longo de diferentes servidores, ou nós, utilizando sistemas para gerenciar a distribuição dos dados e garantir a integridade do processamento (ALI; ABDULLAH, 2019). O escalonamento horizontal é possível graças aos sistemas de computação distribuída e representa a evolução necessária para atender às novas demandas de processamento de grandes volumes de dados.

O conceito de Computação Distribuída passa pelo conceito de Sistema Distri-

buído. Um sistema distribuído é composto por uma coleção de entidades independentes que cooperam para a realização de uma tarefa. Expandindo este conceito para a computação, tem-se um sistema interconectado de computadores capazes de realizar tarefas independentes com seus recursos alocados e comunicando-se através de uma rede, como pode-se observar na Figura 3 (KSHEMKALYANI; SINGHAL, 2011).

Figura 3 – Sistema distribuido de computação.



Fonte: (KSHEMKALYANI; SINGHAL, 2011).

Portanto, com sistemas de computação distribuída, é possível escalar horizontalmente o processamento de volumes massivos de dados, sem a necessidade de aumentar exponencialmente a capacidade de processamento de computadores isoladamente. Algumas necessidades podem ser elencadas para o uso de sistemas de computação distribuída, como:

- **Confiabilidade** - A distribuição de recursos computacionais permite uma maior segurança contra falha, uma vez que os recursos podem ser geograficamente distribuídos e a chance de um erro ocorrer é reduzida. Dessa forma, melhora-se a disponibilidade, integridade e tolerância a falhas dos sistemas.
- **Performance/Custo** - Ao distribuír as tarefas em diferentes computadores utilizando menores porções de recurso é possível otimizar a proporção de Performance/Custo quando comparado com aumentar somente os recursos de um computador apenas.
- **Modularização e Expansão** - Novos recursos computacionais podem ser adicionados ao sistema já existente sem afetar a performance.

Portanto, os sistemas de computação distribuída cumprem um papel fundamental para atender às novas necessidades advindas na era da *Big Data*. A seguir, serão apresentadas soluções de mercado que permitiram a escalabilidade horizontal do processamento de grandes volumes de dados com distribuição de recursos.

3.2.1 Apache Hadoop

Com o desenvolvimento de sistemas de computação distribuída, logo surgiram soluções comerciais ou de código aberto para realizar essas tarefas. Uma das soluções desenvolvidas mais conhecidas e utilizadas nos dias atuais é o Apache Hadoop, uma plataforma de *software* implementada na linguagem Java para computação distribuída, desenvolvida inicialmente por Doug Cutting e Mike Carafella como um motor de pesquisa para dados em larga escala. Baseando-se nos artigos publicados pelo Google elaborando o *Google File System (GFS)* (GHEMAWAT; GOBIOFF; LEUNG, 2003) e a técnica de *MapReduce* (DEAN; GHEMAWAT, 2004), Cutting e Carafella desenvolveram um *framework*, uma estrutura para desenvolvimento de software, utilizando a linguagem Java para implementar um sistema de arquivos com a camada de processamento distribuído. Quando Cutting foi trabalhar para o Yahoo, houve um esforço para transformar o projeto em código aberto, finalmente culminando no Apache Hadoop (ELMASRI; NAVATHE, 2015).

Os principais módulos que compõem o Hadoop são o *Hadoop Distribuited File System (HDFS)* e o *Hadoop MapReduce*, além de outros módulos não pertinentes a este trabalho. O HDFS é um sistema de arquivos distribuído capaz de armazenar os dados em diferentes servidores, comunicando-se entre si através de protocolos baseados em TCP. Os metadados são armazenados em um servidor dedicado chamado *NameNode*, enquanto os dados em si são armazenados em diversos servidores chamados *DataNodes*. Além disso, os dados armazenados são replicados em diferentes servidores, a fim de incrementar a confiabilidade e disponibilidade dos dados, prevenindo contra possíveis perdas de dados por falhas de *Hardware*, sendo possível armazenar grandes conjuntos de dados com segurança e confiança. Já o *MapReduce* será apresentado na seção seguinte por ser uma peça-chave da computação distribuída de dados que permitiu a evolução nos sistemas de processamento de grandes volumes.

3.2.2 MapReduce

Como mencionado anteriormente, o modelo de programação de *MapReduce* foi inicialmente desenvolvido nos laboratórios da Google Research por Jeffrey Dean e Sanjay Ghemawat (DEAN; GHEMAWAT, 2004). A implementação realizada por Dean e Ghemawat foi utilizando a linguagem de programação C++, enquanto a implementação de Cutting e Carafella para o Hadoop *MapReduce* foi com a linguagem Java. Contudo, ambas compartilham os mesmos princípios e funcionam de acordo com a plataforma em que estão desenvolvidas, este sendo o principal objetivo.

O modelo de *MapReduce* foi desenvolvido a partir de uma demanda do Google para lidar com grandes volumes de dados, como dados de sistemas Web. Lidar com

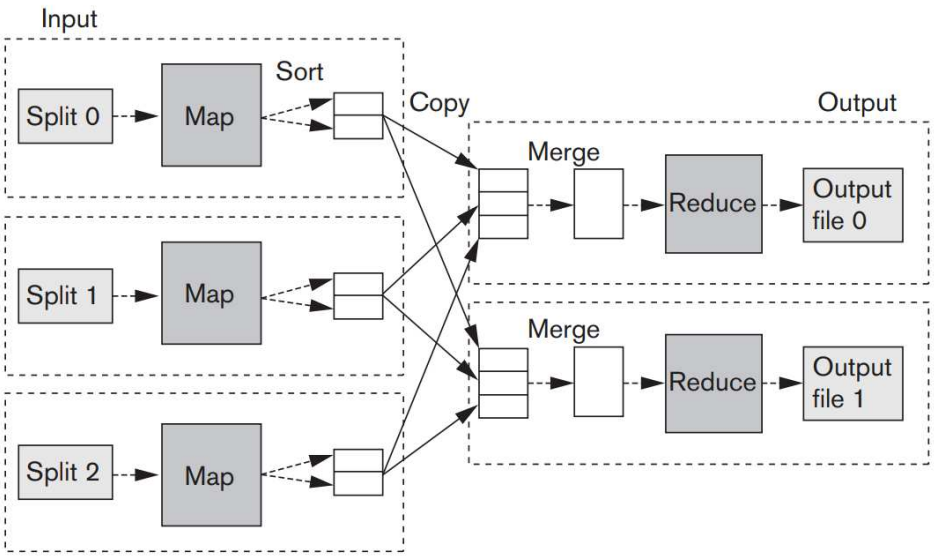
grandes volumes não era trivial, uma vez que eram necessárias implementações especiais capazes de buscar dados espalhados em milhares de máquinas, demandando um grande controle de execução, paralelização de poder computacional, distribuição de dados e controle de falhas. Com o objetivo de realizar essas tarefas de uma forma eficiente em um modelo que pudesse ser aplicado a diferentes conjuntos de dados sem a necessidade de implementações específicas para cada conjunto, o modelo de programação *MapReduce* foi desenvolvido (ELMASRI; NAVATHE, 2015).

O modelo de MapReduce baseia-se nas operações de *Map* e *Reduce* com os dados utilizados como entrada para processamento. A função de *Map* é uma função genérica que recebe os dados em formato de chave-valor na forma (K_1, V_1), os processa e retorna os dados no formato de chave-valor (K_2, V_2) utilizado como entrada para a função de *Reduce* que os processa e retorna o conjunto de dados com chave-valor (K_3, V_3) como resultado da operação. De forma geral, pode ser denotado por:

$$\begin{aligned} map(K_1, V_1) &: List(K_2, V_2) \\ reduce(K_2, List(V_2)) &: List(K_3, V_3) \end{aligned} \tag{1}$$

Onde a lista de valores (K_2, V_2) resultante da operação *Map* é utilizada como entrada para a função *Reduce* gerando o resultado final. O funcionamento base da função de *MapReduce* pode ser observado na Figura 4.

Figura 4 – Funcionamento da função de *MapReduce*.



Fonte: (ELMASRI; NAVATHE, 2015).

Pode-se explicar o modelo de MapReduce a partir do exemplo utilizado por Dean e Ghemawat (DEAN; GHEMAWAT, 2004) de uma tarefa para contabilizar a

frequência de ocorrência de palavras em um documento de texto. Utilizando o modelo de *MapReduce*, a função de *Map* recebe cada linha do documento como um valor (K_1, V_1), separa em palavras e gera uma lista (K_2, V_2) com cada palavra e a ocorrência igual a 1. Já a função *Reduce* recebe o conjunto de dados (K_2, V_2) e é a responsável por contabilizar, para cada palavra, a frequência de ocorrência de acordo com os conjuntos resultantes da função de *Map*. Dessa forma, é possível quebrar um conjunto de dados em pequenas partes e processá-los de maneira eficiente, acessando dados utilizando um sistema de arquivos como, por exemplo, o *HDFS* da Hadoop ou o próprio *GFS* da Google.

Diversas operações computacionais podem ser desenvolvidas aplicando uma função de *Map*, que gera resultados intermediários em formato de conjuntos de chave-valor e, então, aplicando a função de *Reduce* responsável por utilizar os resultados intermediários nos resultados finais da operação realizada. Como observa-se na Figura 4, o modelo de *MapReduce* realiza o particionamento dos dados e os aplica às funções, sendo estas operações realizadas de maneira paralela e distribuída, acelerando o processamento de grandes volumes de dados.

Entende-se, portanto, a relevância do modelo de *MapReduce* para o processamento de dados em larga escala, permitindo que o volume gerado pelas novas tecnologias empregadas nos diversos setores sociais fosse tratado de maneira adequada. A seguir, será apresentada uma solução desenvolvida pela Apache que significou o próximo passo em relação ao *MapReduce* e aplicada diretamente no processamento de dados do projeto implementado neste trabalho.

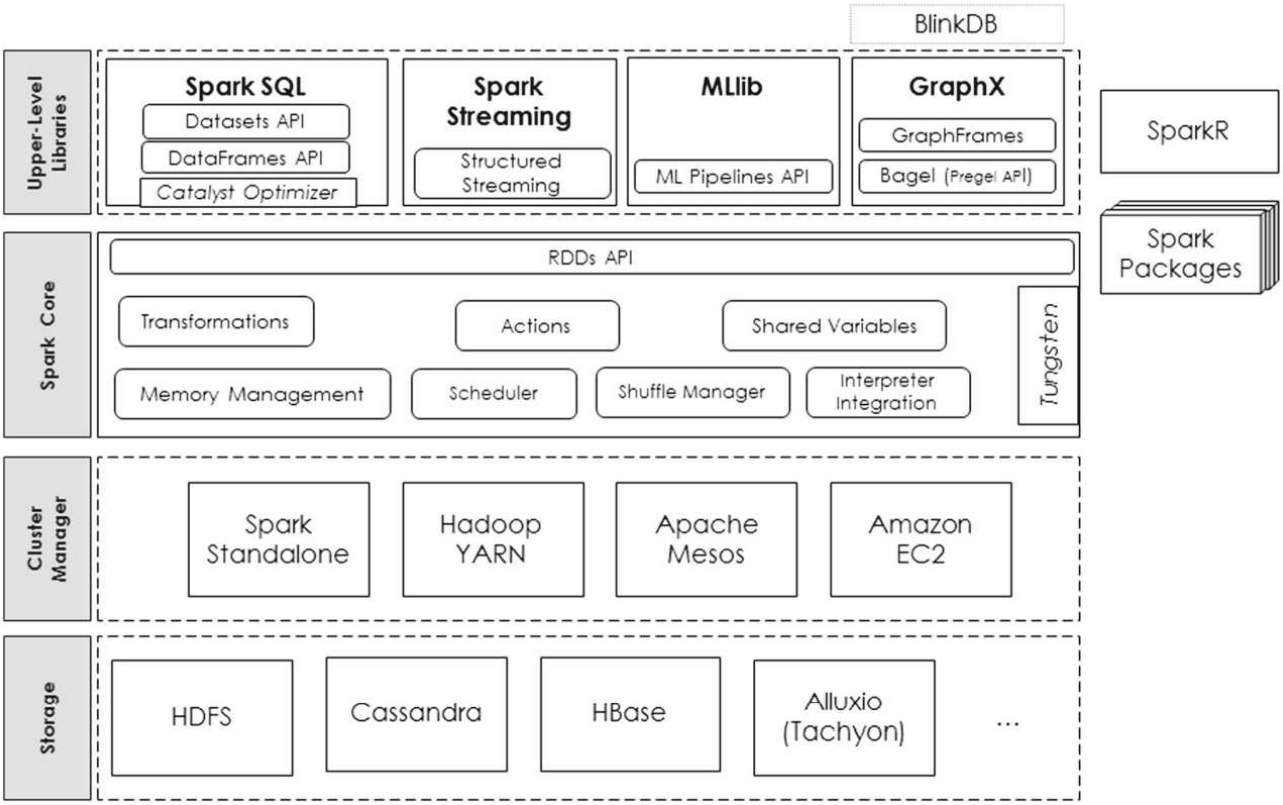
3.2.3 Apache Spark

Como já discutido nas seções anteriores, sistemas de computação distribuída desenvolvidos ao longo das últimas décadas foram essenciais para suprir as demandas de análise e processamento de dados da era da *Big Data*. Como evolução dos métodos, inicialmente desenvolvidos por grupos de pesquisa da Universidade da Califórnia e posteriormente passando para a *Apache Software Foundation (ASF)*, surge o Spark.

O Spark é um projeto de código aberto para implementação de um *framework* para processamento paralelo de dados em memória. Embora o modelo de *MapReduce* tenha sido o pioneiro no processamento paralelo de dados de maneira tolerante a falhas e escalável, fornecendo uma solução aplicável a diversos casos de uso, o mesmo provou-se inadequado para certas aplicações. Um dos casos é onde existe uma forte iteração nas operações com os dados, como modelos de ML que repetem uma mesma operação com o conjunto de dados para otimizar os parâmetros do modelo. Nestes casos, a operação de *MapReduce* precisa ser realizada repetidas vezes, recarregando os dados do disco, resultando em uma perda de eficiência. O outro caso são os de buscas em grandes conjuntos de dados, onde, idealmente, os dados seriam carregados na

memória de um conjunto de máquinas paralelas e posteriormente processados. Porém, utilizando o Hadoop, cada busca dos dados representa uma operação de *MapReduce* que realiza o carregamento das informações direto do disco, penalizando a eficiência da busca. Para superar esse problema, o *framework* Spark propõe um modelo de processamento com a mesma escalabilidade e tolerância a falhas do *MapReduce* baseado em conjuntos de dados em memória (ZAHARIA *et al.*, 2010).

Figura 5 – Arquitetura de alto nível do Apache Spark.



Fonte: (SALLOUM *et al.*, 2016).

A estrutura do Apache Spark está definida como apresentada na Figura 5. É possível notar que o Spark possui quatro principais elementos em sua arquitetura, sendo eles:

- **Storage** - Camada de armazenamento, onde estão alocados os dados podendo estar alocados em diferentes tipos de armazenamento, especialmente do ecossistema Hadoop.
- **Cluster Manager** - Camada de gerenciamento do conjunto de máquinas para processar os dados (*Clusters*), alocando recursos para os trabalhos de processamento chamados *Jobs.*

- **Spark Core** - Camada fundamental de interface de programação do Apache Spark, implementada em Scala e permitindo realizar o processamento de grandes volumes de dados com capacidade de gerenciamento de memória, agendamento de *jobs*, distribuição de dados e recuperação de falhas.
- **Upper-level libraries** - Camada de bibliotecas de funcionalidades do Apache Spark, possuindo diferentes bibliotecas voltadas para processamento de modelos de *ML*, análises através de linguagem SQL, processamento em *Streaming* entre outras funcionalidades.

O processamento via Spark dá-se a partir de alguns elementos que operam para que os dados sejam tratados de maneira adequada. Estes elementos fazem parte da estrutura apresentada na Figura 5, cada um com suas características e funções apresentadas a seguir.

3.2.3.1 Processamento via Spark

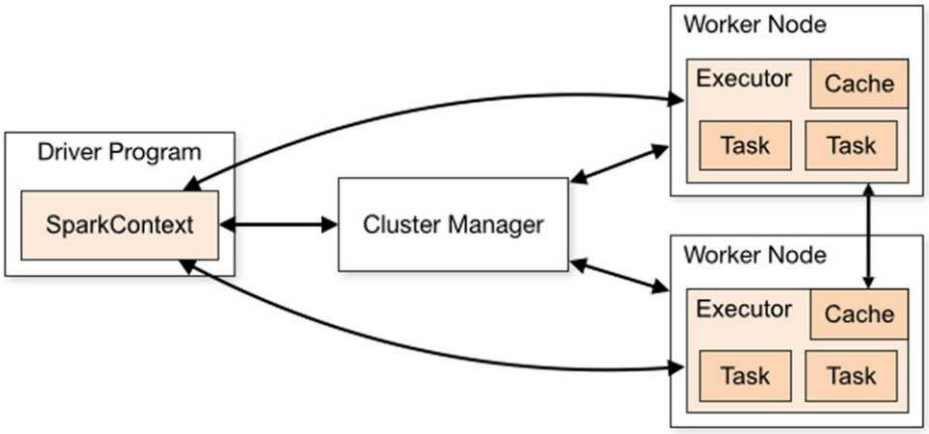
Aplicações Spark

Para rodar uma aplicação Spark, conforme a Figura 6, são necessárias cinco entidades:

- ***Driver Program*** - Camada de controle de fluxo dos processos realizados pelo Spark, utilizando o **SparkContext** para acessar as funções do Spark;
- **Gerenciador de *cluster*** - Camada de controle do conjunto de máquinas alocadas e recursos para a execução das tarefas do Spark;
- ***Workers*** - Camada física de processamento com CPU, memória e armazenamento alocado;
- **Executor** - Camada de software sob o qual o processo acontece, sendo realizado em uma *Java Virtual Machine (JVM)*;
- ***Tasks*** - Menor unidade de processamento do Spark, sendo um conjunto de *tasks* orquestradas dentro de um *Worker*.

Conforme a Figura 6, os elementos de processamento do Spark comunicam-se entre si com o objetivo de garantir que as tarefas sejam entregues para cada um dos elementos computacionais e gerenciadas pelas camadas de software.

Figura 6 – Entidades de uma aplicação Spark.



Fonte: (SALLOUM *et al.*, 2016).

RDDs

O cerne do Spark é desenvolvido a partir do conceito de *Resilient Distribuited Datasets (RDD)*, uma abstração de coleção de registros tolerante a falhas, com estrutura de dados paralela, controle de particionamento de dados e manipulação com uma série de operadores (SALLOUM *et al.*, 2016). Os *RDDs* podem ser criados a partir de dados externos em sistemas de gerenciamento de arquivos como o *HDFS* ou de outros *RDDs*, além de permitir a distribuição dos dados em diferentes processadores, característica essencial para o processamento de grandes volumes de dados distribuídos.

Os RDDs são elementos base do processamento via Spark, servindo como partições dos dados que são distribuídas ao longo do *cluster* para serem processados em pequenas partes, com replicação para controle de falhas e alta escalabilidade.

Transformações e Ações

O Spark possui um conjunto de operações conhecidas como *Parallel Operations* sendo elas Transformações e Ações. As transformações são operações como filtros, mapeamentos, *joins*, agrupamentos, entre outras transformações realizadas com os dados. Já as ações são operações como contagens, busca, etc. e são responsáveis por retornar os resultados de um RDD e salvá-los em um armazenamento externo. As Transformações somente acontecem quando uma Ação é chamada, retornando para o *Driver Program* os resultados do processamento.

***API*s e Bibliotecas**

Uma importante funcionalidade do Spark é permitir trabalhar com diferentes linguagens de programação. Em seu núcleo, o Spark é desenvolvido em Scala como já mencionado, porém, o mesmo possui *APIs* que permitem o uso de outras linguagens de programação como Java, Python e R, permitindo realizar Transformações e Ações com os dados nos RDDs utilizando diferentes sintaxes. Além disso, bibliotecas como SparkSQL permitem trabalhar com a linguagem padrão de manipulação de dados para realizar o processamento dos mesmos. Dessa forma, o Spark prova-se ser uma arquitetura extremamente democrática e um excelente recurso para o processamento e transformação de grandes volumes de dados de maneira eficaz.

3.3 COMPUTAÇÃO EM NUVEM

Um dos grandes desafios advindos da modernização dos processos empresariais e industriais a fim de que suas operações passem a ocorrer de maneira digital é o de prover a infraestrutura necessária para realizar tais operações. Por infraestrutura, entende-se toda a variedade de recursos necessários para manter os processos digitais rodando, desde apenas computadores pessoais até redes com inúmeros servidores, redes de internet internas para proteção das informações internamente, serviços de segurança digital para proteção contra invasões, entre outras necessidades. Como pode-se imaginar, estruturas físicas são necessárias para manter servidores e redes rodando, chamadas de *on premise* (no local), demandando grande custo financeiro e operacional para as empresas. Como consequência, essas demandas tornam-se obstáculos à otimização dos processos de negócio, uma vez que o custo de manutenção de infraestruturas de Tecnologia da Informação (TI) é demasiadamente alto e demanda um setor específico para esse fim.

Com a explosão do uso da internet, especialmente a partir do início do século XXI, surgiu uma nova gama de serviços de TI oferecidos pelas grandes empresas de tecnologia: a Computação em Nuvem. A computação em nuvem é definida como a disponibilização de serviços de computação como servidores, armazenamento de dados, gerenciamento de rede, ferramentas de desenvolvimento, software, entre outros, realizada através da internet (SUSNJARA; SMALLEY, 2024). A computação em nuvem está presente no dia a dia de diversas áreas, desde sistemas de e-mail e redes sociais até sistemas mais complexos, como aplicativos de mobilidade urbana ou sistemas de contagem de votos em eleições. Os serviços de computação em nuvem são disponibilizados por um Provedor de Serviços de Nuvem (*Cloud Service Provider (CSP)* normalmente com uma assinatura mensal associada ao uso ou pagamento conforme o uso (*Pay as You Go (PAYG)*). Como exemplos de CSP pode-se citar a Amazon Web Services (AWS), a Google Cloud Platform (GCP) e a Microsoft Azure como as

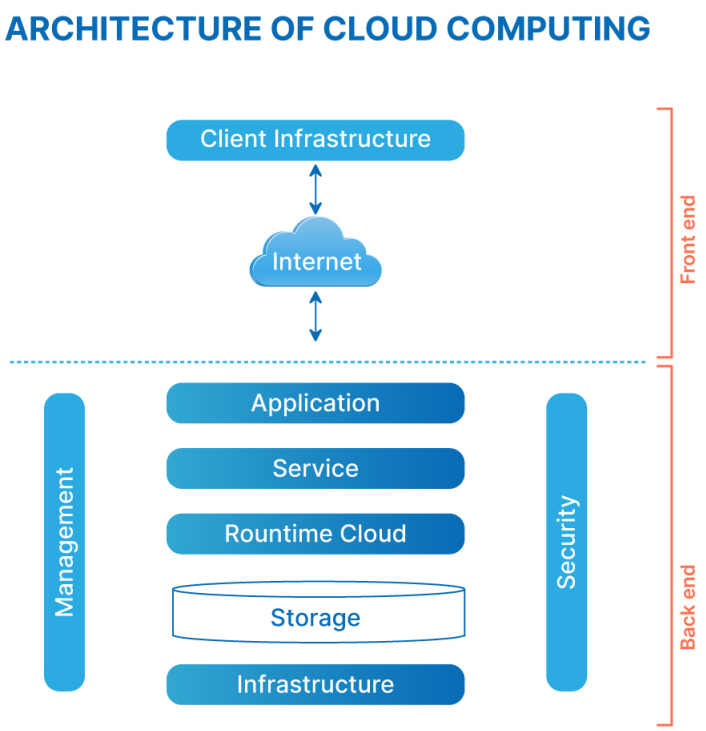
principais *Clouds* utilizadas no mercado, fornecendo serviços semelhantes, cada uma com suas particularidades de implementação.

Como mencionado anteriormente, a manutenção de grandes infraestruturas de computação físicas *on premise* acarreta um custo considerável para as empresas. Considerando diferentes necessidades como servidores, *gateways* de rede, cabeamento, técnicas de segurança e isolamento, energia elétrica e mão de obra qualificada para manutenção, manter uma infraestrutura de computação operando pode tornar-se um grande fardo para as empresas, uma vez que pode-se ter recursos subutilizados ou mal alocados e demandar grande qualificação para manter a estrutura operante. Em contrapartida, serviços de computação em nuvem oferecem os mesmos elementos de uma infraestrutura *on-premise*, porém, sem a necessidade de gerenciamento de estrutura física, de rede e de configurações, sendo cobrados somente pelo uso de cada serviço.

Além do custo, outras vantagens advêm da adoção dos serviços de computação em nuvem. Com a utilização de serviços de nuvem, os processos de TI tornam-se mais rápidos, uma vez que, com poucos cliques e configurações, é possível contratar serviços dos mais variados de maneira agilizada. Os serviços de nuvem também permitem uma melhor escalabilidade da infraestrutura do negócio, permitindo um provisionamento de hardware de maneira rápida e segura, podendo dimensionar a capacidade de computação necessária de acordo com a aplicação ou com eventuais sazonalidades, além de permitir utilizar o serviço em diferentes localidades no globo para assegurar maior disponibilidade e velocidade para o cliente final. A computação em nuvem permite também estar a par com as mais recentes tecnologias de IA, oferecendo a infraestrutura necessária para rodar modelos atuais ou ainda integrando-se com modelos já estabelecidos no mercado (SUSNJARA; SMALLEY, 2024).

A infraestrutura dos serviços de computação em nuvem pode ser representada conforme o diagrama da Figura 7. A mesma conta com diferentes elementos para garantir que toda a infraestrutura forneça aos clientes os serviços necessários. É importante destacar que a arquitetura da Figura 7 é apenas uma abstração de uma rede de proporções globais de datacenters espalhados ao redor do globo para garantir maior abrangência de serviço, garantir segurança e disponibilidade e menor latência de transmissão pela rede de internet. Destacam-se alguns elementos da arquitetura como a infraestrutura física de servidores em larga escala para atender às demandas de todos os clientes, a camada de armazenamento para lidar com diferentes volumes de dados de acordo com a natureza de operação, a camada de gerenciamento responsável pela provisão dos serviços de maneira adequada e a camada de segurança para garantir que as operações realizadas não violem a segurança dos clientes.

Figura 7 – Infraestrutura da Computação em nuvem.



Fonte: shiksha.com.

A computação em nuvem pode ser disponibilizada principalmente em três modelos de serviço, sendo eles:

- **IaaS (Infraestrutura como Serviço)** - Oferece recursos de infraestrutura como servidores, armazenamento e rede;
- **PaaS (Plataforma como Serviço)** - Fornece uma plataforma para desenvolvimento e implementação de aplicativos, sem que o usuário precise gerenciar a infraestrutura subjacente;
- **SaaS (Software como Serviço)** - Oferece acesso a softwares completos pela internet, como Google Workspace e Microsoft 365, sem a necessidade de instalação local.

Percebe-se que a Computação em Nuvem apresenta grandes vantagens econômicas e de desempenho comparada com as infraestruturas *on premise*. Neste trabalho, um Provedor de Serviços de Nuvem foi utilizado inteiramente para que, ao final, a Empresa Cliente tenha uma infraestrutura robusta, confiável e escalável para suas operações.

3.4 INFRAESTRUTURA DE DADOS

Empresas que apresentam grande competitividade de mercado nos dias atuais utilizam dados como suporte absoluto nas tomadas de decisão dos negócios, além de, em certos casos, utilizarem dados como ativos comerciáveis. Para que os dados estejam disponíveis para uso em aplicações que tragam retorno às empresas, é importante que uma infraestrutura de dados seja estabelecida com a confiabilidade e escalabilidade necessárias para as demandas crescentes.

No contexto em que este projeto se encontra, a migração de arquitetura foi realizada com o objetivo de que, ao final, toda a estrutura de relatórios permanecesse a mesma atual. A fim de resultar nas informações finais, conceitos de infraestrutura de dados foram utilizados como Extração, Transformação e Carga, dividida em camadas de transformação, sistemas de processamento de dados analíticos, Armazém de Dados (*Data Wharehouse*), Lago de Dados (*Data Lake*) e conceitos de armazenamento de dados moderno, apresentados adiante.

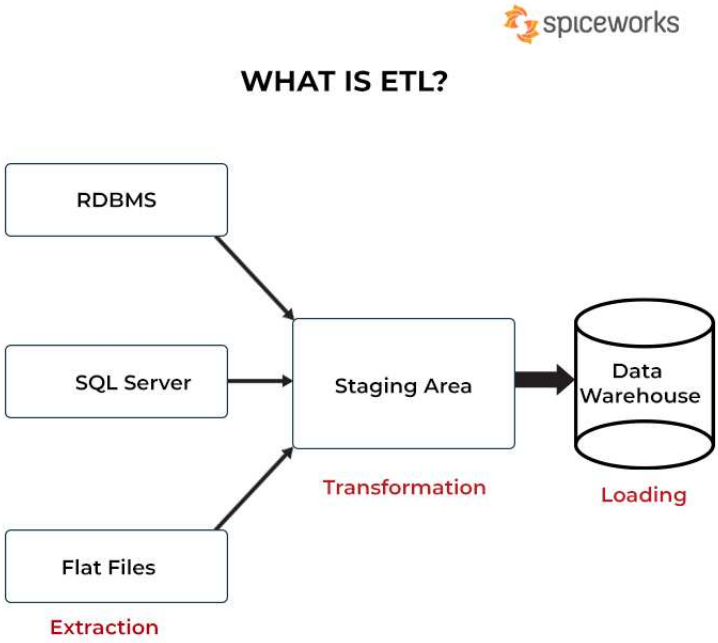
3.4.1 ETL e Camadas

ETL

Como já mencionado, dados são fundamentais para tomadas de decisão estratégicas capazes de ser um diferencial competitivo para empresas e negócios. Com a digitalização dos processos de negócios, sistemas geradores de dados foram cada vez mais adotados em diferentes setores, com informações possivelmente valiosas para seus consumidores. Porém, a descentralização das informações em diferentes sistemas como bancos de dados relacionais, arquivos de texto, planilhas, *APIs*, entre outros, tornou o processo de análise e utilização desses dados um desafio. A partir dessas dificuldades, empresas começaram a desenvolver sistemas de centralização de dados, chamados de Armazéns de dados ou, em Inglês, *Data Warehouse (DW)*. Para realizar a centralização dos dados, surgiu a partir da década de 70 o processo de *ETL* que vem do inglês *Extract* (Extrair), *Transform* (Transformar) e *Load* (Carregar) com o objetivo de realizar uma migração dos dados de diferentes fontes e prepará-los para utilização analítica em sistemas de Inteligência de Negócio (BI) ou ainda em modelos de IA. Nos dias atuais, o processo de *Extract, Transform, Load (ETL)* está no núcleo dos sistemas analíticos de dados, cumprindo papel fundamental para a boa integração de subsistemas em um local único para auxiliar negócios em melhores tomadas de decisão (ASHTARI, 2022).

O processo de ETL consiste em 3 etapas, sendo elas a Extração, a Transformação e a Carga, conforme a Figura 8.

Figura 8 – Processo de ETL.



Fonte: (ASHTARI, 2022).

A etapa de Extração consiste na cópia dos dados das diversas fontes de origem em formato bruto para um armazenamento dentro da arquitetura de dados. Nela, são extraídos dados de diferentes fontes e utilizando diferentes métodos de integração para ter acesso às informações. Os dados podem estar alocados em bancos de dados relacionais de uma infinidade de sistemas, como softwares de Planejamento de Recursos Empresariais (*ERPs*) ou de Gerenciamento de Relacionamento com o Cliente (*CRMs*), em bancos de dados não relacionais com dados não estruturados em formato *XML* ou *JSON*, arquivos de dados como *CSVs*, em *APIs* disponibilizadas por softwares terceiros, em dispositivos IOT, todos os sistemas com particularidades de operação e integração. Portanto, é importante realizar um processo de extração adequado e que se integre com os diferentes sistemas de maneira eficiente e confiável.

A etapa de transformação consiste em, a partir dos dados brutos, gerar uma estrutura de dados que permita o uso analítico dos mesmos. São realizadas tarefas de limpeza dos dados para remover informações desnecessárias ou inconsistentes, normalização para otimizar buscas, junções dos dados de diferentes origens em tabelas únicas, enriquecimento dos dados, conversões de tipos de dados, aplicação de regras de negócios específicas de acordo com a natureza do processo analisado. As transformações podem ter maior ou menor complexidade de acordo com a natureza e a necessidade do negócio analisado e do volume de dados transformado.

Por fim, a etapa de carga realiza o carregamento dos dados nos sistemas utili-

zados como fontes para os processos analíticos. A carga pode ser realizada para um DW, um banco de dados analítico ou ainda para um lago de dados (*Data Lake*) que será apresentado adiante. É a partir da carga que os dados tratados ficam disponibilizados para serem utilizados em sistemas de BI, em modelos de IA, ou em qualquer necessidade analítica que o negócio possa ter.

Nos dias atuais, grandes processos de ETL são realizados utilizando-se de computação em nuvem, uma vez que as soluções comerciais oferecem os serviços necessários para integrar e processar os dados. Adiante, neste trabalho, será apresentado como o processo de ETL foi realizado utilizando um serviço de nuvem disponível comercialmente.

Arquitetura Medalhão

A arquitetura medalhão é um padrão de arquitetura de dados para organização dos dados em um lago de dados (*Data Lake*). A mesma faz parte de um processo de ETL e é utilizada para transformar os dados em diferentes camadas, cada camada correspondendo a um nível de transformação e com o objetivo de aumentar a qualidade dos dados ao longo de cada uma. As camadas são divididas entre *Bronze* (Bronze), *Silver* (Prata) e *Gold* (Ouro) (DATABRICKS, 2024).

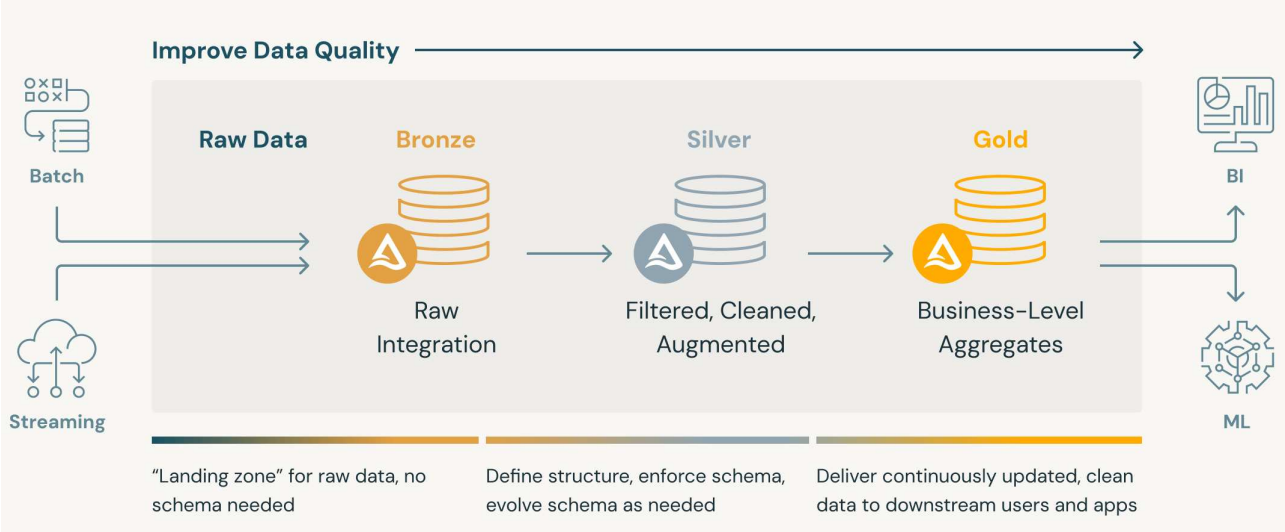
A camada *Bronze* contém os dados brutos como são copiados das fontes de dados, ou *as-is*. O objetivo dessa camada é servir como "área de pouso" para os dados ingeridos no processo de extração, mantendo a estrutura original e capturando as mudanças realizadas na fonte dos dados e reproduzindo as mesmas. Como a camada *Bronze* guarda as informações de mudanças nos dados, a mesma pode ser utilizada como camada de histórico, podendo rearranjar os dados de acordo com como eram em um período específico do tempo.

Já a camada *Silver* contém os dados tratados e limpos para aumentar a qualidade dos mesmos. São realizadas tarefas de remoção de espaços vazios, transformação de tipos de colunas, averiguação de duplicidade, entre outros tratamentos para tornar os dados utilizáveis a nível analítico. Na camada *Silver*, as diferentes áreas de negócio já podem fazer uso dos dados disponíveis para análise e, principalmente, atualmente, aplicar modelos de IA.

Já a camada *Gold* é a camada final, onde os dados são estruturados e preparados para serem utilizados de maneira analítica. Aqui, os dados são normalizados ao máximo para um melhor desempenho nas análises em ferramentas de BI e também são aplicadas regras de negócio e junções de dados para gerar as tabelas finais utilizadas. As tabelas geradas na camada Gold são normalmente voltadas às áreas de negócio e destinadas às análises dessas áreas, podendo seguir diferentes modelos, como, por exemplo, o modelo estrela desenvolvido por Kimball, muito utilizado nos *Data Warehouses* tradicionais.

Neste projeto, a arquitetura medalhão foi utilizada como modelo de transformação dos dados da Empresa Cliente. Nas seções seguintes, será apresentado o desenvolvimento dessa arquitetura.

Figura 9 – Arquitetura Medalhão.



Fonte: databricks.com.

3.4.2 OLTP VS OLAP

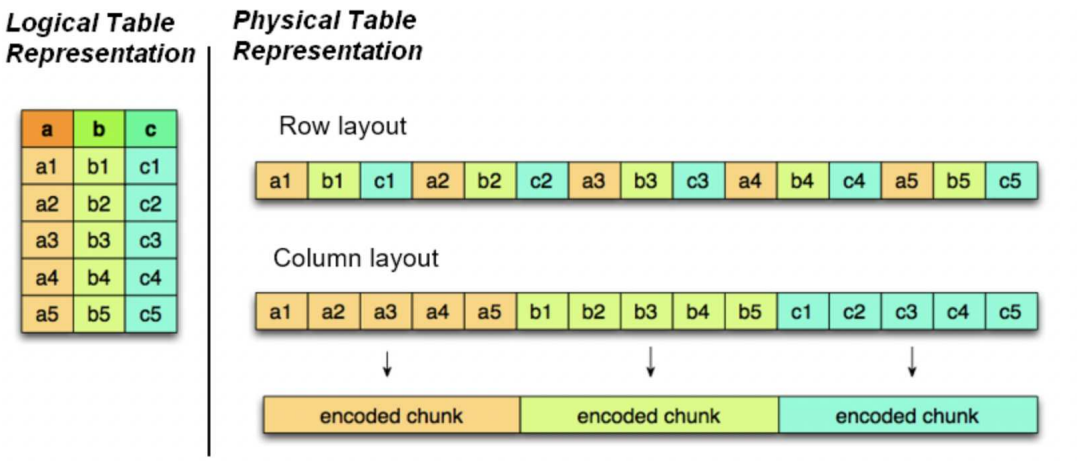
Os Bancos de Dados Relacionais utilizam os SGBDs como camada de controle e segurança para garantir que as transações sejam realizadas de forma segura, tolerante a falhas e com controle de versionamento. Apesar de serem extremamente aplicáveis em uma variedade de contextos de sistemas de dados, quando necessita-se de um sistema analítico cuja leitura e agregação dos dados são realizadas de maneira recorrente, os bancos de dados relacionais acabam perdendo eficiência no processamento dos dados. Neste contexto, surgem os bancos de dados analíticos voltados para análise de dados em grandes volumes e otimizados para operações de agregação e leitura. Bancos transacionais utilizam o sistema de processamento de transação online (*Online Transaction Processing (OLTP)*) enquanto os bancos analíticos utilizam o sistema de processamento analítico online (*Online Analytical Processing (OLAP)*) (CONN, 2005). Estes dois conceitos utilizados em conjunto foram as bases dos grandes sistemas de análises de dados utilizados nos dias atuais e serão apresentados adiante.

Os Bancos de Dados Relacionais utilizam, como já apresentado na Seção 3.1.1, ferramentas de segurança para garantir a eficiência das operações realizadas no mesmo, como adições de dados, atualizações e deleções. Ainda são características dos bancos transacionais as características ACID para garantir que cada transação

realizada nos bancos de dados seja efetiva e perdure por todo o banco. Portanto, bancos de dados relacionais são otimizados para operações de transação de dados, a fim de garantir o controle de inúmeros acessos simultâneos aos dados sem que nenhuma informação seja perdida ou corrompida, configurando um sistema de processamento de transação online (*OLTP*). Outra característica muito relevante de sistemas *OLTP* é o processamento em linha (*row oriented* que significa que em uma tabela de um sistema *OLTP*, os dados de cada tabela são armazenados e processados uma linha por vez, alterando, inserindo ou deletando todas as colunas daquela linha. O processamento em linha apresenta grandes vantagens na performance dos sistemas OLTP, uma vez que o mesmo consegue realizar transações com os dados de maneira rápida, atualizando todas as colunas de cada linha do banco em um único acesso ao Disco Rígido (*Hard Drive (HD)*) em que os dados estão registrados (BHAGAT; GOPAL, 2012).

Apesar da grande vantagem em operações de manipulação dos dados, com operações rápidas e seguras, os sistemas *OLTP* apresentam desvantagens em processamento analítico dos dados. O processamento analítico dos dados costuma utilizar funções de agregação em suas operações, buscando dados para realizar somas, contagens, médias, entre outras operações, realizadas normalmente sobre uma coluna específica do banco de dados. Quando em sistemas *OLTP*, o processamento busca os valores em cada uma das linhas separadamente, percorrendo todas as colunas da linha para buscar o dado que está sendo agregado, resultando em uma perda de eficiência, já que os dados de outras colunas não são necessários para realizar o cálculo desejado. Para suprir esta demanda, temos os sistemas de processamento analítico online *OLAP*, que realizam o processamento dos dados em coluna. Tomando a Figura 10 como exemplo, se uma soma é realizada sobre a coluna “a”, em um sistema *OLTP* todas as colunas “a”, “b” e “c” seriam percorridas para cada linha a fim de acumular os valores somente de “a” para realizar a soma. Já no *OLAP*, o sistema consegue realizar a busca de todos os dados da coluna “a” em um único acesso aos dados do *HD*, uma vez que o sistema realizará o acesso a apenas uma pequena parte de todos os dados registrados (BHAGAT; GOPAL, 2012).

Figura 10 – Comparação entre acesso em linha e em coluna.



Fonte: linkedin.com.

Além do acesso otimizado para consultas e operações de agregação, os sistemas *OLAP* também apresentam outras vantagens, como:

- Melhores atualizações de dados em colunas, uma vez que o acesso a cada coluna é realizado separadamente
- Melhor performance pois todos os dados da coluna possuem o mesmo tipo
- Possibilidade de uso paralelização dos dados, sendo possível organizar a tabelas em múltiplas unidades de processamento
- Melhor aplicado a cargas de trabalho analíticas, focadas em atributos dos dados e não em entidades como os sistemas *OLTP*
- Melhor acesso aos dados em disco ou em memória, utilizando menor processamento para acessar os dados necessários nas cargas analíticas

Entende-se, portanto, a importância de entender quando utilizar sistemas *OLTP* ou sistemas *OLAP*. Ainda pode-se utilizar os dois sistemas em conjunto, utilizando dados de sistemas *OLTP* para formar um *Data Warehouse* em formato *OLAP* a fim de suprir a demanda de cargas analíticas, especialmente em um contexto com grande volume de dados gerados diariamente e a utilização desses dados como fontes de informações valiosas para os negócios.

3.4.3 Data Warehouse, Data Lake e Data Lakehouse

Com a adesão das empresas à arquitetura de análise de dados, sistemas de unificação dos dados tiveram que ser desenvolvidos. Através dos processos de ETL,

passou-se a implementar uma infraestrutura analítica centralizada capaz de fornecer às empresas informações pertinentes a cada área de negócio que possam fornecer diretrizes importantes para as decisões tomadas e entendimento abrangente das operações da organização. Os conceitos dos sistemas de unificação dos dados serão apresentados a seguir.

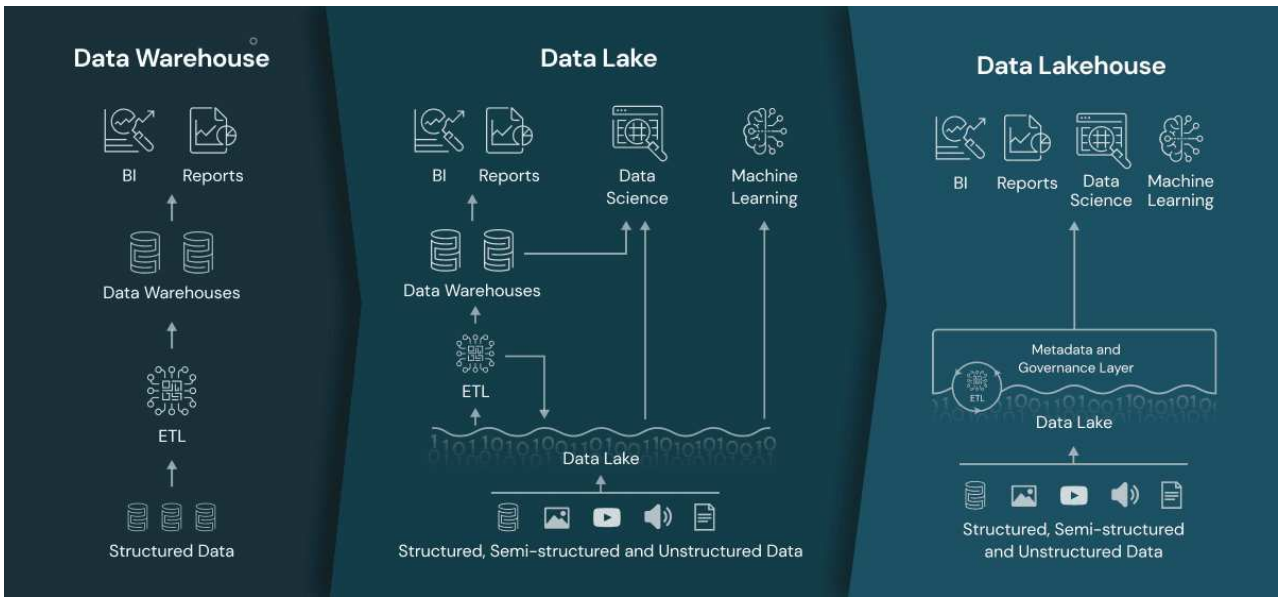
Tradicionalmente, as estruturas analíticas das empresas contavam somente com um armazém de dados, em inglês *Data Warehouse (DW)*. O DW é um modelo de armazenamento de dados, vindos normalmente de bancos de dados estruturados, voltado para a utilização em relatórios analíticos com dados limpos, transformados e preparados para função puramente analítica. Diferentes modelagens para o DW podem ser utilizadas a depender da natureza do processo analisado, sendo as principais baseadas nos modelos de Ralph Kimball e William H. Inmon. Na metodologia Inmon, busca-se o princípio de fonte única de verdade (*single source of truth*), onde os dados são modelados de maneira centralizada, a partir do qual pode-se criar segmentos específicos para as áreas do negócio. Já na metodologia Kimball, o foco está em modelar os dados de maneira focada em cada um dos times ou processos em que a análise está sendo realizada, fornecendo estruturas analíticas com foco em áreas e podendo ser compartilhadas entre elas (NAEEM, 2020). Independentemente da metodologia de modelagem de *Data Warehouse* utilizada, o objetivo é que os dados sejam concentrados em um armazenamento único, com a preparação dos mesmos para utilização principalmente analítica em sistemas de *Business Intelligence* e relatórios.

Como já exposto, o aumento do volume de dados produzidos especialmente pelo processo de digitalização dos processos empresariais e industriais, soluções mais robustas e amplas para armazenamento e tratamento desses dados também tiveram de ser criadas. Nesse cenário, surge o conceito de lago de dados, em inglês *Data Lake*, uma evolução dos tradicionais *DWs*. O *Data Lake* é também um centralizador de dados onde busca-se a criação de um grande repositório, contendo todos os dados gerados pela empresa vindos de diferentes fontes e com diferentes formatos (estruturados ou não) (KHINE; WANG, 2018). A partir desse conglomerado de dados, pode-se derivar diferentes aplicações e funcionalidades, haja vista que em um *Data Lake* os dados podem derivar tanto de diferentes origens quanto possuir diferentes formatos, como informação estruturada de bancos de dados relacionais ou ainda informação desestruturada, como vídeos, imagens, áudios, documentos, entre outros. Entre as aplicações, o *Data Lake* pode alimentar uma estrutura analítica (*Data Warehouse*) em sistemas de *BI*, realizar análises exploratórias, processar dados de dispositivos *IoT* em tempo real ou ainda alimentar modelos de Inteligência Artificial para categorização ou previsão.

Apesar de suprir a demanda de modelos para lidar com o crescente volume e com dados em diferentes formatos e origens, os *Data Lakes* apresentaram problemas em relação à sua performance. Entre os principais problemas, pode-se citar a dificul-

dade em processar os dados brutos obtidos das fontes, especialmente pelo fato destes apresentarem conteúdo e composição voláteis, levando a erros e inconsistências nos dados, obrigando os sistemas a constantemente buscarem metadados (dados sobre os dados) para estruturar as informações processadas. Esse processo acaba levando o *Data Lake* a um estado de *Data Swamp* (pântano de dados) com informações pouco coerentes e com necessidade de etapas de verificação e validação, desperdiçando o valor dos *Data Lakes* (HARBY; ZULKERNINE, 2022). Para superar estes problemas, o conceito de *Data Lakehouse* foi desenvolvido, com o objetivo de unir as capacidades de processamento analítico dos *Data Warehouses* e o suporte ao armazenamento de *Big Data* dos *Data Lakehouses*, mantendo partes dessas tecnologias em sua estrutura. Tecnicamente e de maneira geral, os *Data Lakehouses* buscam integrar tecnologias de processamento de grandes volumes de dados, como sistemas de arquivos distribuídos baseados no Hadoop *HDFS* e Hadoop MapReduce, assim como uma camada de metadados e governança para assegurar as características ACID dos bancos de dados relacionais nas cargas analíticas dos dados. A camada de governança é a responsável por garantir que os dados tenham sua estrutura definida e não se percam com as atualizações e mudanças de estrutura. Dessa forma, é possível garantir que cargas analíticas baseadas em OLAP sejam realizadas sem perder as propriedades ACID de confiança nas transações dos dados e ainda fornecer uma arquitetura de dados capaz de lidar com diferentes fontes, formatos e aplicações dos dados de uma organização, tornando-se um possível padrão de ouro para a indústria (HARBY; ZULKERNINE, 2022).

Figura 11 – *Data Warehouse*, *Data Lake* e *Data Lakehouse*



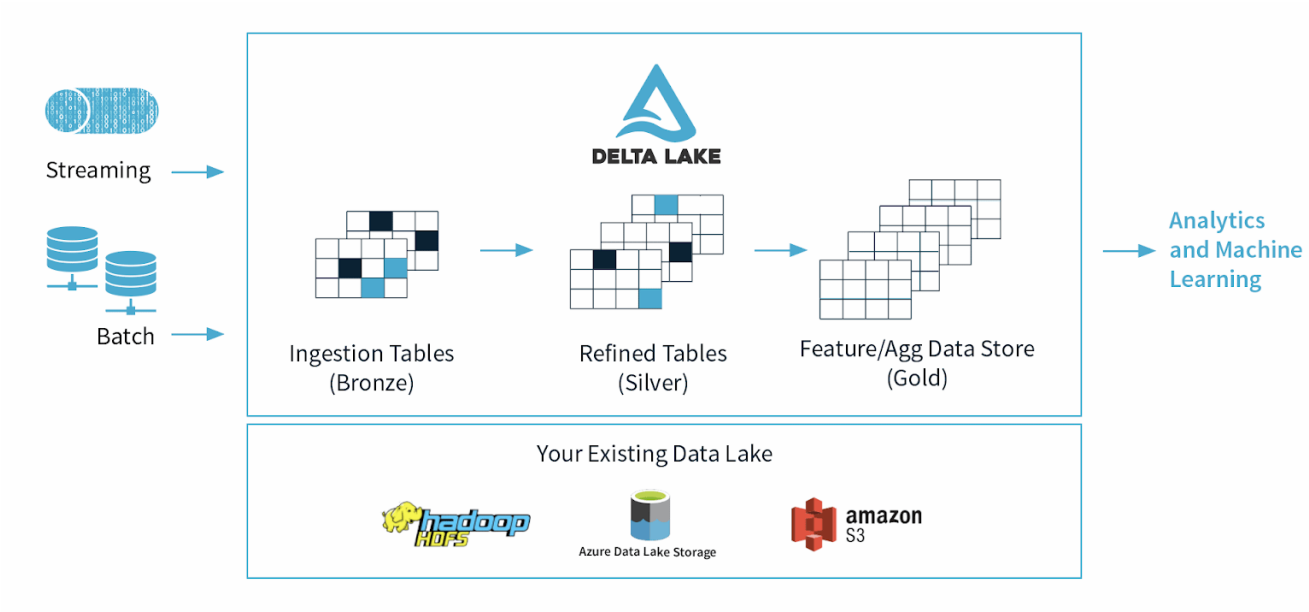
Fonte: databricks.com.

3.4.4 Delta Lake

Uma tecnologia muito importante que permitiu a criação dos *Data Lakehouses* foi o Delta Lake. O Delta Lake consiste em um *framework* de código aberto de armazenamento de dados voltado ao desenvolvimento de *Data Lakehouse* que garante Atomicidade, Consistência, Isolamento e Durabilidade, configurando um sistema com características ACID (L'ESTEVE, 2021). Possuir características ACID permite que o Delta Lake seja uma solução muito adequada para cenários em que se trabalha com dados estruturados sobre uma arquitetura de *Data Lakehouse*, uma vez que se tem o controle e segurança das transações, assim como nos bancos de dados relacionais, ao mesmo tempo que opera em uma arquitetura de dados capaz de lidar com grandes volumes de dados com grande eficiência. Como visto na Figura 11, o *Data Lakehouse* possui uma camada de governança e metadados para assegurar que os dados ingeridos em seu armazenamento possuam uma estrutura e organização definidas. O Delta Lake aproveita-se da estrutura de metadados para lidar com atualizações e transações dos dados, salvando suas informações em formato de arquivo '.parquet', um formato otimizado, com arquivos auxiliares de histórico de mudanças nas informações registradas. O Delta Lake é desenvolvido sobre uma solução de armazenamento de dados como, por exemplo, o Hadoop HDFS, Amazon S3 ou ainda o Azure Data Lake Storage, vide Figura 12, utilizando técnicas otimizadas de gerenciamento e armazenamento de dados. Dessa forma, o Delta Lake torna-se muito útil em um contexto de *Data lakehouse*, onde busca-se uma plataforma robusta para processamento de dados estruturados ou não, podendo alimentar camadas analíticas das empresas ou camadas de exploração de dados com modelos de ciência de dados, utilizando uma arquitetura única e centralizada.

Para desenvolver um Delta Lake, pode-se utilizar uma série de ferramentas disponíveis no mercado como Apache Spark, Google BigQuery, Databricks ou ainda *APIs* para trabalhar de maneira autônoma com linguagens como Java, Python ou Rust. A Figura 12 mostra im *pipeline* utilizando o Delta Lake como formato de armazenamento de dados, podendo ser aplicado em um Data Lake ou Data Lakehouse existente sobre uma ferramenta de armazenamento de dados, realizando todo o processo de ETL com as tabelas em formato Delta.

Figura 12 – *Pipeline* Delta Lake.



Fonte: purestorage.com.

3.5 *APLICATION PROGRAMING INTERFACE (API)*

No desenvolvimento de sistemas, é necessário que sejam feitas integrações entre diferentes programas, seja para compartilhar dados, funcionalidades ou recursos. Para esta finalidade existem as Interfaces de Programação de Aplicativos, do inglês, *Application Programing Interface (API)*. As *APIs* seguem um conjunto de regras e protocolos de comunicação para realizar a interconexão entre diferentes sistemas, exportando desde dados de um sistema para outro até mesmo funcionalidades específicas de um sistema, não havendo necessidade de um programa desenvolver uma dada funcionalidade internamente (GOODWIN, 2024). Um exemplo cotidiano são os sistemas de pagamento online, que normalmente precisam conectar-se a diferentes serviços como cartão de crédito, geradores de boleto, códigos PIX no caso de pagamentos no Brasil para gerar uma operação de pagamento. Os sistemas de pagamento disponibilizam uma API para que um *e-commerce* utilize o serviço através de uma *API* e realize a transferência sem que a plataforma do *e-commerce* necessite desenvolver tal funcionalidade. A utilização de uma *API* pode ser entendida como uma comunicação entre um cliente e um servidor, no qual o cliente realiza uma requisição ao servidor utilizando um protocolo de comunicação e o servidor devolve uma resposta a esta requisição quando a transação é autorizada, estabelecendo a conexão entre os sistemas (GOODWIN, 2024).

As *APIs* podem ter diferentes finalidades a depender do caso de uso. Podem ser *APIs* de Dados com a função de compartilhar informações entre os SGBDs, *APIs*

de Sistema Operacional quando a mesma conecta aplicativos dentro de um sistema operacional, *APIs* Remotas quando diferentes aplicações são conectadas ou ainda *APIs* Web quando funcionalidades e dados são compartilhados através da internet. Nos dias atuais, grande parte das *APIs* desenvolvidas são voltadas à web, utilizando o protocolo HTTP com conexão TCP para transferências de dados.

As *APIs* são desenvolvidas utilizando padrões e estilos de comunicação para garantir a interoperabilidade entre elas. Entre os exemplos de padrão de comunicação, pode-se citar:

- *Simple Object Access Protocol (SOAP)* - Utilizam comunicação SMTP e HTTP para realizar transferências utilizando estrutura de informação XML
- *Remote Procedure Call (RPC)* - Utilizam comunicação TCP/IP ou UDP entre cliente e servidor para comunicação entre aplicativos em outro computador ou em rede
- *WebSocket* - Utilizam comunicação bidirecional TCP para troca de informações com troca contínua e em tempo real
- *Representational State Transfer (REST)* - Utilizam o protocolo HTTP com operações de GET, PUT, HEAD ou DELETE para trocas de dados entre sistemas de maneira sem estado, ou seja, sem persistência das informações

Entre as vantagens do uso das *APIs* tem-se a colaboração entre plataformas e aplicativos, permitindo o uso de diferentes funcionalidades e compartilhamento de informações, monetização dos dados obtidos através da venda do acesso aos mesmos via *APIs* e a segurança através de credenciais específicas para o acesso às *APIs* para evitar permissões indevidas.

A utilização de *APIs* em arquiteturas de dados é de extrema importância para a integração entre diferentes fontes de dados que possam fazer parte do ecossistema de uma empresa. Com as *APIs*, é possível conectar-se a diferentes origens de dados, sejam elas internas ou externas, ou ainda utilizar funcionalidades dos sistemas de processamento como Apache Spark para tratar dados sem a necessidade de implementar funcionalidades internas nos *pipelines de dados*.

4 DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS

Nesse capítulo será apresentado o problema encontrado e como o mesmo foi solucionado com o desenvolvimento do projeto apresentado neste documento. Serão explorados os contextos da Empresa Cliente e por que o projeto implementado foi a solução adequada para suas necessidades. Também serão descritos os requisitos técnicos do projeto para garantir que o resultado final esteja alinhado com as expectativas da Empresa Cliente.

4.1 CONTEXTO DA EMPRESA CLIENTE

A Empresa Cliente possui um longo histórico de projetos realizados com a BIX TECNOLOGIA. Ao longo de anos de parceria, foram desenvolvidas arquiteturas de dados para suportar as diferentes áreas da Empresa Cliente, com relatórios visuais de BI modelados e implementados para atender diferentes demandas e níveis de análise. As mais variadas estruturas, desde visuais simples com poucos gráficos e tabelas descritivas, até visuais mais complexos com estrutura interativa, múltiplas telas para níveis diferentes de análises e menus complexos de controle da informação apresentada.

Toda a infraestrutura de dados atual da Empresa Cliente foi executada utilizando as ferramentas oferecidas pela empresa Microsoft, especialmente os serviços de software como serviço (*Software as a Service (SaaS)*) da Microsoft Power Platform. A Microsoft Power Platform engloba um conjunto de ferramentas de pouco código (*Low-code*) para desenvolvimento de automações a nível de negócio, criação de aplicativos, chatbots e análise de dados. O objetivo da plataforma é oferecer às empresas um conjunto de ferramentas simples de serem configuradas, fortemente integradas entre si e com conexões com diversos serviços externos muito utilizados pelas empresas como *ERPs* ou Bancos de Dados amplamente utilizados no mercado, auxiliando os negócios em suas tomadas de decisão, automação de tarefas e desenvolvimento de aplicações específicas do contexto de cada organização.

Entre os serviços fornecidos na Microsoft Power Platform, no contexto da arquitetura de dados da Empresa Cliente, destacam-se dois: o serviço de automação Power Automate e o serviço de *Business Intelligence* Power BI. Estes dois serviços permitem que empresas realizem automações de processos e desenvolvam análises de dados pertinentes ao seu negócio. Por se tratar de uma grande empresa do ramo da extração, produção e comercialização de sal, a Empresa Cliente possui uma grande estrutura física com diferentes etapas e camadas para a extração e produção do sal, além de uma estrutura de TI capaz de suportar os diferentes sistemas e subsistemas utilizados em suas operações. A fim de que se pudesse ter um entendimento mais amplo das operações da empresa e da qualidade de produção, em projetos anteri-

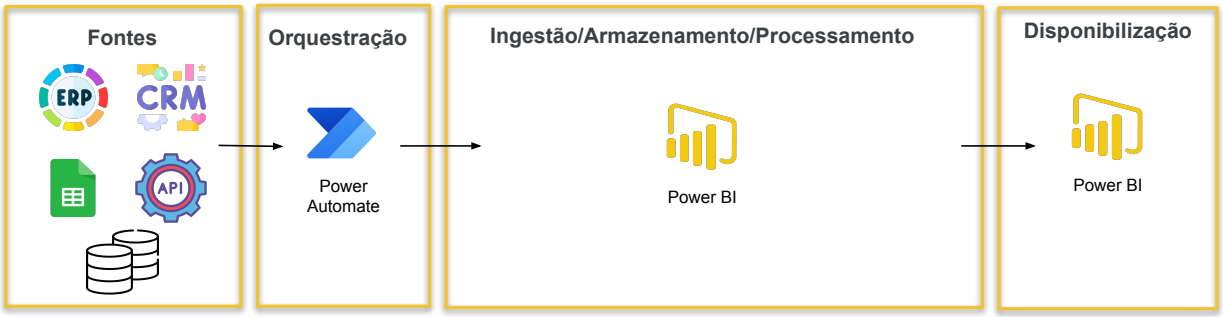
ores realizados pela BIX TECNOLOGIA, uma infraestrutura de relatórios visuais de *BI* foi desenvolvida para a Empresa Cliente, utilizando os serviços Power BI e Power Automate. Apesar de muito aplicáveis em diferentes contextos de negócio, os sistemas de *Business Intelligence* possuem limitações quanto ao desempenho com diferentes volumes de dados analisados, podendo tornar-se um gargalo para a boa visualização das informações da empresa e dificultando as análises. A seguir, serão discutidas as características das arquiteturas voltadas a serviços de *Business Intelligence* e suas limitações, especialmente no contexto da Empresa Cliente.

4.2 ARQUITETURA VOLTADA A SERVIÇOS DE *BUSINESS INTELLIGENCE (BI)*

Os sistemas de *BI* disponibilizados no mercado são amplamente difundidos entre as empresas e cumprem um papel fundamental na análise de dados dessas organizações. Com um sistema de *BI* bem modelado, entendendo-se a natureza do negócio, os detalhes de operação e onde estão as informações relevantes, é possível auxiliar a tomada de decisão e organização do negócio com relatórios visuais apresentando as informações de maneira prática e benéfica.

Para alimentar os dados de um relatório de BI, diferentes maneiras podem ser utilizadas, desde usar o próprio armazenamento interno do serviço de BI, conectar com uma fonte externa consumindo dados em tempo real e utilizando a capacidade de transição de dados do serviço de BI e da disponibilidade da fonte ou ainda desenvolver um *Data Warehouse* ou um *Data Lake* ou um *Data Lakehouse*, sendo esta a que considera-se a boa prática para desenvolvimento de arquiteturas de dados voltadas à análise. Para pequenos e até médios negócios, armazenar os dados utilizando o próprio armazenamento interno dos serviços de BI pode ser muito útil e simples de implementar, uma vez que normalmente pequenas empresas possuem poucos dados armazenados, normalmente em planilhas ou em bancos de dados relacionais simples. Contudo, quando uma organização possui uma expressiva presença no mercado, atendendo milhares ou milhões de clientes, as estruturas geradoras de dados tornam-se complexas e heterogêneas. Desde softwares de controle de operação, RH, *ERPs*, *CRMs*, Comercial até planilhas e estruturas de dados menores utilizadas pelas áreas, uma grande organização tende a possuir uma vasta quantidade de dados para serem processados e analisados, o que é o caso da Empresa Cliente atendida pela BIX TECNOLOGIA nesse projeto. A Figura 13 apresenta a arquitetura utilizada pela Empresa Cliente anteriormente, utilizando os serviços da Microsoft.

Figura 13 – Arquitetura de dados anterior da Empresa Cliente.



Fonte: Autor.

Apesar de limitado, os serviços oferecidos pelo Microsoft Power BI foram capazes de atender, por um período, às necessidades da Empresa Cliente. No Power BI, as tabelas que alimentam os relatórios são transformadas internamente em Modelos Semânticos, este sendo a estrutura base para alimentar os relatórios visuais. A Empresa Cliente possui dados advindos de diferentes fontes, desde bancos de dados relacionais de softwares utilizados, API de serviços até planilhas de organização interna alocadas em serviços da Microsoft. Essa ampla variedade de sistemas integra-se ao Power BI através de conectores internos implementados para cada uma e alimenta os modelos semânticos com as informações que são apresentadas nos relatórios visuais. Os Modelos Semânticos possuem um limite de tamanho de 1 GB (um gigabyte) por modelo que, mesmo sendo capaz de armazenar uma quantidade de dados elevada, acaba tornando-se ineficiente com o constante aumento das informações copiadas das fontes de dados de origem. Além disso, o tipo de conta utilizada pela Empresa Cliente (Power BI Pro) possui um limite de processamento em termos de memória RAM e CPU alocados para os tratamentos nos modelos semânticos, o que resulta em lentidão e perda de eficiência. Por um período, os próprios modelos semânticos do Power BI foram capazes de extrair os dados das diferentes fontes e transformá-los para serem utilizados nos relatórios visuais, mas que se tornaram ineficientes com o passar do tempo e o aumento de dados gerados pelas fontes, resultando em travamentos constantes, demora para processamento e perda de eficiência e segurança da infraestrutura atual da Empresa Cliente. Dessa forma, entende-se a necessidade de desenvolvimento de uma arquitetura de dados nova, mais robusta, que permita a conexão com as fontes de dados dos *softwares* utilizados pela Empresa Cliente e que seja robusta o suficiente para garantir a eficiência necessária para processar o volume de dados atual da empresa, este sendo o projeto desenvolvido e apresentado neste documento.

4.3 REQUISITOS TÉCNICOS DO PROJETO

Para o desenvolvimento do projeto, os requisitos funcionais e não funcionais apresentados nas Tabelas 1 e 2, respectivamente, foram considerados.

Tabela 1 – Requisitos Funcionais.

ID	REQUISITO	CATEGORIA
F1	A arquitetura deve ser capaz de extrair os dados dos diferentes sistemas utilizados pela Empresa Cliente	Essencial
F2	A arquitetura deve ser capaz de processar o volume de dados atual da empresa de maneira eficiente	Essencial
F3	Os dados devem ser tratados em camadas utilizando arquitetura medalhão e armazenados em serviço adequado	Essencial
F4	A camada *bronze* deve possuir um histórico de registros	Essencial
F5	A camada *silver* deve realizar o pré-processamento dos dados para assegurar a qualidade	Essencial
F6	A camada *gold* deve realizar os agrupamentos e tratamentos finais das tabelas	Essencial
F7	O processamento será realizado com Delta Lake para controle de versões e otimização	Essencial
F8	Tabelas incrementais devem ser utilizadas sempre que necessário	Importante
F9	As tabelas implementadas devem refletir exatamente as tabelas da arquitetura atual	Essencial
F10	Melhorias e manutenções devem ser realizadas sem alterar o fluxo principal	Essencial

Fonte: Autor.

Tabela 2 – Requisitos Não Funcionais.

ID	REQUISITO	CATEGORIA
NF1	Os *pipelines* de dados devem ser atualizados de acordo com a arquitetura atual	Essencial
NF2	O monitoramento dos *pipelines* deve ser realizado com logs de execução	Essencial
NF3	A arquitetura deve ser escalável, permitindo o aumento do poder de processamento de acordo com a demanda	Essencial
NF4	A arquitetura deve conectar-se com a ferramenta de visualização Power BI para migração dos relatórios visuais	Essencial
NF5	O custo de manutenção da arquitetura deve ser o mais baixo possível de acordo com o volume de dados processado	Essencial

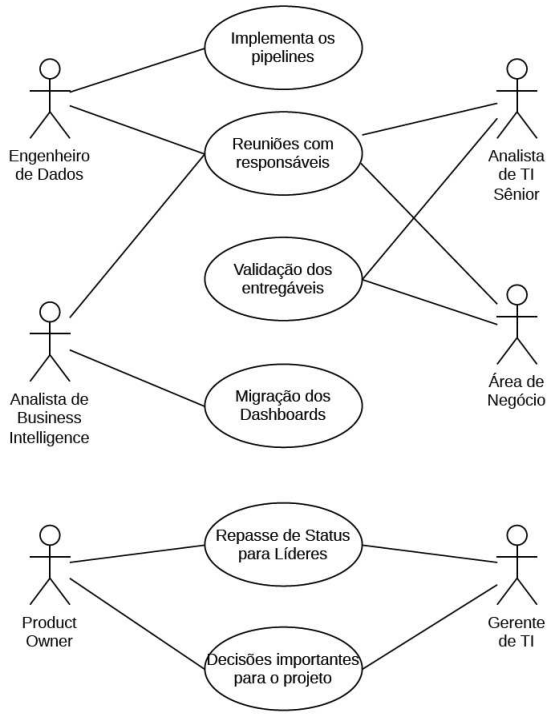
Fonte: Autor.

4.3.1 Diagrama de Caso de Uso

As interações do projeto foram fundamentais para o andamento contínuo do mesmo. A Figura 14 apresenta o diagrama de caso de uso, considerando cada agente e suas funções no andamento do projeto. O Engenheiro de Dados, neste caso o autor, é o responsável pela implementação dos pipelines de migração para a nova arquitetura, também realizando reuniões e contatos com os agentes da Empresa Cliente. O Analista de *BI* e o *Product Owner* são agentes da BIX TECNOLOGIA responsáveis

por, respectivamente, implementar os novos relatórios visuais com dados vindos da nova arquitetura e gerenciar o projeto, mantendo contato com os líderes do projeto do lado da Empresa Cliente. O Analista de TI Sênior é o responsável pelo contato com o Engenheiro de Dados e o Analista de *BI* para auxílio no entendimento da arquitetura atual e das regras de negócio aplicadas nos dados. Já a Área de Negócio é responsável pela validação dos entregáveis do projeto para confirmar que as implementações foram realizadas de maneira satisfatória.

Figura 14 – Diagrama de Caso de Uso da nova arquitetura.



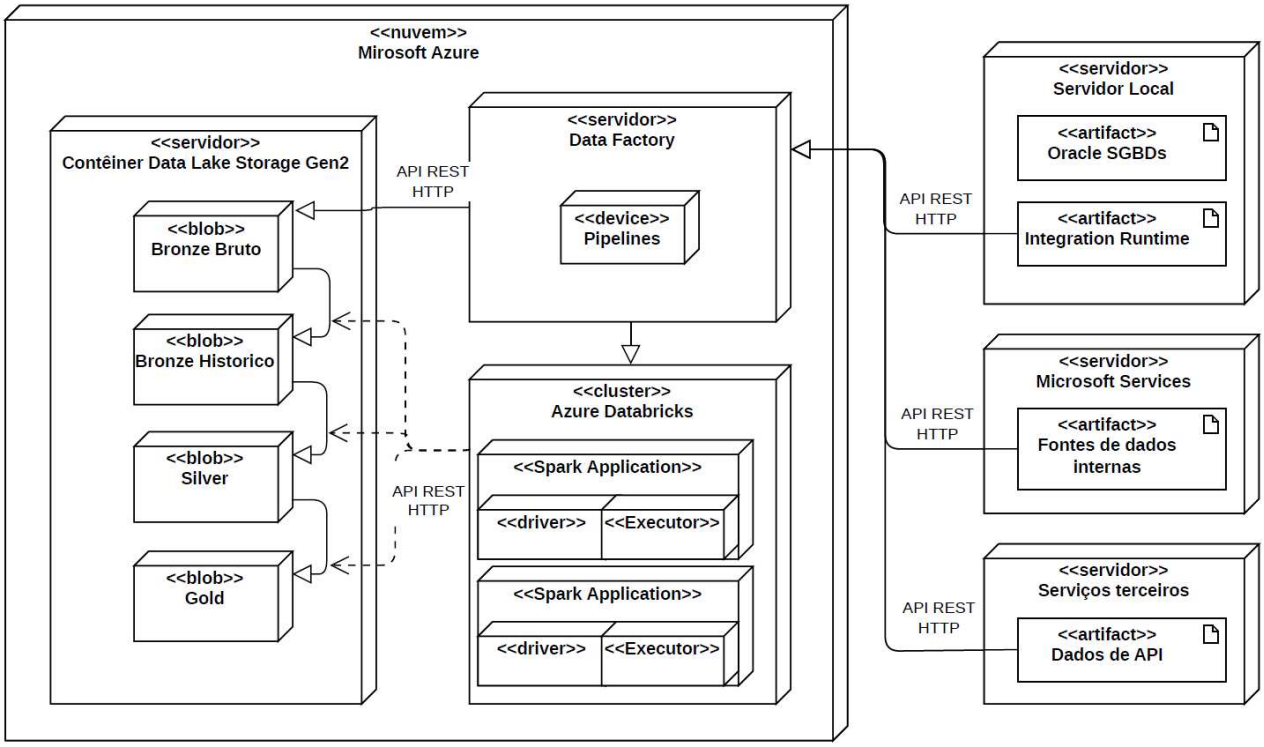
Fonte: Autor.

4.3.2 Diagrama de deployment

A fim de realizar toda a estruturação do projeto e a implantação (*deployment*), são necessárias infraestruturas fornecidas pelo serviço de nuvem escolhido e conexões entre essas infraestruturas. A Figura 15 apresenta o Diagrama de *Deployment* da nova arquitetura completa. Nela, é possível observar os elementos da arquitetura como o Data Lake Storage Gen2, o Azure Databricks, Azure Data Factory, todos parte dos servidores de nuvem da Microsoft Azure, além dos servidores locais (*on-premise*) da Empresa Cliente, dos serviços da Microsoft externos utilizados pela Empresa Cliente e dos serviços terceiros utilizados que disponibilizam *APIs* para a extração dos dados. Toda a comunicação entre os elementos da arquitetura é realizada via protocolo HTTP e *APIs* de conexão entre os diferentes serviços. Dessa forma, é possível entregar toda

a infraestrutura necessária em nuvem para implementar a nova arquitetura de dados da Empresa Cliente.

Figura 15 – Diagrama de *Deployment* da nova arquitetura.



Fonte: Autor.

5 SOLUÇÃO PROPOSTA E METODOLOGIA UTILIZADA

Neste capítulo, serão apresentados os principais aspectos da solução proposta para solucionar o problema elencado na seção anterior. Serão apresentadas as metodologias utilizadas, os detalhes da nova arquitetura de dados proposta para atender à Empresa Cliente, os motivos das escolhas das ferramentas e o impacto que a arquitetura tem sobre o cliente.

5.0.1 Metodologia

As ferramentas e serviços necessários para implementar a nova arquitetura foram selecionados de acordo com as metodologias empregadas no projeto. Os métodos utilizados no processo de construção do *Data Lakehouse* da Empresa Cliente, bem como os processos de extração, transformação e carga, agrupam um conjunto de operações realizadas com os dados que serão descritas a seguir.

Para criar uma arquitetura que atendesse às necessidades da Empresa Cliente por uma infraestrutura capaz de fornecer um processamento otimizado e escalável, com disponibilização dos dados para múltiplas funções em uma plataforma de nuvem robusta e confiável, foi proposta a criação de um *Data Lakehouse*. O *Data Lakehouse* tem a capacidade de armazenar dados de diferentes origens e formatos, estruturados ou não, permitindo o uso de formatos de arquivo otimizados para leitura e escrita rápida. Dessa forma, é possível fornecer uma estrutura de dados contendo informações brutas para serem utilizadas da maneira que for conveniente (Data Lake) seja em modelos de IA ou análises exploratórias, além de fornecer uma estrutura analítica (*Data Warehouse)* para alimentar os relatórios visuais já utilizados pela Empresa Cliente. Para o armazenamento dos dados no *Data Lakehouse* o Azure Data Lake Storage é utilizado. Como se trata de um projeto de migração, os dados da estrutura analítica do *Data Lakehouse* serão modelados como já estão na arquitetura atual da empresa cliente, utilizando o modelo estrela em que têm-se tabelas fato centrais com os fatos do negócio como Vendas, Logística ou Manutenção e tabelas de dimensão com os detalhes dos fatos como Funcionários, Motoristas ou Clientes.

A proposta de utilização do *Data Lakehouse* em um serviço de armazenamento de objetos habilita a arquitetura a utilizar o *framework Delta Lake* com a arquitetura medalhão como padrão de design dos dados. O *Delta Lake* permite que os dados possuam, junto de suas estruturas de pastas no armazenamento de objetos, informações relacionadas ao tempo de vida do dado, como quando ele foi registrado, se sofreu alteração, se foi deletado, habilitando uma camada ACID de confiança nos dados, como para os Bancos de Dados relacionais, porém em uma estrutura otimizada de processamento de arquivos mais rápida e otimizada para consultas analíticas. Além disso, a arquitetura medalhão organiza as camadas dos dados processados a fim de permitir o

uso em diferentes contextos, desde os dados brutos conforme originam-se nas fontes na camada *Bronze*, os dados pré-processados e limpos na camada *Silver* até os dados preparados analiticamente na camada *Gold*, habilitando a Empresa Cliente a usar os dados da maneira que for mais conveniente.

Para o tratamento dos dados, propôs-se a utilização da metodologia de processamento distribuído de dados com o Apache Spark no Azure Databricks para realizar o controle de operações e distribuição das cargas de trabalho a fim de otimizar o processo. A partir da utilização do Databricks, a Empresa Cliente ganha uma poderosa ferramenta de processamento, capaz de escalar seus processos de acordo com o volume de dados armazenados no Azure Data Lake Storage. Dessa forma, é possível que, ao passo que o volume de dados aumente, a ferramenta de processamento não se torne um gargalo na arquitetura, somente necessitando de uma configuração para atender à nova demanda.

A proposta contempla também a metodologia de controle de versões para registrar o histórico das alterações realizadas nos códigos da arquitetura, bem como permitir a implementação de novas funcionalidades sem interferir no processo já operante. Técnicas de criptografia também são utilizadas para proteger os *tokens* e senhas utilizadas nos serviços para comunicação e acesso. Por fim, a visualização será realizada com a metodologia de *Business Intelligence Reports*, com relatórios visuais contendo informações valiosas para a Empresa Cliente que auxiliam no desempenho e na tomada de decisão do negócio.

5.1 NOVA ARQUITETURA PROPOSTA

Conforme apresentado na seção anterior, a arquitetura de dados atual da Empresa Cliente, puramente baseada em sistemas de Business Intelligence, passou a não atender mais à demanda de dados atual da organização. Com isso, um trabalho de estudo foi realizado pela BIX TECNOLOGIA para entender como a arquitetura atual estava desenvolvida e de que forma seria possível implementar uma nova arquitetura mais robusta, eficiente e com maior poder de processamento, utilizando as mesmas fontes de dados atuais da empresa e mantendo a estrutura de relatórios visuais já utilizados. Uma possibilidade discutida foi o update das contas atuais do Microsoft Power BI Pro para Microsoft Power BI Premium, porém, a Empresa Cliente indicou em reuniões que gostaria de expandir a maturidade em dados para atender não somente demandas analíticas, mas também à utilização de Inteligência Artificial. Dessa forma, uma nova arquitetura com maior nível de maturidade e robustez foi proposta.

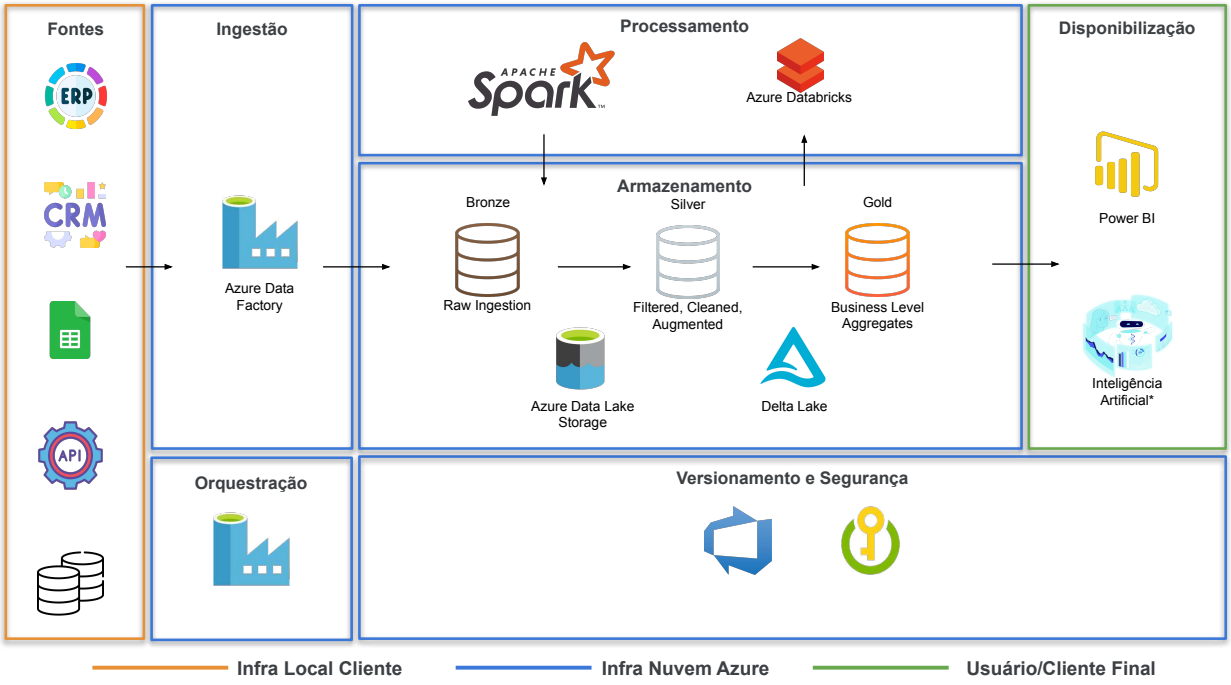
Para atender à demanda atual, foi proposta uma migração da arquitetura baseada em serviços de *BI* para uma nova baseada em serviços de nuvem. Como a Empresa Cliente já utiliza os serviços da Microsoft em suas operações, foi proposta a construção de toda a arquitetura utilizando os serviços do Microsoft Azure, o serviço

de nuvem da Microsoft Corporation. A Figura 16 apresenta a estrutura da arquitetura proposta, contando com serviços específicos para cada etapa do fluxo de dados. Estão presentes serviços utilizados para realizar a ingestão dos dados das fontes de origem, para realizar a orquestração das etapas do fluxo, para armazenamento dos dados em camadas, para processamento dos dados de maneira distribuída, para versionamento dos códigos utilizados na arquitetura e para segurança das chaves e segredos utilizados no fluxo. Por fim, ainda há a camada de disponibilização dos dados, com a mesma ferramenta já existente na arquitetura atual da Empresa Cliente, porém, agora somente responsável pela etapa de visualização dos dados e não mais pelo processamento.

A nova arquitetura foi proposta com o objetivo de permitir um processo confiável de extração, transformação e carga (ETL) dos dados que fazem parte das operações da empresa. Os serviços propostos possuem uma série de benefícios em relação à arquitetura anterior, especialmente em relação ao poder de processamento do Azure Databricks, que, além de realizar o processamento distribuído dos dados, possui integrações com diversos serviços. Objetivou-se com a nova arquitetura implementar uma estrutura de processamento de dados robusta, confiável e escalável que atenda ao crescente volume de dados da empresa cliente sem haver a necessidade de grandes mudanças, apenas necessitando de pequenas configurações.

Os elementos da arquitetura proposta serão brevemente apresentados com suas características a seguir.

Figura 16 – Nova Arquitetura de Dados Proposta.



Fonte: Autor.

5.1.1 Serviços da Microsoft Azure

A Microsoft Azure dispõe de uma infinidade de serviços com diferentes funcionalidades, como serviços de armazenamento e processamento de dados, criação de máquinas virtuais e contêineres utilizando o poder de computação em nuvem, controle de rede e tráfego online de maneira escalável, infraestrutura de TI, segurança da informação, criação de *pipelines* de fluxo de dados, entre outros diversos serviços fornecidos. No contexto deste projeto, essencialmente, cinco serviços da Microsoft Azure foram necessários para estabelecer toda a arquitetura, cada um com suas particularidades e funcionalidades. Os serviços propostos foram:

- **Azure Data Factory** - Serviço de integração e orquestração de *pipelines*, voltado para realizar processos de ETL. O Data Factory é completamente gerenciado, sem haver a necessidade de realizar configurações de rede e protocolos, esta tarefa sendo realizada pelo próprio serviço. Com o Data Factory, é possível conectar as diferentes fontes de dados da Empresa Cliente e orquestrar todo o processo de Extração dos dados para armazená-los no serviço de armazenamento, Transformação dos dados conectando-se ao serviço de processamento e realizar a Carga na ferramenta de visualização, configurando todo o processo de ETL. O Data Factory opera como o cérebro da arquitetura, realizando os comandos de cada etapa do *pipeline*.

- **Azure Data Lake Storage Gen2** - Serviço de armazenamento de dados em formato de objeto utilizado para guardar os dados advindos das fontes da Empresa Cliente, bem como armazenar as informações nas camadas *Bronze*, *silver* e *Gold* conforme a arquitetura medalhão. O armazenamento em objetos permite salvar dados em formatos estruturados ou não e a criação de um *Data Lakehouse*, objetivo principal desde projeto, salvando os arquivos em formato otimizado para leitura em cargas analíticas de dados OLAP.

- **Azure Databricks** - O Azure Databricks é a plataforma de processamento de dados em escala com habilitação para cargas de trabalho de análise e Inteligência Artificial. O serviço opera como o coração da arquitetura, contectando-se ao Data Lake Storage e buscando os dados brutos vindos das fontes, processando-os e entregando-os em suas respectivas camadas de armazenamento. A característica mais importante do Azure Databricks é o processamento distribuído utilizando o Spark para realizar um processamento escalável e forma eficiente e otimizada. O Databricks conecta-se com diversos serviços externos e habilita a utilização de modelos de IA com os dados da arquitetura de forma fácil e rápida de configurar.

- **Azure DevOps** - Serviço de versionamento e gerenciamento de projetos de software. O Azure Devops dispõe de funcionalidades de automação de processos

de distribuição de código e gerenciamento de lançamento de versões, porém, neste projeto, será utilizada somente a funcionalidade de versionamento de código. Com o DevOps, é possível salvar o código do projeto e mantê-lo operando em produção ao mesmo tempo em que são realizadas alterações para melhorias e manutenções, garantindo o funcionamento pleno do projeto.

- **Azure Key Vault** - Serviço de segurança voltado para armazenamento de segredos. Segredos são valores que não podem ser expostos como por exemplo *tokens* de *APIs*, senhas de serviços ou chaves de conexão de serviços. O Key Vault integra-se aos demais serviços da Microsoft Azure, sendo possível resgatar os valores contidos em cada segredo sem expor seu conteúdo de maneira explicida, criptografando o conteúdo do segredo para garantir segurança e isolamento destes valores.

5.2 CRONOGRAMA

Para o desenvolvimento do projeto, um cronograma previsto inicial foi estipulado levando-se em consideração o volume de trabalho necessário e a quantidade de relatórios a serem migrados. A Figura 17 apresenta o cronograma com as atividades realizadas ao longo dos meses do projeto. Foram previstas atividades de ambientação e configuração dos serviços da Microsoft Azure utilizados, bem como um longo período de implementação dos *pipelines* de dados e, por fim, ajustes e validações das entregas.

Figura 17 – Cronograma previsto para o projeto.

Atividades Previstas	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro
	Número de semanas						
	4	5	4	5	4	4	2
Ambientação e Acessos							
Estruturação da arquitetura na Azure							
Implementação dos *pipelines*							
Ajustes na arquitetura/*pipelines*/tabelas							
Implementação de otimizadores							
Ajustes finais							

Fonte: Autor.

6 DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS OBTIDOS

Neste capítulo, serão apresentadas as etapas de desenvolvimento do projeto completo de migração de arquitetura de dados. As etapas para elaboração de cada elemento do processo serão descritas a fim de elucidar como foi a implementação, os elementos técnicos e fundamentos envolvidos no processo e de que forma impactaram a Empresa Cliente. Também serão apresentados os resultados obtidos com a nova arquitetura, especialmente em relação à maturidade da infraestrutura de dados atingida pela Empresa Cliente. Todo o desenvolvimento da nova arquitetura de dados e implementação dos componentes da mesma apresentados neste capítulo foram desenvolvidos integralmente pelo autor deste documento.

6.1 AMBIENTAÇÃO E ESTRUTURAÇÃO DA ARQUITETURA

Como primeira etapa do projeto, foi realizada toda a ambientação do mesmo. Inicialmente, foram alinhados com a Empresa Cliente os acessos aos ambientes necessários para a configuração e desenvolvimento do projeto. Duas contas internas foram disponibilizadas para acesso tanto ao ambiente da Microsoft Azure quanto ao ambiente do Power BI para realizar, respectivamente, a criação dos novos *pipelines* para os dados e a migração dos relatórios visuais baseados na arquitetura antiga para a arquitetura nova, ficando cada um dos times da BIX TECNOLOGIA com uma das contas.

6.1.1 Acesso aos Ambientes de Nuvem da Empresa Cliente

A Empresa Cliente utiliza-se dos serviços de uma terceirizada para faturar as ferramentas da Microsoft Corporation utilizadas em suas operações. Por conta disso, para ter acesso ao ambiente da Microsoft Azure e realizar as implementações do projeto, foi necessário aguardar a liberação por parte da empresa terceirizada, o que culminou em um pequeno atraso no início do projeto. Porém, liberados os acessos ao ambiente Azure, todas as configurações iniciais e dos serviços foram possíveis.

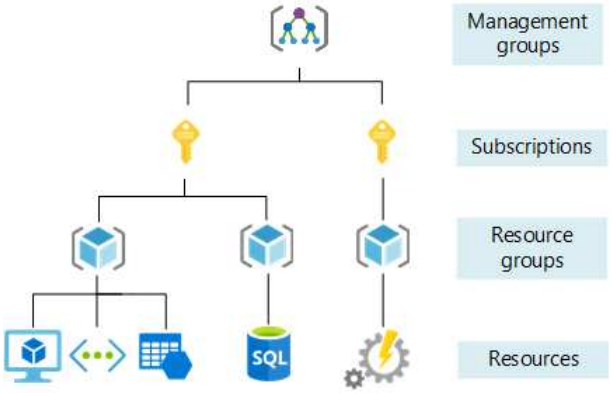
6.1.2 Hierarquia de gestão da Azure

A fim de permitir que uma mesma organização consiga gerenciar todas as suas aplicações utilizando os serviços e os usuários dessas aplicações, a Microsoft Azure possui uma estrutura hierárquica para organizar os recursos utilizados pelos clientes. A Figura 18 apresenta a estrutura organizacional da Azure. No topo, estão os Grupos de Gerência (*Management groups*), que configuram os acessos e permissões dos usuários para acessar determinados recursos e funcionalidades da Microsoft Azure. Já as Assinaturas (*subscriptions*) são criadas para permitir a um determinado usuário da

Azure criar recursos e associar esses recursos com a assinatura, criando a conexão entre as entidades. Os Grupos de Recursos (*Resource groups*) são contêineres que isolam os Recursos (*Resources*) da Azure utilizados para melhor gestão e controle de acesso. No sistema hierárquico da Azure, os recursos são herdados; logo, se um usuário é associado a uma *Subscription*, ele herda todas as permissões abaixo como *Resource groups* e *Resources* efetivamente.

Para o desenvolvimento do projeto, uma *Subscription* foi disponibilizada pela empresa terceirizada que realiza o faturamento da Empresa Cliente, à qual as duas contas de usuário utilizadas durante o desenvolvimento do projeto foram adicionadas para acesso aos *Resources* da Microsoft Azure, a fim de estruturar todos os serviços necessários na nova arquitetura.

Figura 18 – Hierarquia gerencial da Microsoft Azure.



Fonte: microsoft.com.

6.1.3 Estruturação e Criação dos Serviços

A partir da *Subscription* criada pela empresa terceirizada, foi possível iniciar a etapa de estruturação do projeto, criando os serviços necessários para a execução completa dos pipelines, com os elementos da nova arquitetura apresentada na Figura 16. Na *Subscription* um *Resource group* também foi criado a fim de organizar todos os *Resources* constituintes da nova arquitetura. Toda a criação e configuração dos serviços utilizados foram realizadas através do Portal Azure, uma interface *web* para controle e gerenciamento dos serviços oferecidos pela Microsoft Azure.

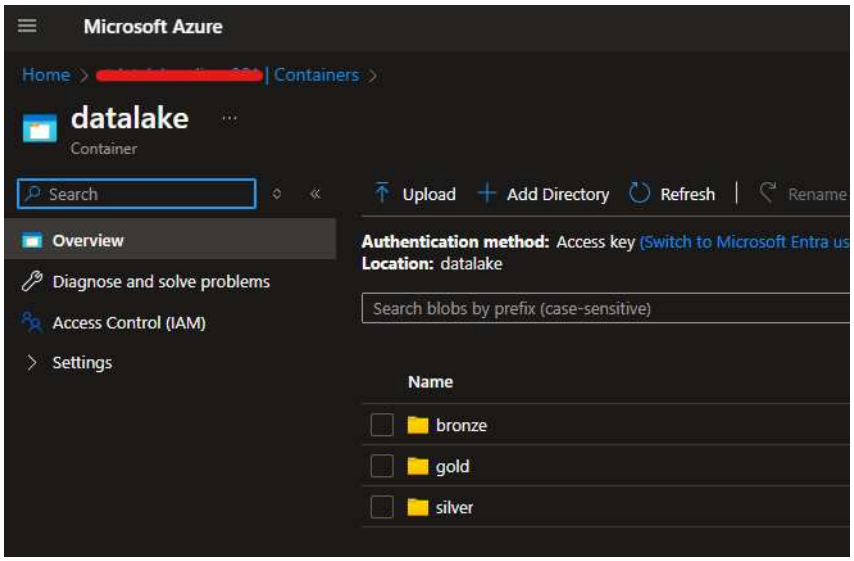
6.1.3.1 Data Lake Storage Gen2

O primeiro serviço da Microsoft Azure configurado para compor a nova arquitetura foi o Data Lake Storage Gen2. Como já apresentado anteriormente, o Azure Data Lake Storage Gen2 é um serviço de armazenamento de dados em formato de objetos. Os objetos armazenados no Data Lake Storage podem ser de diferentes tipos

e formatos, desde imagens, vídeos e áudios até arquivos estruturados como *JSON* ou *XML*. Um formato muito utilizado em estruturas analíticas é *Parquet*, criado pela Apache Foundation. O *Parquet* é utilizado para salvar dados em formato de coluna, sendo este formato o cerne dos sistemas OLAP para cargas de trabalho analíticas. Além deste fato, os arquivos *Parquet* são otimizados e comprimidos para oferecer uma melhor relação de custo e tempo de processamento do que arquivos convencionais como *CSV*, por exemplo. Além de permitir o uso de dados em formato otimizado para lidar com grandes volumes de dados com custo baixo e boa performance de processamento, o Azure Data Lake Storage também oferece a possibilidade de criação de um *Delta Lake* muito otimizado. O *Delta Lake* é um *framework* muito bem aplicado em estruturas analíticas, pois permite a confiança das propriedades ACID dos bancos de dados relacionais em sistemas OLAP, muito mais rápidos e eficientes para dados analíticos em grande escala.

Para configurar o Data Lake Storage, uma *Storage Account* (conta de armazenamento) foi criada dentro do serviço e, nesta, um contêiner chamado **datalake** foi criado para conter os dados da nova arquitetura. Dentro do contêiner, foram criados "subdiretórios" chamados de **blobs** para conter os dados de cada camada da arquitetura medalhão proposta. A Figura 19 apresenta a estrutura criada na Azure para conter os dados, as camadas *bronze*, *silver* e *gold* sendo alocadas em *blobs* específicos para cada.

Figura 19 – Contêner do *Data Lake* com as camadas.



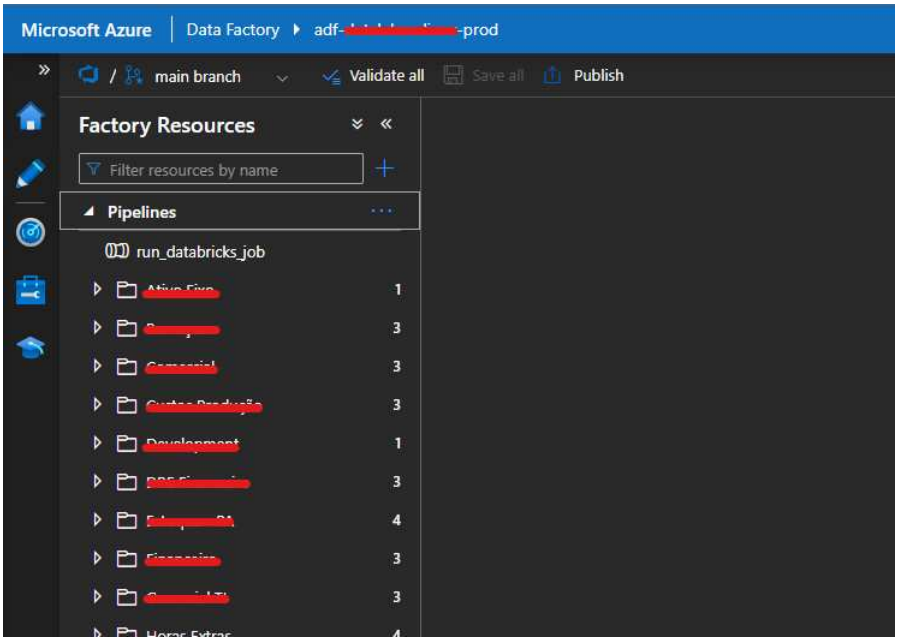
Fonte: Autor.

6.1.3.2 Data Factory

O segundo serviço configurado foi o Azure Data Factory. O Azure Data Factory é um serviço de criação e orquestração de *pipelines* (fluxos) de dados para criação de processos de Extração, Transformação e Carga. O Data Factory conecta-se a uma infinidade de fontes de dados, sendo possível realizar a extração desde arquivos locais em serviços da Microsoft, bancos de dados comerciais até outros serviços de nuvem como o GCP ou a AWS. A Figura 20 apresenta a estrutura inicial do Data Factory, onde é possível realizar a gestão dos *pipelines* criados, editando-os de acordo com o fluxo necessário dos dados de cada *pipeline*. Também é possível, no menu de edição, adicionar e gerenciar as fontes e destinos dos dados brutos extraídos pelo Data Factory. Neste projeto, foram adotadas como fontes dos dados sistemas de arquivos internos da Microsoft, como o Microsoft SharePoint e bancos de dados relacionais Oracle de sistemas utilizados pela Empresa Cliente, todas conectadas através das funcionalidades do Azure Data Factory. Os pipelines foram configurados para terem como destino dos dados brutos o Data Lake Storage Gen2.

Os *pipelines* de dados foram configurados para extrair os dados brutos das fontes, salvá-los no Data Lake Storage Gen2 e, a partir daí, seguir para a transformação na ferramenta de processamento de dados escolhida (Databricks) e, por fim, realizar a carga e atualização dos modelos semânticos do Power BI com os dados atualizados. Com o Data Factory, é possível realizar o monitoramento de todo o processo através do *monitor* (*Gauge* da Figura 20), onde são apresentadas as informações de início, meio e fim do processo, bem como *logs* de operações e erros no decorrer do processo. Também são configurados no Data Factory os *Linked Services*, estruturas internas do serviço para conectar-se às fontes externas e a serviços internos da Microsoft Azure. Estes serão apresentados adiante neste documento.

Figura 20 – Estrutura inicial do Data Factory.



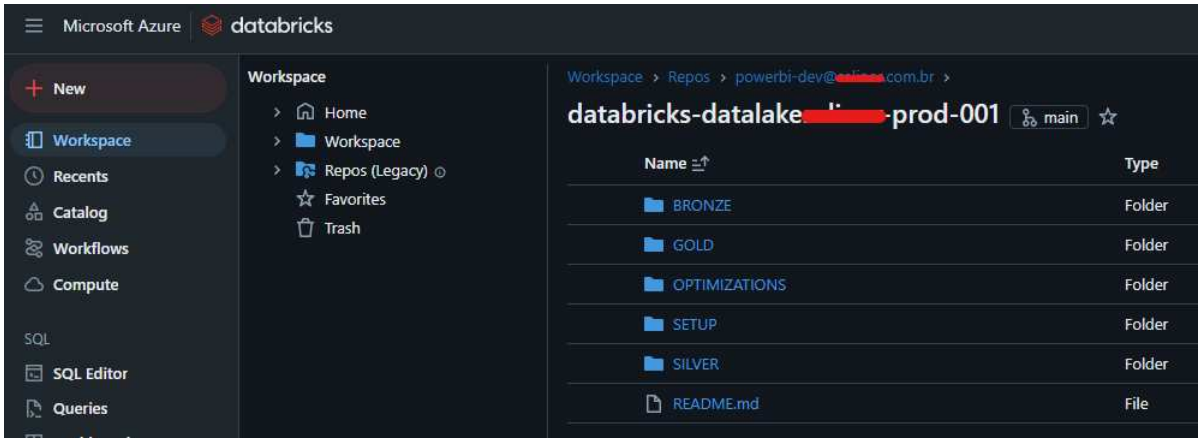
Fonte: Autor.

6.1.3.3 Databricks

O terceiro e principal elemento da nova arquitetura configurado foi o Azure Databricks. O Azure Databricks é uma plataforma de processamento de dados distribuído com clusters e utilizando o Spark como *framework* de processamento. Na nova arquitetura, o Databricks é o responsável por toda a etapa de transformação dos dados extraídos das fontes, realizando o processamento nas camadas de dados brutos, *bronze*, *silver* e *gold*. O Databricks conecta-se ao Azure Data Lake Storage Gen 2 a fim de realizar a leitura dos dados brutos advindos das origens, realizar o processamento e, por fim, escrever os dados processados nos *blobs* de cada camada do *Data Lakehouse*.

Conforme presente na Figura 21, uma instância do Azure Databricks foi criada para ser utilizada no *data lakehouse* da Empresa Cliente. Como observa-se, o Azure Databricks possui subdiretórios criados para cada uma das camadas dentro das quais são alocados os códigos necessários para o tratamento dos dados. Os códigos são escritos em *notebooks*, estruturas desenvolvidas para escrever códigos de maneira interativa e dinâmica, podendo-se observar os resultados durante o desenvolvimento, facilitando a implementação e permitindo que o código seja desenvolvido de maneira modular e sequencial. Adiante neste documento serão apresentadas as estruturas dos notebooks criados tanto para a criação das tabelas, quanto para o processamento dos dados nas camadas.

Figura 21 – Estrutura inicial do Databricks.



Fonte: Autor.

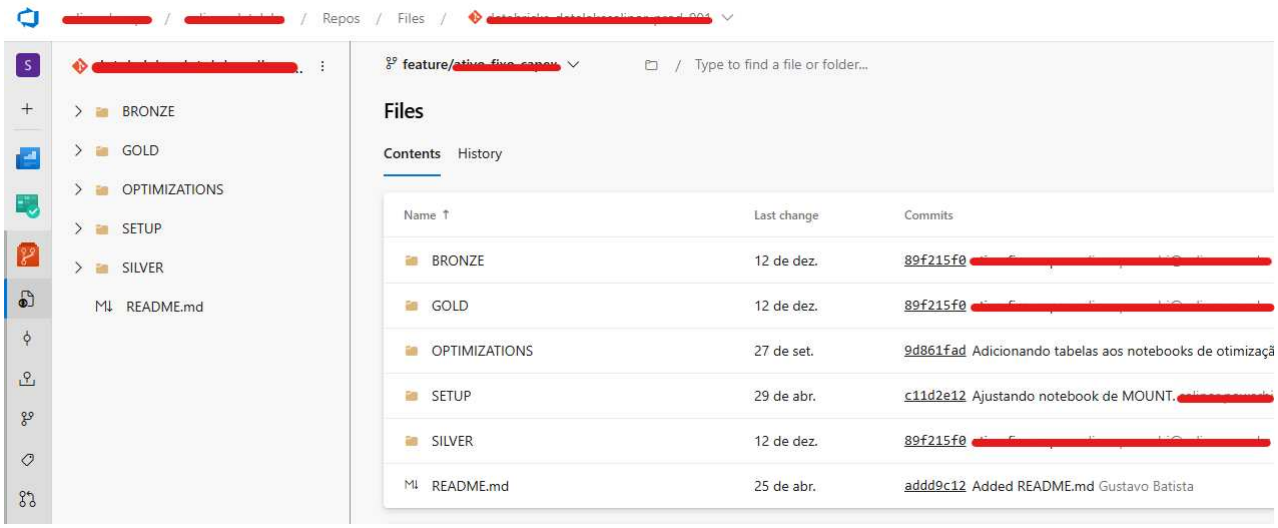
Para que o Azure Databricks conecte-se ao Azure Data Lake Storage Gen 2, é necessário que seja realizada uma etapa de configuração do serviço de armazenamento de objetos que será utilizado para armazenar os dados. Um *notebook* de montagem (localizado na pasta *setup* da Figura 21) foi criado para estabelecer a conexão entre o Databricks e o Data Lake Storage. Além disso, é necessário configurar o *Unity Catalog*, um serviço do Databricks para governança e controle de dados utilizado para gerenciamento das tabelas. Realizadas estas configurações, o Databricks está habilitado para a implementação dos códigos responsáveis pela transformação dos dados do *Data Lakehouse*.

6.1.3.4 DevOps

O quarto serviço configurado foi o Azure DevOps. O Azure DevOps é um serviço de controle de versão, gerenciamento e testes de projetos de software, voltado para o auxílio no desenvolvimento destes. Das funcionalidades oferecidas, neste projeto somente a ferramenta de versionamento de código foi utilizada, chamada de Repos (abreviação de “Repository”). O Repos funciona com repositórios que guardam arquivos com os códigos responsáveis pelas funcionalidades, especialmente dos serviços Azure Data Factory e Databricks. Assim que o repositório é criado, os códigos são armazenados em um ramo principal, chamado de *main*. A partir do ramo principal, é possível criar novos ramos com uma cópia do código original e alterá-lo com ajustes ou novas funcionalidades sem afetar o código principal, além de permitir o retorno do código para uma versão anterior em caso de algum erro ou outras eventualidades. Neste projeto, o Repos foi utilizado para armazenar os códigos Python de transformação dos dados em cada camada do data lake no Databricks, bem como o controle da estrutura dos *pipelines* de fluxo dos dados no Data Factory. A Figura 22 apresenta a estrutura

do repositório do Databricks, com cada uma das camadas da arquitetura.

Figura 22 – Repositório dos códigos do Databricks no Azure DevOps.

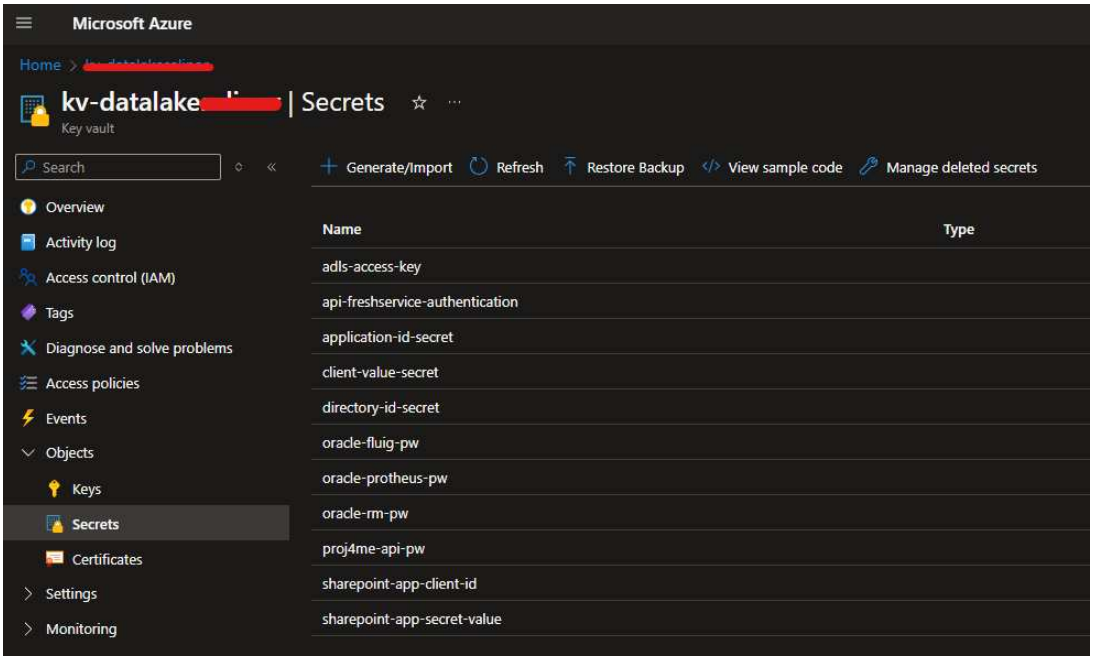


Fonte: Autor.

6.1.3.5 Key Vault

O último serviço configurado da nova arquitetura foi o Azure Key Vault. O Key Vault atua como uma ferramenta de segurança, conectando-se aos demais serviços da Azure para fornecer informações sensíveis de maneira segura e criptografada. Através do Key Vault no Portal Azure, é possível criar segredos para senhas, tokens, chaves de *API*, credenciais de contas de serviço da Azure, entre outros valores de maneira sigilosa. A Figura 23 apresenta a estrutura de configuração dos segredos do Key Vault, entre eles senhas, chaves e segredos sigilosos que não podem ser expostos nas operações dos serviços. Os segredos criados possuem uma validade de funcionamento e um controle de versões, aumentando ainda mais a segurança oferecida para a arquitetura.

Figura 23 – Tela de configuração de segredos do Key Vault.



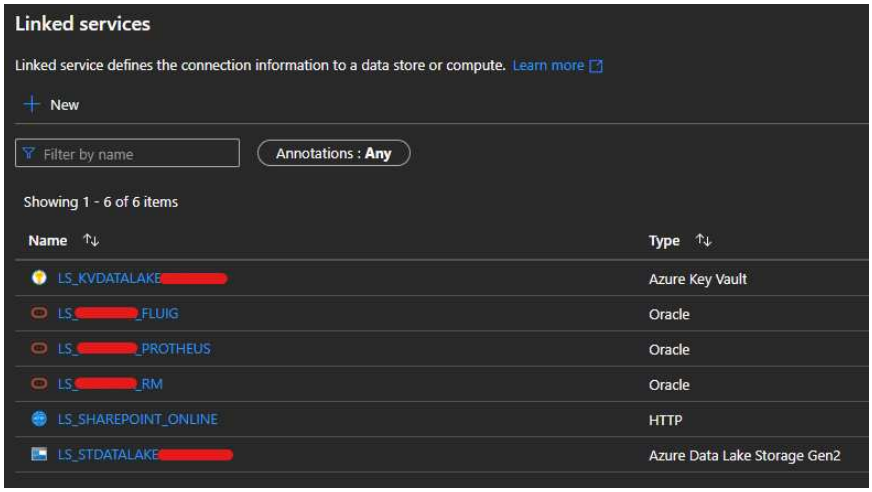
Fonte: Autor.

6.1.4 Integração entre os serviços e fontes de dados (*Linked Services)*

Em uma arquitetura de dados, especialmente quando adotados os processos de ETL, é necessário que inicialmente sejam realizadas conexões com as origens dos dados para que os mesmos possam ser ingeridos e processados em cada etapa. Nas arquiteturas baseadas em Azure Data Factory, um elemento específico é utilizado, sendo ele o *Linked Service*, ou Serviço Vinculado. Os *Linked Services* são criados para estabelecer conexões entre o Azure Data Factory e fontes de dados externas à Azure ou ainda serviços dentro da própria plataforma de serviços da Microsoft Azure. Como observa-se na Figura 24, diferentes *linked services* são criados para as conexões, desde os específicos para ligação com bancos de dados externos ou a *APIs* externas à Azure, como os bancos de dados Oracle dos sistemas utilizados pela Empresa Cliente ou *API* do Microsoft SharePoint com dados em tabelas Excel gerenciadas pela organização. Também são criados *linked services* para a conexão com os serviços da própria Azure, como o Azure Data Lake Storage Gen2 para armazenar os dados extraídos das fontes ou ao Key Vault para resgatar as senhas e credenciais para conexão com os bancos de dados com as *APIs*. Os protocolos utilizados pelos *linked services* são abstraídos pela própria infraestrutura de nuvem da Microsoft Azure, porém, entende-se pelos padrões de comunicação adotados comercialmente que protocolos como ODBC/JDBC para comunicação com bancos de dados, HTTP para APIs REST ou ainda protocolos proprietários devem ser empregados para estabelecer as

conexões.

Figura 24 – *Linked Services* utlizados na arquitetura.



Fonte: Autor.

6.1.5 Conexão Entre Origens de Dados *On-Premise* e Data Factory (*Integration Runtime*)

É comum que se encontre em grandes organizações com uma infraestrutura de tecnologia da informação no local (*on-premise*), bancos de dados alocados nessas infraestruturas próprias. Na maioria dos casos, essas infraestruturas comunicam-se em uma rede interna, sem acesso direto à internet pública, principalmente por motivos de segurança institucional. Nesses casos, é necessário que sejam utilizadas ferramentas para realizar a conexão entre as infraestruturas *on-premise* e a Microsoft Azure, a fim de que os dados possam ser acessados através da internet. Para este fim, o Azure Data Factory utiliza-se dos *Integration Runtimes (IR)*. O *integration runtime* é uma estrutura de computação que permite estabelecer a comunicação entre redes públicas e privadas para realizar a transferência de dados entre essas redes. Especificamente neste projeto, o *Self-Hosted Integration Runtime* (*Integration Runtime* auto-hospedado) foi utilizado. O *Self-hosted integration runtime* funciona como uma porta de entrada, ou um *gateway*, que opera em um computador conectado ao servidor em que o banco de dados está alocado, conectando-se aos recursos do servidor, especialmente aos relacionados com a movimentação de dados. Neste projeto, um *Self-Hosted Integration Runtime* foi instalado no servidor da Empresa Cliente com o auxílio da equipe de TI da mesma, habilitando o Data Factory a acessar os dados alocados nos bancos de dados Oracle que armazenam os dados dos sistemas adotados pela Empresa Cliente, utilizando, para isso, os *linked services* criados anteriormente.

Com todas as estruturas apresentadas anteriormente devidamente configuradas

e testadas, foi possível iniciar a implementação dos *pipelines* de extração, transformação e carga dos dados de cada um dos relatórios da Empresa Cliente. A seguir, serão descritos os detalhes da implementação dos pipelines, desde a ingestão dos dados brutos das origens até a entrega das tabelas finais aos relatórios do Power BI.

6.2 DESENVOLVIMENTO DOS PIPELINES DOS RELATÓRIOS

Nesta etapa do documento, serão apresentados os detalhes de implementação dos *pipelines* de dados criados no projeto. Serão descritas as etapas necessárias para o fluxo completo dos dados, desde a ingestão dos dados brutos com os conectores configurados inicialmente via Data Factory e armazenamento na camada bruta de dados, bem como o processamento dos dados em cada camada da nova arquitetura e o armazenamento dos dados processados no Azure Data Lake Storage.

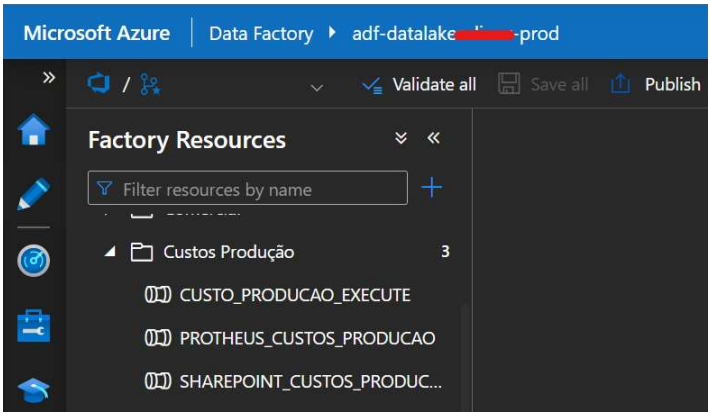
6.2.1 Pipelines de orquestração

Ao todo, 17 *pipelines* foram implementados para migrar todos os relatórios do Power BI utilizados pela empresa cliente em suas análises. Os *pipelines* criados processam dados para diferentes fins, como relatórios de RH, Logística, Comercial até análises mais específicas de times de desenvolvimento interno. Por vezes, o mesmo *pipeline* foi utilizado para diferentes relatórios pela afinidade de suas tabelas, mas todas as tabelas utilizadas atualmente pela empresa em sua estrutura analítica foram migradas para a nova arquitetura. Para o desenvolvimento, optou-se por migrar cada um dos relatórios separadamente por questão de facilidade na validação, tanto pelo time de *Business Intelligence* da BIX, quanto pela equipe da Empresa Cliente. Ao todo, 20 relatórios foram migrados para a nova arquitetura.

Os *pipelines* podem ser entendidos como o cérebro orquestrador de todas as etapas de um fluxo de dados. A partir dessas estruturas, é possível organizar cada etapa, como copiar os dados das origens para o *Data Lake*, acionar a transformação dos dados no Databricks e atualizar os dados no Power BI, utilizando todas as conexões e integrações disponibilizadas pelo Azure Data Factory apresentadas anteriormente.

Conforme apresentado na Figura 20, os *pipelines* foram alocados em diretórios do Data Factory, sendo um diretório para cada *pipeline* implementado. De acordo com as origens dos dados de cada relatório, diferentes *sub-pipelines* para cada origem foram criados para a extração dos dados brutos e um *pipeline* principal foi criado para o gerenciamento de todo o fluxo, ordenando as etapas quando o fluxo é acionado. A figura 25 apresenta a estrutura base dos diretórios dos pipelines, onde é possível encontrar os *sub-pipelines* de cada origem do referido relatório e o *pipeline* principal de gerenciamento do fluxo de operação com o sufixo *EXECUTE*.

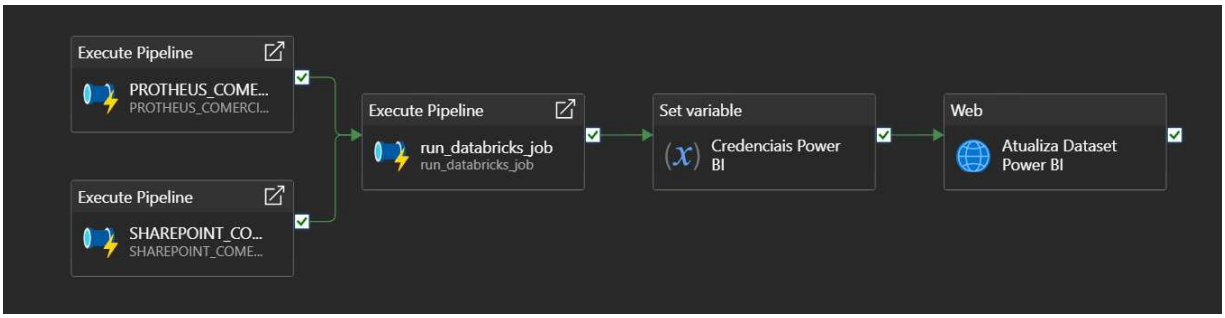
Figura 25 – Estrutura dos diretórios dos pipelines.



Fonte: Autor.

Os pipelines do Data Factory são construídos utilizando-se de uma estrutura de blocos intuitiva e altamente modularizável. Estão disponíveis blocos com as mais diversas funções, como operações de cópia e movimentação de dados, criação de variáveis, operações lógicas de *if/else*, *loops* de iteração, até operações mais complexas de integração com funções externas escritas em diferentes linguagens, conexão com serviços do ecossistema Hadoop e funções voltadas à Inteligência Artificial. A Figura 26 apresenta a estrutura do *pipeline* principal *EXECUTE* de um dos relatórios implementados. Nela, é possível observar a presença de blocos de *sub-pipelines* que realizam a ingestão dos dados de cada uma das fontes de dados do relatório, um bloco de *sub-pipeline* específico para acionar o processamento do Databricks e um bloco de atividade WEB para ativar a *API* do Power BI e atualizar os dados do relatório final. Essa mesma estrutura foi replicada em todos os 17 *pipelines* implementados para orquestrar o fluxo completo e gerenciar todas as etapas executadas.

Figura 26 – Exemplo de *pipeline* principal de um dos relatórios implementados.



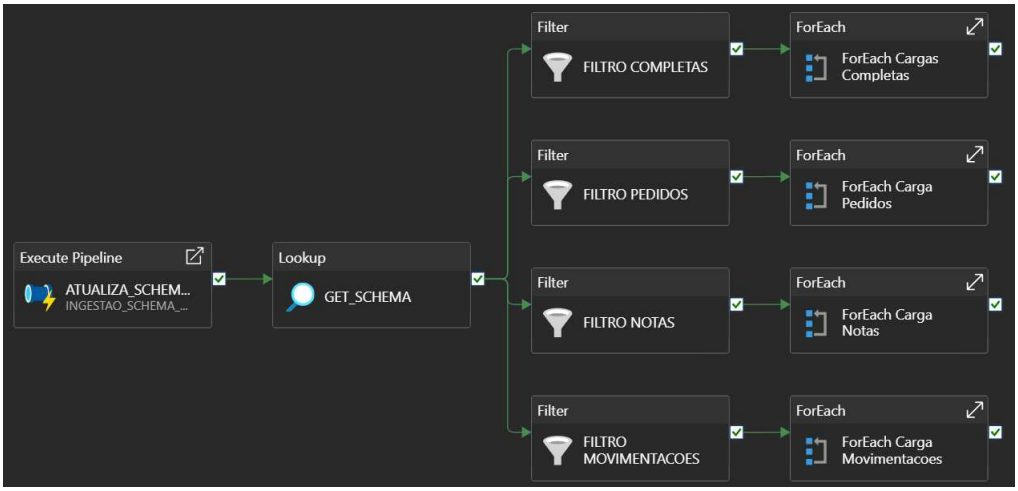
Fonte: Autor.

6.2.1.1 Ingestão dos dados brutos

Para realizar a ingestão dos dados brutos vindos das origens, como já apresentado na Figura 26, são utilizados *sub-pipelines* para cada fonte de dados. Os *sub-pipelines* de cada uma das fontes possuem blocos específicos para realizar a extração dos dados brutos das origens e trazê-los para o *Data Lake* da Empresa Cliente. As Figuras 27 e 28 abaixo apresentam exemplos da estrutura utilizada em duas origens de dados distintas.

A Figura 27 possui a estrutura de ingestão dos dados dos bancos de dados relacionais dos sistemas utilizados pela Empresa Cliente, inicialmente realizando a atualização do *Schema* (colunas das tabelas ingeridas). Em seguida, são distribuídas em trabalhos paralelos de acordo com a natureza de ingestão dos dados, podendo ser incrementais ou cargas completas. As cargas completas realizam sempre a cópia da tabela integral da origem, buscando todos os registros e salvando-os no *Data Lake*. Já as cargas incrementais realizam a ingestão somente de uma porção dos dados, utilizando um parâmetro de data para definir a partir de que período os dados devem ser atualizados. Essa abordagem é utilizada em grandes tabelas (acima de 200 mil registros) para otimizar a performance da ingestão dos dados, utilizando-se de um período retroativo adequado de acordo com a natureza de atualização dos dados. Dentro de cada bloco *forEach* estão alocados blocos de atividade de cópia que utilizam-se do *Integration Runtime* instalado na máquina da Empresa Cliente e dos *Linked Services* para buscar os dados. Com o *Integration Runtime*, é possível escrever uma consulta SQL na atividade de cópia do Data Factory, que será executada diretamente no banco de dados alocado na infraestrutura local da Empresa Cliente, buscando os dados e ingerindo-os no *Data Lake*. Para os casos de tabelas ingeridas completas, sempre que o *pipeline* é acionado, a atividade de cópia realiza uma busca no banco e salva um novo arquivo com a tabela completa conforme está o estado atual da mesma no banco de dados. Já os casos de tabelas incrementais, a cada ativação do *pipeline* uma parte da tabela é salva em um novo arquivo de acordo com o parâmetro de tempo retroativo para busca dos dados atualizados e, posteriormente, via Databricks, estas partições são processadas para definir e executar as atualizações realizadas na tabela, sejam deleções, atualizações ou inserções.

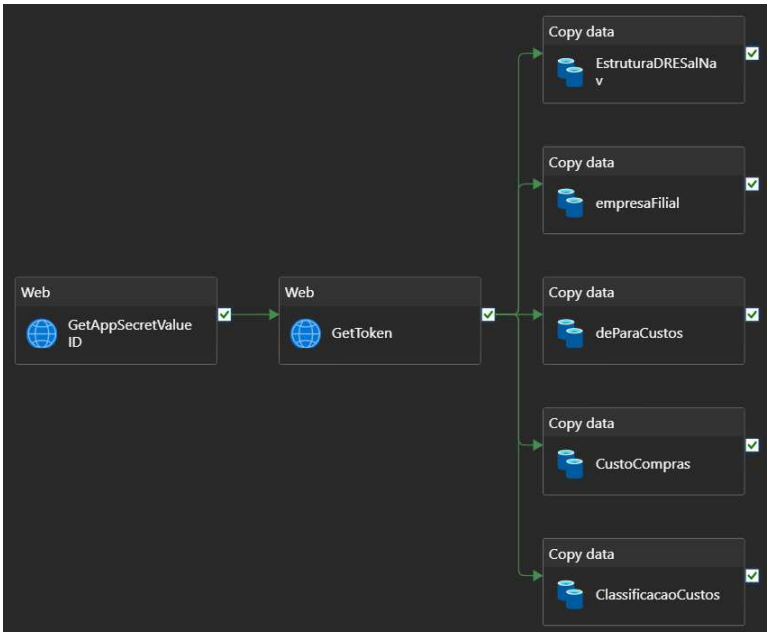
Figura 27 – Estrutura do *pipeline* de ingestão dos bancos de dados relacionais.



Fonte: Autor.

Já Figura 28 apresenta a estrutura de ingestão dos dados do Microsoft SharePoint. A mesma é ligeiramente mais simples que a anteriormente apresentada, com apenas blocos para conexão WEB com a *API* do Microsoft SharePoint. A partir do *endpoint* da *API*, é possível especificar o arquivo Excel contendo as tabelas que se deseja ingerir e utilizar uma atividade de cópia para realizar a duplicação dos dados para o *Data Lake*.

Figura 28 – Estrutura do *pipeline* de ingestão das tabelas do Microsoft SharePoint.



Fonte: Autor.

Essencialmente, as duas estruturas de *sub-pipelines* apresentadas foram utilizadas no decorrer de todo o projeto para implementar a ingestão de dados dos *pipelines* de todos os relatórios migrados. Bastou realizar cópias das mesmas, adaptando-as de acordo com as tabelas necessárias em cada *pipeline*.

6.2.1.2 Ingestão dos dados de *APIS*

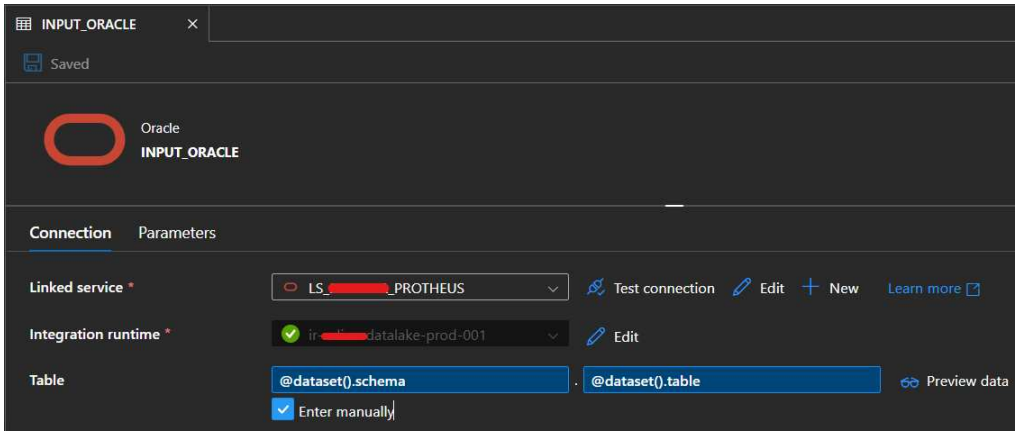
Além do Microsoft SharePoint e dos bancos de dados ORACLE como fontes de dados, também existem os dados vindos de *APIs* dos sistemas utilizados pela Empresa Cliente, essencialmente sistemas de gestão de projetos internos e de suporte ao usuário. Para a ingestão dos dados brutos dessas fontes, não foram adotados os mesmos métodos de criação dos pipelines como para as fontes SharePoint e Oracle. Consumir dados dessas *APIs* utilizando o Data Factory demandaria novos serviços Azure a serem configurados e aplicados à arquitetura, o que elevaria o custo de operação de toda a nova infraestrutura. A fim de manter os custos em um nível o mais baixo possível, toda a ingestão dos dados das *APIs* foi realizada utilizando os *notebooks* do Databricks, conforme o apresentado adiante neste documento. Utilizou-se de bibliotecas Python para acesso a requisições *HTTP/REST* e as chaves de acesso via Azure Key Vault para realizar a chamada nos *endpoints* das *APIs* e buscar os dados brutos destas. Dessa forma, manteve-se o custo dentro do esperado e conseguiu-se ingerir os dados das *APIs* de maneira satisfatória.

6.2.1.3 *Datasets* e Parâmetros

Duas configurações importantes a serem realizadas em *pipelines* de dados no Data Factory são os *Datasets* e os Parâmetros dos *pipelines*.

Os *Datasets* são configurações estabelecidas para indicar ao Data Factory a estrutura de tabelas que serão processadas nos *pipelines*. Indica-se qual é a origem (*input*) dos dados ingeridos no fluxo e qual o destino destes dados (*output*), configurando-se também tanto os *Linked Services* utilizados para conectar com a fonte quanto, para os casos de dados vindos de bancos de dados relacionais, o *Integration Runtime* utilizado para conectar ao servidor da Empresa Cliente. A Figura 29 apresenta um exemplo de uma das estruturas de *input* configuradas, onde observa-se o *Linked Service* e o *Integration Runtime* utilizado para conectar com o banco de dados Oracle.

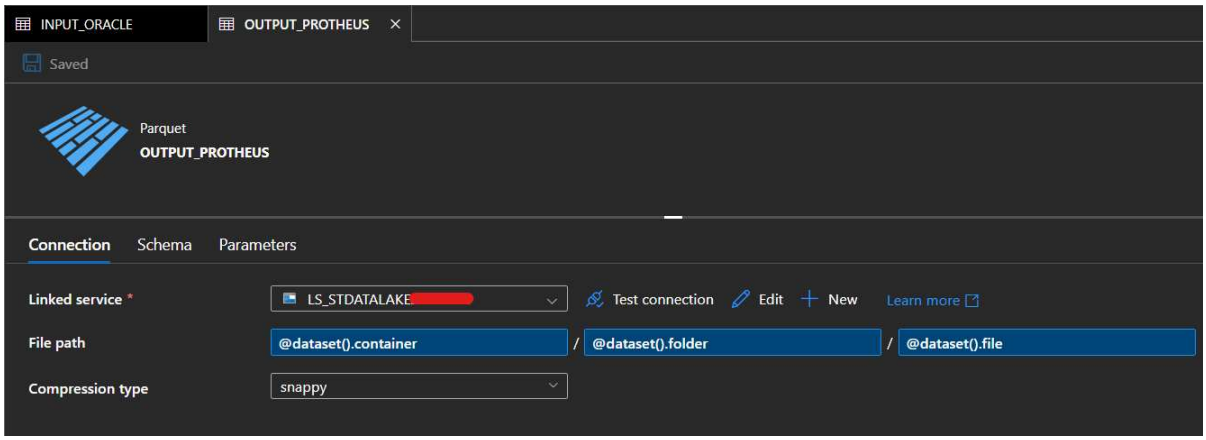
Figura 29 – Exemplo de *Input* dos *datasets* do Data Factory.



Fonte: Autor.

Já a Figura 30 apresenta um exemplo da estrutura de *output* configurada. Como todos os dados ingeridos pelos *pipelines* são salvos no serviço de armazenamento definido para o projeto Azure Data Lake Storage, todos os *outputs* dos *datasets* configurados são os mesmos, salvando os dados em formato *parquet* e utilizando o *Linked Service* para conexão com o Data Lake Storage.

Figura 30 – *Output* dos *datasets* do Data Factory.

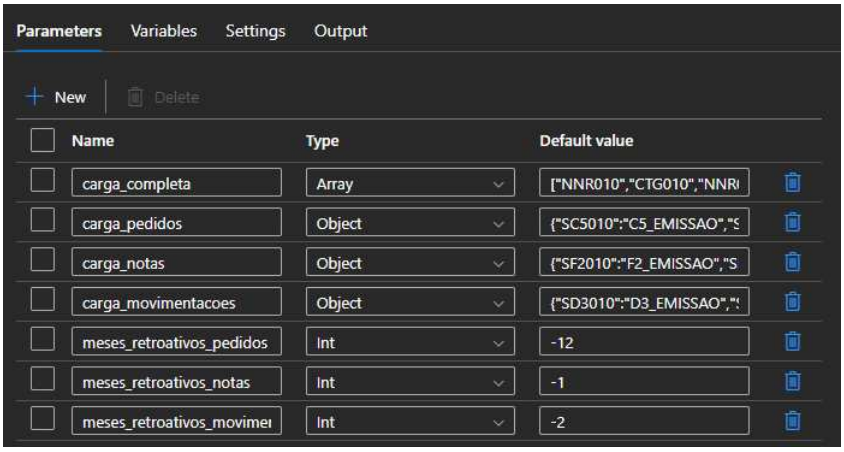


Fonte: Autor.

A outra configuração realizada nos *pipelines* para garantir a modularização são os Parâmetros de *pipeline*. Os parâmetros são valores salvos em variáveis, que são utilizados em diferentes operações dentro de um *pipeline* ou de múltiplos pipelines. A Figura 31 apresenta um exemplo de configuração de parâmetros do pipeline de ingestão dos dados dos bancos de dados Oracle. Na mesma, é possível observar que os parâmetros podem adotar diferentes formatos, como objetos, *arrays*, valores inteiros

e diferentes valores ou conjuntos de valores de acordo com a necessidade, podendo ser utilizados em diferentes etapas dos *pipelines* de acordo com a necessidade. No exemplo apresentado, e em grande parte do projeto, os parâmetros foram utilizados para definir as tabelas que são ingeridas dos bancos de dados relacionais, se são ingestões completas ou incrementais e, para as incrementais, qual o período retroativo utilizado para buscar os dados.

Figura 31 – Exemplo de parâmetros dos *pipelines*.



Fonte: Autor.

6.2.1.4 Gatilhos e Monitoramento

As últimas implementações necessárias para o funcionamento completo dos *pipelines* de dados do Data Factory são os Gatilhos, ou *triggers*. Os *triggers* permitem configurar e agendar períodos em diferentes formatos para que os fluxos de dados sejam ativados e executados de maneira completa. Essas estruturas são fundamentais para o projeto, pois permitem automatizar a operação de execução dos pipelines, configurando quando cada um deve rodar, podendo variar os períodos desde minuto a minuto até mensalmente, com alto grau de liberdade de configuração. Dessa forma, sempre que o período definido for alcançado, o *pipeline* é executado por conta própria conforme o agendamento e, de maneira automática, realiza todo o processo estabelecido.

Com as configurações anteriormente apresentadas, a estrutura de orquestração e ingestão dos dados brutos do projeto está completa, bastando replicar o que foi apresentado para cada um dos *pipelines* implementados, cada um com suas particularidades de tabelas, mas seguindo a mesma estrutura. Segue-se, então, para a implementação das etapas de processamento de dados via Azure Databricks, passando por cada uma das camadas *Bronze*, *Silver* e *Gold* e, por fim, na disponibilização das tabelas finais para consulta via Power BI.

Um recurso que não precisou ser implementado, mas que é muito útil no Data Factory é a seção de monitoramento. Nesta seção, pode-se acompanhar a execução dos *pipelines* (*Pipeline Runs*) em tempo real, com *logs* para acompanhar o andamento de cada etapa. O Monitor começa a ser atualizado sempre que um *pipeline* é executado, seja de maneira manual ou através dos *triggers*, informando ao usuário sobre o sucesso da execução. Também é possível configurar *e-mails* de alerta em caso de falha nas execuções dos *pipelines*, tornando o monitoramento dos mesmos bastante eficiente. A seguir, serão apresentados os detalhes de implementação da etapa de transformação dos dados.

6.2.2 Processamento dos dados na nova arquitetura

Como já mencionado, o Azure Databricks possui um papel fundamental no processamento dos dados, servindo como o coração de toda a nova arquitetura proposta. Além do processamento distribuído de dados que garante maior eficiência no tratamento de grandes volumes de dados, o mesmo possui funcionalidades de governança e gestão das tabelas, bem como funcionalidades de Inteligência Artificial que tornam a ferramenta uma excelente opção para o uso em muitas arquiteturas de dados.

A seguir, serão descritas as implementações realizadas no Databricks para realizar o tratamento dos dados da nova arquitetura da Empresa Cliente. Serão descritas as implementações para o processamento dos dados desde a camada bruta até a ponta final nos relatórios de Power BI, bem como os processos automatizados criados para executar a transformação em cada camada e os mecanismos de otimização das tabelas da arquitetura.

6.2.2.1 Camadas

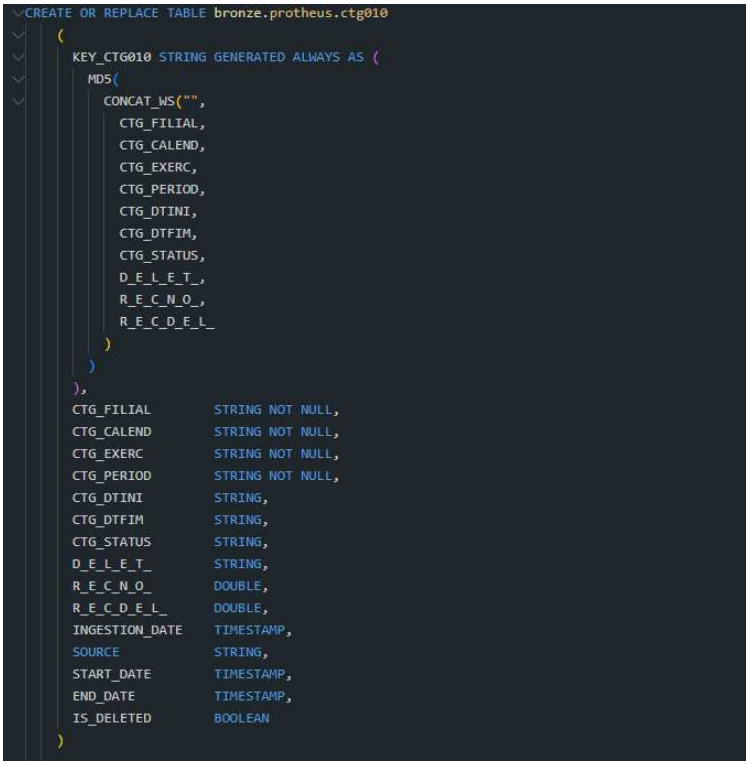
A metodologia para criação do *Data Lakehouse* da Empresa Cliente foi a Arquitetura Medalhão. Essa arquitetura prevê que os dados sejam distribuídos em três camadas, *Bronze*, *Silver* e *Gold*, cada uma com suas particularidades de processamento e finalidades de uso. No Azure Databricks, conforme a Figura 21, foram criados subdiretórios para cada uma das camadas para armazenar os *notebooks* de criação e ingestão das tabelas, respectivamente alocados em diretórios com nomes *CREATE_TABLE* e *INGESTION* dentro do diretório de cada camada. De maneira geral, a estrutura dos *notebooks* de criação e ingestão das tabelas é bastante semelhante entre as camadas. A título de apresentação, neste trabalho optou-se por focar em apresentar a estrutura base da camada *Bronze*, descrevendo as particularidades da camada *Silver* e *Gold* conforme forem pertinentes. As linguagens utilizadas para a implementação dos *notebooks* foram o Pyspark, da *API* de Python para uso do Spark e o Spark SQL, da *API* de SQL para uso do Spark.

Como mencionado, existem dois componentes básicos de processamento das tabelas, os *notebooks* de criação e os *notebooks* de ingestão das mesmas. Os *notebooks* são uma estrutura de escrita de código que permite a implementação em blocos e de forma modular, de maneira a criar e executar códigos de maneira sequencial.

***Notebooks* de criação das tabelas.**

A Figura 32 a primeira parte do *notebook* de criação das tabelas. Todas as tabelas criadas possuem o comando SQL *CREATE OR REPLACE TABLE*, seguido de uma estrutura de referência das tabelas do *Unity Catalog* do Databricks, com o padrão *camada.origem.nome_da_tabela*.

Figura 32 – Primeira parte do notebook de criação das tabelas.



Fonte: Autor.

Em seguida, são listadas todas as colunas da tabela a ser criada, seguidas do tipo específico de cada uma. A estrutura de *Delta Lake* utilizada na arquitetura pressupõe que as tabelas criadas possuem as propriedades ACID de segurança e confiabilidade dos dados. Além disso, outra propriedade implementada na arquitetura foi a de *Slowly Changing Dimensions (SCD)* tipo 2, ou Dimensões de Alteração Lenta tipo 2. O *SCD* é uma metodologia de registro que permite armazenar mudanças nos dados ao longo do tempo, registrando linhas novas conforme os dados são alterados e modificando colunas específicas que indicam essas mudanças. Dessa forma, é

possível rastrear as mudanças realizadas em uma tabela, mantendo um histórico, e permitindo a viagem no tempo com os dados da arquitetura, podendo rastrear um ponto no tempo em específico e acompanhar o comportamento dos dados em um dado período. Para que o *SCD* seja possível, é importante que sejam registradas na camada bronze colunas com os dados necessários para rastrear as mudanças na tabela. Uma delas é a coluna com nome *KEY_NOME_DA_TABELA*, que cria um valor *hash* aleatório único baseado nos valores das colunas da tabela para identificar cada linha como única. Também são definidas as chaves primárias da tabela, que representam um valor único para cada linha, podendo ser apenas uma coluna ou um conjunto delas (exemplo da Figura 32 nos campos marcados com *NOT NULL*). Além disso, são adicionados dados importantes como *INGESTION_DATE*, *START_DATE*, *END_DATE* e *IS_DELETED*, que indicam o ciclo de vida dos dados registrados nas colunas, permitindo registrar quando algum valor de uma coluna é alterado ou a linha é deletada na fonte, refletindo as mudanças no Data Lakehouse da Empresa Cliente e mantendo o histórico de alterações. Essas implementações tornam possível garantir as propriedades ACID em arquiteturas de dados baseadas em sistemas OLAP, utilizando a estrutura de *Delta Lake* para gerenciar os arquivos Parquet dentro do *Data Lakehouse* com os dados ingeridos e seu histórico de atualização. É importante salientar que o *SCD* é aplicado somente na camada *Bronze*, onde guarda-se o histórico dos dados para, caso necessário, sejam resgatados e repercutidos nas camadas *Silver* e *Gold*.

Para que o *SCD* seja possível, na criação da tabela são adicionadas propriedades que definem características de comportamento da tabela no *Data Lakehouse*. As propriedades configuradas podem ser observadas na Figura 33. O comando *TBLPROPERTIES* é chamado para definir as configurações de comportamento das tabelas Delta, como, por exemplo, a configuração *enableChanegDataFeed* que define que a tabela será alimentada com novos registros que alteram o estado de uma dada linha ao longo do tempo, *deletedFileRetationDuration* que define o período que um arquivo deletado é mantido no *Data Lakehouse* antes de ser excluído permanentemente e também o *targetFileSize* que define o tamanho máximo dos arquivos finais com os dados para manter a otimização dos mesmos.

Figura 33 – Segunda parte do notebook de criação das tabelas.

Fonte: Autor.

Ao longo das camadas, todas as tabelas criadas possuem as configurações apresentadas na Figura 33, tanto para *Bronze*, quanto para a *Silver* quanto para a *Gold*. Em relação à criação das colunas, na camada *Silver* e *Gold* não são criadas as colunas voltadas ao *SCD*, bastando na *Silver* herdar a chave *KEY_NOME_DA_TABELA* para controle de versão das linhas que serão alocadas nas tabelas finais desta camada e adicionando, tanto na *Silver* quanto na *Gold*, colunas de *Last_Update_Date*com um *timestamp* indicando quando os dados foram atualizados pela última vez. Com as tabelas criadas, parte-se para a implementação dos *notebooks* de ingestão dos dados.

***Notebooks* de ingestão das tabelas.**

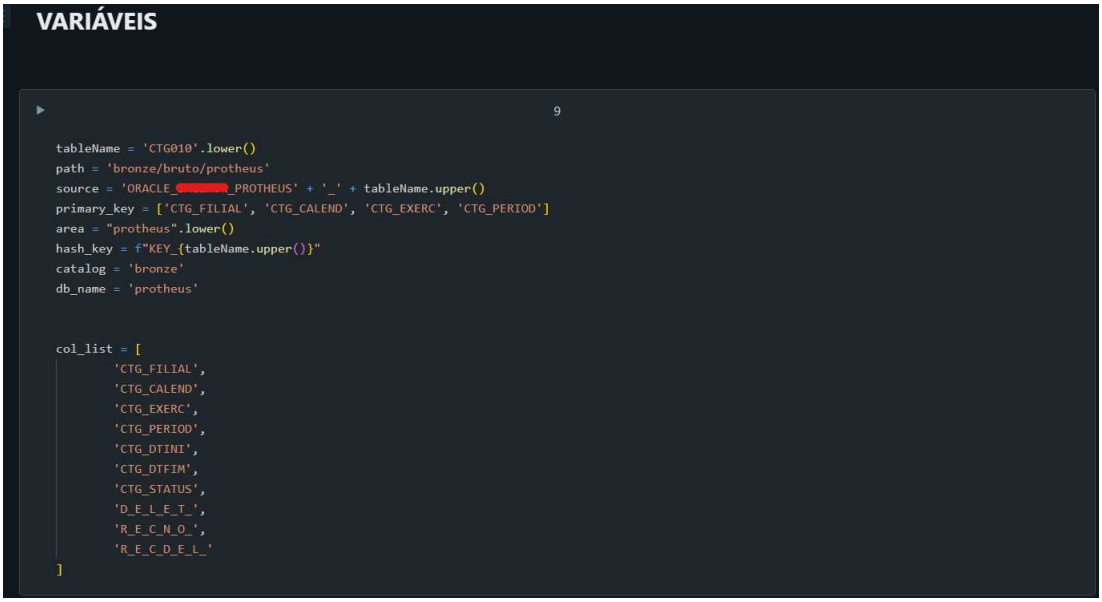
Os *notebooks* de ingestão, assim como os de criação, seguem uma estrutura semelhante entre as camadas, com alterações de acordo com a natureza de processamento em cada camada. De maneira geral, todos os *notebooks* de ingestão iniciam como na Figura 34, onde são configurados detalhes do Spark, importadas bibliotecas Python para utilizar o PySpark (biblioteca para acesso à *API* Spark via Python) e criadas funções que serão utilizadas ao longo do código para tratamentos ou execuções. No caso da camada *Bronze*, a função criada é utilizada para rodar o notebook de criação caso a tabela não exista no *Unity Catalog*. Já na camada *Silver*, que é a camada de tratamento e limpeza dos dados, são criadas funções de remoção de espaços em branco, remoção de duplicatas, transformação de colunas de datas, essenciais para a estruturação adequada das tabelas. Os *notebooks* da camada *Gold* seguem a mesma estrutura da camada *Bronze* em termos de bibliotecas e funções.

Figura 34 – Primeira parte do notebook de ingestão das tabelas.

Fonte: Autor.

Em seguida, são definidas variáveis que serão utilizadas tanto como parâmetros de nomes das tabelas, camadas e origens, quanto listas com as colunas da tabela para serem processadas dos arquivos brutos. Nas camadas *Silver* e *Gold*, somente a estrutura de parâmetros é executada.

Figura 35 – Segunda parte do notebook de ingestão das tabelas.



Fonte: Autor.

Assim que são definidos os parâmetros e colunas da tabela, segue-se para a leitura dos arquivos brutos ingeridos pelo Data Factory, conforme Figura 36. Verifica-se se a tabela já foi criada e, em caso negativo, roda-se o *notebook* de criação das mesmas. Após, o Spark é utilizado para ler os arquivos *parquet* das tabelas brutas e adicionadas as colunas com as informações do *SCD*. Por fim, uma *View* temporária em SQL, uma espécie de tabela não salva em memória, é criada para manipular os dados na fase final de processamento da camada *Bronze*. Este mesmo processo de verificação e leitura das tabelas é realizado nas camadas *Silver* e *Gold*, porém, não diretamente nos arquivos da camada bruta, mas utilizando a sintaxe SQL do Spark para ler os arquivos da camada anterior, respectivamente, *bronze.origem.nome_da_tabela* e *silver.origem.nome_da_tabela*. Por fim, a *view* temporária também é criada em cada tabela a fim de seguir para a etapa final de ingestão das mesmas.

Cabe comentar que nas transformações de limpeza de dados da camada *Silver*, somente recursos do Pyspark são utilizados. Já na camada *Gold*, todas as transformações são realizadas utilizando a API do Spark SQL, uma sintaxe de SQL para tratamento de dados via Spark. Essa opção foi escolhida pela facilidade de operações como união de tabelas, agregação de valores e sumarização de informações utilizando a sintaxe do SQL. Como essas operações são essencialmente realizadas na camada *Gold* para gerar as tabelas analíticas finais da arquitetura, é conveniente utilizar uma linguagem especificamente voltada para esse fim. Na camada *Gold*, todas as regras de criação das tabelas já utilizadas pela Empresa Cliente nos *dataflows* do Power BI foram replicadas para manter as tabelas fiéis à estrutura original.

Figura 36 – Terceira parte do notebook de ingestão das tabelas.

Fonte: Autor.

Na última etapa dos *notebooks* de ingestão, é realizado o processo de salvar o estado da tabela. Em todas as camadas, esse processo é realizado a partir de uma operação **MERGE INTO** do SQL sobre a *view* temporária criada anteriormente, conforme a Figura 37. Seleciona-se a tabela criada na camada *Bronze* como a tabela alvo, ou *TARGET* para receber as atualizações de estado dos dados. Em seguida, define-se a origem dos dados novos, ou *SOURCE*, baseada nos dados da *view* temporária criada e nos dados já presentes na tabela atualmente, realizando um *join* entre as tabelas com base na chave primária delas para validar os registros que já existem e devem ser mantidos, os que já existem e devem ser atualizados e os que não existem mais na fonte e devem ser excluídos.

Figura 37 – Quarta parte do notebook de ingestão das tabelas.

```
15

%sql

MERGE INTO

bronze.protheus.ctg010 AS CTG010_TARGET

USING (

    SELECT CONCAT_WS('', CTG_FILIAL, CTG_CALEND, CTG_EXERC, CTG_PERIOD) AS MERGEKEY,
           *
      FROM TEMP_UPDATES

    UNION ALL

    SELECT NULL AS MERGEKEY,
           UPD.*
    FROM TEMP_UPDATES UPD
    JOIN bronze.protheus.ctg010 OFI
      ON OFI.CTG_FILIAL  = UPD.CTG_FILIAL
     AND OFI.CTG_CALEND  = UPD.CTG_CALEND
     AND OFI.CTG_EXERC   = UPD.CTG_EXERC
     AND OFI.CTG_PERIOD  = UPD.CTG_PERIOD
     AND OFI.KEY_CTG010  != UPD.KEY_CTG010
   WHERE OFI.END_DATE    = '9999-12-31T00:00:00.000+0000'

) CTG010_SOURCE
```

Fonte: Autor.

Por fim, a comparação entre os registros que já estão na tabela e as novas atualizações ingeridas na camada bruta é realizada para definir o estado atual da tabela, conforme Figura 38. A comparação é realizada utilizando uma chave de junção, ou *MERGEKEY*, que é a concatenação da chave primária da tabela e também da coluna criada *KEY_NOME_DA_TABELA*. Quando existe uma correspondência (**WHEN MATCHED**) da chave na tabela original com as novas atualizações e a chave das colunas não corresponde, o valor de alguma das colunas daquela linha foi alterado; logo, aquele registro não é mais o atual de acordo com a origem dos dados. Logo, essa linha é marcada com uma data de fim *END_DATE* e uma nova linha é criada com os dados atualizados conforme a origem, assim como a adição dos novos registros que não existiam anteriormente na etapa de **WHEN NOT MATCHED**. Já a etapa final, **WHEN NOT MATCHED BY SOURCE**, processa os registros que não possuem correspondência na origem; logo, foram excluídos da tabela de origem ingerida, nesses casos sendo marcados como excluídos com a *flag IS_DELETED* e *END_DATE* do *Slowly Changing Dimension* tipo 2. Por fim, tem-se a tabela atualizada e com o histórico de alterações registrado na camada *Bronze*.

Cabe destacar que o processo de *join* com a tabela original e a verificação dos registros que já existem e foram alterados, já existem e devem ser mantidos, não existem e devem ser adicionados, não existem mais na origem e devem ser marcados como excluídos é executado exclusivamente na camada *Bronze* por conta do *SCD* tipo 2. Já para as camadas *Silver* e *Gold*, somente uma operação simples de **MERGE**

INTO é realizada para verificar os registros que devem ser atualizados, inseridos ou excluídos, pois não há registro histórico nessas camadas.

Figura 38 – Quinta parte do notebook de ingestão das tabelas.

```
) CTG010_SOURCE

  ON CONCAT_WS("", CTG010_TARGET.CTG_FILIAL, CTG010_TARGET.CTG_CALEND, CTG010_TARGET.CTG_EXERC, CTG010_TARGET.CTG_PERIOD) = CTG010_SOURCE.
  MERGEKEY

WHEN MATCHED
  AND CTG010_TARGET.END_DATE = '9999-12-31T00:00:00.000+0000'
  AND CTG010_SOURCE.KEY_CTG010 != CTG010_TARGET.KEY_CTG010
  AND CTG010_SOURCE.MERGEKEY IS NOT NULL

THEN
    UPDATE SET CTG010_TARGET.END_DATE = current_timestamp()

WHEN NOT MATCHED
THEN
    INSERT * EXCEPT(
        KEY_CTG010,
        MERGEKEY
    )

  WHEN NOT MATCHED BY SOURCE
    AND CTG010_TARGET.IS_DELETED = 'false'
  THEN
  UPDATE SET CTG010_TARGET.IS_DELETED = 'true',
  CTG010_TARGET.END_DATE = current_timestamp()
```

Fonte: Autor.

Importante salientar que o processo completo de ingestão das tabelas apresentado acima é o aplicado às tabelas ingeridas completas. Para o caso das tabelas ingeridas com atualização incremental, um processo ligeiramente diferente é aplicado. No caso das incrementais, as atualizações seguem o mesmo processo de **MERGE INTO** das tabelas completas, porém utilizando o recurso de *Auto Loader* do Databricks e Spark para processar os arquivos das tabelas incrementais em lote. O *Auto Loader* é utilizado para gerenciar automaticamente todos os arquivos com pequenas partes da tabela sendo ingeridos de maneira incremental e, em lotes, processá-los para que as atualizações sejam aplicadas à tabela final da camada *Bronze*. Para não tomar espaço no documento principal, o APÊNDICE A foi adicionado ao final do documento com o código *MERGE INTO* incremental para maior entendimento do processo.

Com as etapas anteriormente descritas de criação e ingestão, todas as tabelas da nova arquitetura podem ser processadas utilizando os recursos Spark do Azure Databricks. Como já mencionado, o grande ganho no uso do Databricks é a capacidade de processamento distribuído entre os diferentes nós em servidores diferentes com uma JVM alocada para cada carga de trabalho paralela. Em termos de grandes volumes de dados, essa provou-se uma das melhores práticas e uma das ferramentas mais aplicáveis do mercado para processamento de grandes volumes de dados, sendo este o caso da Empresa Cliente e de sua arquitetura.

6.2.2.2 *Workflows*

As últimas configurações realizadas em cada um dos *pipelines* de dados criados foram a criação dos *Workflows*. Os *workflows* são estruturas criadas no Databricks para executar *notebooks* de maneira sequencial e ordenada. Para isso, o Databricks possui uma estrutura de *jobs* e *tasks* onde é possível configurar as tabelas a serem executadas e a sequência de execução. A Figura 39 apresenta uma das estruturas de *workflow* implementadas. Nela, é possível notar os blocos correspondentes às *tasks* e o conjunto de tasks que representam um *job*.

Notam-se diferentes nomenclaturas para cada *task* configurada. As *tasks* que levam nomes de tabelas, como por exemplo **SC5010_BRONZE_INCREMENTAL**, referem-se ao processamento das tabelas incrementais, que por receberem o parâmetro de meses retroativos para atualizar os dados do Data Factory, devem ser executadas separadamente. Já as tasks com o nome **CAMADA_INGESTAO_PARALELA**, levam *noteboks* especiais, responsáveis por orquestrar e executar os *notebooks* de cada uma das tabelas processadas completas do *pipeline* (e do *workflow*) de maneira paralela para aproveitar melhor os recursos computacionais do Databricks.

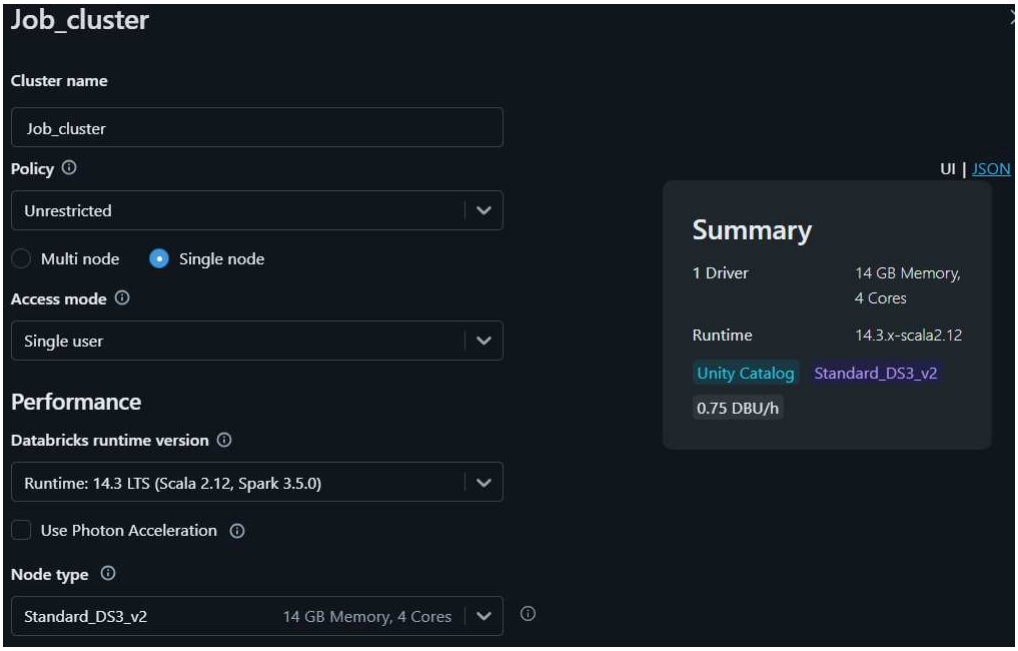
Figura 39 – Estrutura dos *workflows* do Databricks.

Fonte: Autor.

Os recursos computacionais, chamados de *clusters*, são também configurados nas *tasks* conforme a Figura 40. Os *clusters* são os elementos que tornam o Azure Databricks uma ferramenta altamente escalável. Nota-se que o cluster utilizado na Empresa Cliente possui uma baixa capacidade (14 GB de memória RAM e 4 núcleos de processamento), decisão tomada por questão de custos associados à nova arquitetura, que deve ficar em um patamar abaixo em um primeiro momento, conforme solicitação da Empresa Cliente. Porém, com o uso do Databricks, é possível que o *cluster* rapidamente seja reconfigurado para uma versão mais potente, com maior capacidade de

memória e processamento, múltiplos nós e versões mais recentes do Spark.

Figura 40 – Configuração dos *clusters* do Databricks.



Fonte: Autor.

Para executar os *workflows*, a *API* do Databricks é chamada via Azure Data Factory, conforme o bloco *run_databricks_job* da Figura 27. Via *API*, é possível parametrizar o *job* Databricks que se deseja executar, e acompanhar a execução via Azure Data Factory. Dessa forma, tem-se o controle tanto da ingestão dos dados quanto do processamento até a camada *Gold*, esta consumida pelo Power BI para alimentar os relatórios migrados para a nova arquitetura.

6.2.2.3 Otimizações

A fim de que o *Data Lakehouse* tenha a melhor performance possível, sem gargalos e arquivos processados desnecessariamente, *notebooks* de otimização foram criados no Databricks, com um pipeline específico no Data Factory. Duas funcionalidades de otimização foram utilizadas, o **VACUUM** e o **OPTIMIZE**. O **VACUUM** é uma operação comum aos bancos de dados, especialmente os PostgreSQL, onde busca-se uma série de funções para acelerar a performance dos bancos, como remoção de arquivos não utilizados, organização de indexadores e desfragmentação de tabelas. No Databricks, esta operação realiza a remoção dos arquivos e logs não mais utilizados pelas operações das tabelas delta, evitando que, ao processar, o mesmo utilize arquivos desnecessários que atrapalham a performance das operações. Já o **OPTMIZE** pode ser utilizado em diferentes operações para alterar a disposição de dados nos

arquivos *parquet* do *Delta Lake*. No caso da nova arquitetura da Empresa Cliente, a função **OPTMIZE ZORDER** foi utilizada, onde busca-se organizar os dados ordenados por uma coluna específica da tabela, determinada pelo usuário. Essa reordenação tem o objetivo de colocar mais acima nas tabelas registros mais acessados, com o objetivo de acelerar a busca nos sistemas OLAP colunares, caso do *Delta Lake*. As funções **VACUUM** e **OPTMIZE ZORDER** foram aplicadas em conjunto nas tabelas da camada *Gold*, enquanto, para a camada *Bronze* e *Silver*, somente o **VACUUM** foi aplicado. Com essas operações, é possível garantir a boa performance das operações de leitura e escrita das tabelas da nova arquitetura.

6.2.3 Fluxo de desenvolvimento e ferramentas auxiliares

Para o desenvolvimento dos *pipelines*, metodologias ágeis foram utilizadas, com ciclos de desenvolvimento (*sprints*) de uma a duas semanas que foram estabelecidos para cada um dos entregáveis. A cada semana, reuniões de acompanhamento (*daily*) foram realizadas com a Empresa Cliente, a fim de reportar a evolução dos relatórios, bem como alinhar a ordem de prioridade dos próximos relatórios a serem desenvolvidos conforme a demanda interna da empresa. Esses ciclos de desenvolvimento foram utilizados ao longo de todo o andamento do projeto.

Como já apresentado, ao todo 17 pipelines foram implementados no Data Factory e Databricks, cobrindo os dados dos 20 relatórios utilizados pela Empresa Cliente. As estruturas apresentadas anteriormente do Data Factory e Databricks serviram como base para a criação de todos os *pipelines* e foram aplicadas com pequenas alterações, conforme a estrutura das tabelas ingeridas e processadas. Assim que os primeiros *pipelines* foram implementados e validados, tanto pela BIX TECNOLOGIA quanto pela Empresa Cliente, obteve-se uma estrutura de base para a criação dos demais. Os detalhes de implementação das tabelas das fontes foram entendidos e estabelecidos, facilitando o desenvolvimento futuro dos relatórios a serem implementados, uma vez que as grandes dificuldades foram mapeadas logo nos primeiros. Assim, o desenvolvimento do projeto seguiu de maneira constante e sem grandes problemas, respeitando o cronograma estabelecido e os prazos para finalização do mesmo.

Ao longo do desenvolvimento, todas as ferramentas apresentadas na seção 6.1.3 foram utilizadas a fim de estruturar toda a nova arquitetura de dados da Empresa Cliente. Detalhes de uso do Azure Key Vault não foram apresentados por não serem pertinentes ao objetivo central da arquitetura, focada na implementação dos *pipelines* e no processamento dos dados via Databricks; porém, o Key Vault foi utilizado para guardar de forma segura todos os segredos, *tokens* e senhas utilizados no projeto. Outra ferramenta muito utilizada ao longo de todo o desenvolvimento foi o Azure DevOps, especialmente como ferramenta de versionamento dos códigos da arquitetura. Toda a estrutura visual de blocos do Data Factory utilizada para criar os *pipelines* é

convertida em estruturas de arquivos JSON que são interpretadas pelas máquinas da Azure para realizar as operações com os dados. Esses arquivos contêm a descrição de configuração de todos os blocos e como estes se comunicam entre si. Para controlar a versão oficial que será executada sempre que um *pipeline* for ativado, o Azure DevOps é utilizado, ficando salvo no repositório principal os arquivos de configuração de todos os *pipelines*, podendo ser alterados em um ramo de desenvolvimento separado sem alterar a versão utilizada em produção. O mesmo é realizado com os códigos Pyspark e Spark SQL utilizados na criação e processamento das tabelas da arquitetura, mantendo uma versão principal do código no repositório e editando ramos de desenvolvimento para implementar novas funcionalidades ou corrigir possíveis erros nos códigos de tratamento dos dados.

6.2.4 Disponibilização

Para realizar a disponibilização dos dados para alimentar os relatórios do Power BI, uma funcionalidade do Databricks foi utilizada, o *Delta Sharing*. O *Delta Sharing* consiste em um protocolo *REST* de código aberto criado pela Databricks e a Linux Foundation para acessar, de maneira segura e simples, dados estruturados no formato de *Delta Lake*. O *Delta Sharing* é configurado via Databricks, onde é criado um *share* e, nele, são selecionadas as tabelas do *Unity Catalog* que se deseja compartilhar via *Delta Sharing*. Ao final, uma *token* de *API* e um *link* de acesso aos dados são gerados. Nas ferramentas de visualização, é possível selecionar o *Delta Sharing* como fonte de dados, utilizando o *link* de acesso e o *token* da *API* para ter acesso a todos os dados compartilhados no *share* criado. A criação do *share* foi realizada pelo autor deste trabalho, enquanto a conexão com os dados no Power BI foi realizada pelo time de *Business Intelligence* da BIX TECNOLOGIA no projeto da Empresa Cliente, habilitando a disponibilização dos dados gerados na nova arquitetura.

6.3 ANÁLISE DOS RESULTADOS

Diferentes aspectos devem ser levados em consideração para avaliar os resultados obtidos com o projeto apresentado neste documento. De maneira geral, o projeto foi executado de maneira satisfatória e atendeu às expectativas da empresa cliente, especialmente em termos de aprimoramento de maturidade de arquitetura de dados e abertura de novas possibilidades de aplicação dos dados de sua operação, principalmente por agora possuir ferramentas capazes de habilitar o uso dos dados em aplicações não somente analíticas. Do ponto de vista dos objetivos gerais elencados na seção 1.2, o projeto cumpriu com as necessidades da Empresa Cliente. Foi possível migrar para a nova arquitetura todos os *pipelines* de dados da Empresa Cliente, ingerindo dados de diferentes origens e integrando-os em um *data lakehouse*

completo, replicando as regras de negócio já utilizadas e garantindo a conformidade com a estrutura anterior, tudo isto realizado em uma plataforma de nuvem altamente escalável. Em uma avaliação mais técnica, existem camadas a serem exploradas para analisar o impacto da nova arquitetura, como performance em termos de tempo de processamento, impacto na rotina de desenvolvimento e melhoria contínua da estrutura analítica e o nível de maturidade em dados da Empresa Cliente quando utilizadas as ferramentas desenvolvidas no presente projeto.

6.3.1 Performance da nova arquitetura

Em termos de tempo de processamento de ponta a ponta, desde a ingestão dos dados brutos até a entrega dos dados processados, a nova arquitetura apresentou um tempo semelhante ao já apresentado pela arquitetura anterior para alguns *pipelines* e, para outros, um tempo superior comparado à arquitetura baseada em sistema de *BI*. Apesar da nova arquitetura apresentar uma performance abaixo da arquitetura anterior, esse resultado não necessariamente deve ser interpretado como ruim. Uma das premissas apresentadas pela Empresa Cliente foi a de manter o custo financeiro da nova arquitetura no patamar mais baixo possível, considerando o uso mínimo dos recursos da Azure em um primeiro momento, até que o período de transição entre a arquitetura antiga e a nova seja completado. Por este fator, especialmente nos *clusters* do Databricks utilizados, a configuração de poder de processamento foi a mais baixa possível para respeitar o limite de gastos da arquitetura total. Em um momento futuro e com mais recursos financeiros disponíveis, é possível que os clusters da nova arquitetura sejam escalados para versões mais potentes, acelerando muito o processo de transformação dos dados e reduzindo consideravelmente o tempo de processamento dos dados e otimizando a performance.

6.3.2 Impacto da nova arquitetura na rotina da Empresa Cliente

Em termos de impacto na rotina da Empresa Cliente, a nova arquitetura implementada possui vantagens, especialmente relacionadas à qualidade e segurança de desenvolvimento e robustez nos processos analíticos da empresa. Na arquitetura anterior, as novas implementações eram realizadas utilizando arquivos locais *.pbix* do Power BI, muitas vezes necessitando de *backups* manuais para a alteração de uma parte do fluxo sem alterar o fluxo principal. Essa prática demanda tempo e grande esforço manual no desenvolvimento de novas funcionalidades ou manutenção no fluxo de dados. Já na nova arquitetura, todo o desenvolvimento de novas análises ou alterações no fluxo atual é realizado utilizando ferramentas de versionamento de código, boa prática adotada por todo o mercado para garantir a segurança de que o fluxo principal de dados não será alterado sem testes em um ambiente de desenvolvimento antes.

Dessa forma, garante-se a saúde de desenvolvimento, utilizando-se das melhores práticas em controle de versão e fluxo de implementações.

Para criar as regras de negócio da camada *Gold*, a nova arquitetura utiliza o Spark SQL, que se assemelha ao ORACLE SQL já utilizado pela Empresa Cliente, facilitando a transição para a nova arquitetura. Apesar de não haver familiaridade com o PySpark, treinamentos foram ministrados pelo autor deste documento para que os membros da Empresa Cliente entendam o fluxo e como utilizar o Python e PySpark na nova infraestrutura criada. Apesar da inércia inicial, a familiaridade com a nova arquitetura mostrou-se rápida, uma vez que novas implementações realizadas pelos membros da Empresa Cliente foram feitas antes do final do projeto, com depoimentos positivos quanto à robustez e qualidade dos novos processos.

Em termos financeiros, a nova arquitetura apresentou vantagens em relação à anterior. Para realizar as operações realizadas via Microsoft Azure com o Data Factory e Databricks no Power BI, como ingestão incremental de dados e maior poder de processamento, seriam necessárias licenças Premium do Power BI, o que dobraria os custos atuais da arquitetura. Com a implementação da nova arquitetura, o custo mensal ficou semelhante à compra de licenças premium do Power BI, porém entregando uma plataforma totalmente nova e com funcionalidades que não seriam possíveis somente utilizando o Power BI. Ingestão de dados incremental, arquitetura medalhão com diferentes tratamentos nos dados como o histórico de dados na camada *Bronze*, escalabilidade do poder de processamento do Databricks, habilitação para uso de modelos de Inteligência Artificial com o Databricks, versionamento de códigos dos *pipelines* e de transformação dos dados, segurança de senhas e segredos com Key Vault e rápida disponibilidade utilizando o *Delta Sharing*.

6.3.3 Avaliação do nível de maturidade da nova arquitetura frente à anterior

Diante dos pontos expostos anteriormente, é possível notar a evolução da nova arquitetura de dados da Empresa Cliente em relação à anterior. A troca de filosofia de arquitetura de um sistema puramente baseado em Sistemas de **Business Intelligence** e relatórios visuais para uma arquitetura de dados baseada em nuvem coloca a empresa em um novo patamar de maturidade de dados. Anteriormente, a estrutura de dados da empresa tratava-se de uma arquitetura puramente analítica, para análises descritivas utilizando os dados gerados pelos sistemas e subsistemas adotados em suas operações. Contudo, agora a Empresa Cliente está habilitada a utilizar sua infraestrutura de informação para diferentes fins e aplicações, desde as já utilizadas análises descritivas com relatórios visuais de *BI* até a análise prescritiva com modelos de Inteligência Artificial capazes de prever demandas, faturamentos, ou qualquer indicador que se deseje avaliar. Apesar da nova arquitetura, em um primeiro momento, apresentar uma performance em termos de tempo inferior à anterior por conta dos

clusters configurados no Databricks, entende-se que o potencial desbloqueado com a mesma possui grande valor para a Empresa Cliente. Além de fornecer o que há de mais moderno e eficiente nas infraestruturas de dados, a nova arquitetura possui elementos e boas práticas que elevam o nível de maturidade de dados da empresa, este sendo o objetivo principal do projeto.

6.4 AVALIAÇÃO DAS ENTREGAS FRENTE AOS REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Inicialmente, estipularam-se requisitos de funcionamento da arquitetura que atendessem às necessidades da Empresa Cliente. Requisitos funcionais e não funcionais foram estabelecidos, a fim de nortear o desenvolvimento do projeto e avaliar as medidas de sucesso da implementação realizada. Com base na avaliação dos requisitos, pode-se considerar que o resultado final foi bastante satisfatório, tanto para os requisitos funcionais quanto para os não funcionais.

Do ponto de vista dos requisitos funcionais, a maioria dos requisitos foi cumprida; a nova arquitetura é capaz de ingerir os dados de todos os sistemas utilizados na operação da Empresa Cliente, utilizando a arquitetura medalhão para obter diferentes níveis de tratamento nos dados e contando com histórico de alteração dos mesmos. Tabelas incrementais foram implementadas para realizar a ingestão de maneira mais eficiente. As tabelas refletem exatamente o que está definido na arquitetura anterior e melhorias e manutenções podem ser realizadas de maneira a não interferir no fluxo principal dos *pipelines*. Em relação ao requisito de eficiência no processamento do volume de dados, a nova arquitetura possui capacidade de processamento para tratar os dados de maneira extremamente eficiente, bastando escalar os *clusters* do Databricks para uma versão com capacidade ligeiramente maior para que o tempo de processamento melhore consideravelmente.

Do ponto de vista dos requisitos não funcionais, considera-se que todos foram alcançados. Na nova arquitetura, os *pipelines* de dados são atualizados de acordo com a atualização da arquitetura anterior, o monitoramento dos *pipelines* é realizado de maneira eficiente com logs de execução, a escalabilidade da arquitetura é garantida com apenas pequenos ajustes para evoluir o poder de processamento dos dados, a conexão com o Power BI foi realizada a fim de utilizar os dados da nova arquitetura nos relatórios visuais e manteve-se o custo financeiro de operação da arquitetura no patamar mais baixo possível dado o volume atual de dados da Empresa Cliente.

Dessa forma, entende-se que as entregas realizadas no projeto cumpriram com os objetivos e atenderam às expectativas. *Feedbacks* foram coletados da Empresa Cliente em uma reunião de finalização do projeto, os quais foram bastante satisfatórios e reforçaram a parceria que a mesma possui com a BIX TECNOLOGIA. Assim, concluiu-se que o projeto foi implementado com sucesso.

7 CONCLUSÃO

O projeto desenvolvido neste trabalho de fim de curso foi o de desenvolvimento de uma arquitetura completa de dados, realizando uma migração de uma arquitetura de dados preexistente baseada em sistemas de *Business Intelligence* para uma nova arquitetura de dados baseada em serviços de nuvem. A Empresa Cliente atendida pela BIX TECNOLOGIA possuía toda a sua infraestrutura analítica baseada na arquitetura anterior, com limitações de processamento, falta de escalabilidade e de recursos modernos de arquiteturas de dados, estando aquém do que é estabelecido no mercado como uma arquitetura robusta e segura. Além disso, o crescente volume de dados analisados pela empresa levou a arquitetura atual ao limite, necessitando de aprimoramentos ou novas ferramentas capazes de suportar a nova demanda. Por esse motivo, o projeto desenvolvido neste trabalho de fim de curso foi proposto, com o objetivo de oferecer uma solução segura, escalável e com as melhores tecnologias de mercado para o tratamento de grandes volumes de dados com segurança e eficiência.

A solução proposta foi o uso dos serviços de nuvem da Microsoft Azure para implementar uma arquitetura de dados completa, desde a ingestão dos dados brutos das origens até a entrega dos dados finais em diferentes níveis de tratamento e alimentando os já existentes relatórios visuais de *BI* utilizados pela Empresa Cliente. Diferentes serviços foram configurados e conectados para desenvolver toda a nova arquitetura, desde serviços de orquestração dos *pipelines* e processamento dos dados em camadas, até ferramentas de segurança e versionamento de código. As melhores práticas de mercado foram adotadas no projeto, como a criação de um *Data Lakehouse* com dados tratados em diferentes níveis para aumentar o leque de aplicação destes pela Empresa Cliente. Ferramentas robustas de processamento distribuído de dados foram adotadas para garantir performance e escalabilidade da solução e boas práticas de desenvolvimento de software foram adotadas para oferecer uma solução sólida e completa em todos os aspectos.

7.1 RESULTADOS GERAIS OBTIDOS

Com o projeto desenvolvido, foi possível fornecer à Empresa Cliente uma nova infraestrutura de dados completa baseada no serviço de nuvem Microsoft Azure. A infraestrutura conta com ferramentas modernas de análise de dados baseadas em um *Data Lakehouse*, com ferramentas de armazenamento de dados otimizadas para leitura de dados analíticos (Azure Data Lake Storage), uma ferramenta altamente escalável e robusta de processamento capaz de distribuir o processamento dos dados de maneira eficiente (Azure Databricks), além de ferramentas de orquestração e ingestão dos dados capazes de gerenciar os fluxos de dados e garantir o funcionamento adequado dos mesmos (Azure Data Factory). Soma-se às tecnologias anteriores também

uma ferramenta de versionamento capaz de adicionar uma camada de boas práticas de desenvolvimento, realizando o versionamento dos códigos utilizados de maneira automatizada e direta, sem interferir no código principal do projeto (Azure DevOps). Dessa forma, foi possível entregar ao cliente uma infraestrutura completa e moderna de dados, capaz de realizar o processamento dos dados desde as diferentes origens até a ponta final, seja ela em relatórios de análises descritivas quanto no uso em modelos de Inteligência Artificial e ferramentas mais modernas de análise.

7.2 TRABALHOS FUTUROS

Mesmo que o projeto implementado neste trabalho de fim de curso tenha cumprido com os objetivos iniciais estabelecidos, existem melhorias que podem ser feitas para que a nova arquitetura tenha um desempenho ainda melhor. Como sugestão para trabalhos futuros, sugere-se:

- **Aumentar a capacidade de processamento**: Com o processamento realizado via Databricks, quanto mais potente o *cluster* utilizado, maiores os custos por hora de operação do mesmo. Existe um equilíbrio que se pode buscar entre processamento x custo, que pode ser explorado aumentando-se o poder computacional dos *clusters* utilizados atualmente. Apesar de aumentar o custo por hora, com um *cluster* de maior capacidade os *pipelines* rodariam mais rapidamente, utilizando por menos tempo o recurso e reduzindo o valor/hora do mesmo. Recomenda-se que sejam exploradas as possibilidades de aumento de capacidade dos *clusters* a fim de encontrar uma melhor relação custo x tempo.
- **Otimizar os *pipelines***: Inicialmente no projeto, foi decidido por realizar a implementação da nova arquitetura baseada nos relatórios já utilizados pela Empresa Cliente e nos fluxos de dados destes. Essa decisão provou-se eficiente na validação dos relatórios conforme a entrega dos mesmos. Porém, para diversos *pipelines* implementados, as tabelas ingeridas são compartilhadas e existe um nível de sobreposição das tabelas atualizadas em cada *pipeline* executado. Sugere-se que seja realizada uma rodada de otimização dos *pipelines*, reduzindo as tabelas compartilhadas pelos *pipeline* somente para o executado com maior frequência. Dessa forma, reduz-se a carga de trabalho repetido nos *pipelines* de ingestão e processamento e economiza-se tempo e custo com os dados reprocessados recorrentemente.

Com estas melhorias, a nova arquitetura pode apresentar ganhos ainda maiores de eficiência de processamento e eficiência financeira.

REFERÊNCIAS

ALI, Ahmed Hussein; ABDULLAH, Mahmood Zaki. A Survey on Vertical and Horizontal Scaling Platforms for Big Data Analytics. **International Journal of Integrated Engineering**, 2019. DOI: https://doi.org/10.30880/ijie.2019.11.06.015. Disponível em: https://publisher.uthm.edu.my/ojs/index.php/ijie/article/view/2892.

ASHTARI, Hossein. What Is ETL (Extract, Transform, Load)? Meaning, Process, and Tools. **Spiceworks**, 2022. Disponível em: https://www.spiceworks.com/tech/devops/articles/extract-transform-load-etl/.

BHAGAT, Vandana; GOPAL, Arpita. Comparative Study of Row and Column Oriented Database. **IEEE**, 2012. 2012 Fifth International Conference on Emerging Trends in Engineering and Technology. DOI: https://doi.org/10.1109/ICETET.2012.56. Disponível em: https://ieeexplore.ieee.org/abstract/document/6495251.

CAMBRIGE. **data**: Meaning of data in English. [*S.l.*], 2024. Disponível em: https://dictionary.cambridge.org/dictionary/english/data.

CODD, Edgar Frank. A Relational Model of Data for Large Shared Data Banks. **Communications of the ACM**, v. 13, 1970. Disponível em: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf.

CONN, Samuel S. OLTP and OLAP data integration: a review of feasible implementation methods and architectures for real time data analysis. **Proceedings. IEEE SoutheastCon, 2005.**, p. 515–520, 2005. Disponível em: https://ieeexplore.ieee.org/abstract/document/1423297.

CRUTH, Mark. **Descubra o modelo do Spotify**: O que a empresa de tecnologia musical mais popular tem para ensinar sobre a escalabilidade da agilidade. [*S.l.*], 2022. Disponível em: https://www.atlassian.com/br/agile/agile-at-scale/spotify.

DATABRICKS. **What is a medallion architecture?** [*S.l.*], 2024. Disponível em: www.databricks.com/glossary/medallion-architecture.

DEAN, Jeffrey; GHEMAWAT, Sanjay. MapReduce: Simplified Data Processing on Large Clusters. **Google Research**, 2004. Disponível em:

https://static.googleusercontent.com/media/research.google.com/pt-BR//archive/mapreduce-osdi04.pdf.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Fundamentals of Database Systems**. 7. ed. [*S.l.*]: Pearson, 2015.

GHEMAWAT, Sanjay; GOBIOFF, Howard; LEUNG, Shun-Tak. The Google File System. **Google Research**, 2003. Disponível em: https://static.googleusercontent.com/media/research.google.com/pt-BR//archive/gfs-sosp2003.pdf.

GOODWIN, Michael. **What is an API?** [*S.l.*], 2024. Disponível em: https://www.ibm.com/topics/api.

HARBY, Ahmed A.; ZULKERNINE, Farhana. From Data Warehouse to Lakehouse: A Comparative Review. **2022 IEEE International Conference on Big Data (Big Data)**, p. 389–395, 2022. DOI: 10.1109/BigData55660.2022.10020719. Disponível em: https://ieeexplore.ieee.org/document/10020719.

ISO. **ISO/IEC 9075-1:2023**: Information technology — Database languages SQL. [*S.l.*], 2023. Disponível em: https://www.iso.org/standard/76583.html.

KHINE, Pwint; WANG, Zhao. Data lake: a new ideology in big data era. **ITM Web of Conferences**, v. 17, p. 03025, jan. 2018. DOI: 10.1051/itmconf/20181703025.

KSHEMKALYANI, Ajay D.; SINGHAL, Mukesh. **Distributed Computing: Principles, Algorithms, and Systems**. 1. ed. [*S.l.*]: Cambridge University Press, 2011.

L'ESTEVE, Ron C. **The Definitive Guide to Azure Data Engineering**. 1. ed. [*S.l.*]: Apress, Berkeley, CA, 2021. DOI: https://doi.org/10.1007/978-1-4842-7182-7_15. Disponível em: https://link.springer.com/chapter/10.1007/978-1-4842-7182-7_15#citeas.

NAEEM, Tehreem. Data Warehouse Concepts: Kimball vs. Inmon Approach. **Astera**, 2020. Disponível em: https://www.astera.com/type/blog/data-warehouse-concepts/.

SALLOUM, Salman; DAUTOV, Ruslan; CHEN, Xiaojun; PENG, Patrick Xiaogang; HUANG, Joshua Zhexue. Big data analytics on Apache Spark. **International Journal**

of Data Science and Analytics, 2016. Disponível em: https://link.springer.com/article/10.1007/s41060-016-0027-9.

SAMSON, Grace L.; LU, Joan; XU, Qiang. Large spatial datasets: Present Challenges, future opportunities. **International Conference on Change, Innovation, Informatics and Disruptive Technology**, 2016. Disponível em: https://www.researchgate.net/publication/316216480_Large_spatial_datasets_Present_Challenges_future_opportunities.

SUSNJARA, Stephanie; SMALLEY, Ian. **What is cloud computing?** [*S.l.*], 2024. Disponível em: https://www.ibm.com/topics/cloud-computing.

ZAHARIA, Matei; CHOWDHURY, Mosharaf; FRANKLIN, Michael J.; SHENKER, Scott; STOICA, Ion. Spark: Cluster Computing with Working Sets. **University of California, Berkeley**, 2010. Disponível em: https://amplab.cs.berkeley.edu/wp-content/uploads/2011/06/Spark-Cluster-Computing-with-Working-Sets.pdf.

APÊNDICE A – CÓDIGO MERGE INTO DAS TABELAS INCREMENTAIS

Figura 41 – Parte 1 MERGE INTO Incremental.

```
17

# upsertToDelta para ingestão incremental com o Auto-loader

def upsertToDelta(catalog, db_name, table_name, table_id, date_mod):
    def _do_work(microbatchOutputDF, batchId):

        #add ingestion date e outros metadados para cada microbatch
        microbatchOutputDF = add_metadados(microbatchOutputDF, 'KEY_SC5010')

        #remove duplicates em caso de acumulos de batchs
        microbatchOutputDF = microbatchOutputDF.withColumn("rn", row_number().over(Window.partitionBy([col(x) for x in table_id])\
                                                                                   .orderBy(microbatchOutputDF[date_mod].desc())))
        microbatchOutputDF = microbatchOutputDF.filter('rn = 1')
        microbatchOutputDF = microbatchOutputDF.drop(col('rn'))

        #createTempView para os novos registros
        microbatchOutputDF.createOrReplaceTempView('UPDATES')

        #upsert merge
        microbatchOutputDF.sparkSession.sql(f"""
                    MERGE INTO
                    {catalog}.{db_name}.{table_name} AS SC5010_TARGET -- Alvo
                    USING (
                        SELECT CONCAT_WS('', C5_FILIAL, C5_NUM) AS MERGEKEY,
                               *
                          FROM UPDATES

                         UNION ALL

                        SELECT NULL AS MERGEKEY,
```

Fonte: Autor.

Figura 42 – Parte 2 MERGE INTO Incremental.

```
        #upsert merge
        microbatchOutputDF.sparkSession.sql(f"""
                    MERGE INTO
                    {catalog}.{db_name}.{table_name} AS SC5010_TARGET -- Alvo
                    USING (
                        SELECT CONCAT_WS('', C5_FILIAL, C5_NUM) AS MERGEKEY,
                               *
                          FROM UPDATES

                         UNION ALL

                        SELECT NULL AS MERGEKEY,
                               UPD.*
                          FROM UPDATES UPD
                          JOIN {catalog}.{db_name}.{table_name} OFI
                            ON OFI.C5_FILIAL    = UPD.C5_FILIAL
                            AND OFI.C5_NUM      = UPD.C5_NUM
                            AND OFI.KEY_SC5010  != UPD.KEY_SC5010
                          WHERE OFI.END_DATE     = '9999-12-31T00:00:00.000+0000'

                    ) SC5010_SOURCE

                      ON CONCAT_WS("", SC5010_TARGET.C5_FILIAL, SC5010_TARGET.C5_NUM) = SC5010_SOURCE.MERGEKEY

                    WHEN MATCHED
                     AND SC5010_TARGET.END_DATE = '9999-12-31T00:00:00.000+0000'
                     AND SC5010_SOURCE.KEY_SC5010 != SC5010_TARGET.KEY_SC5010
                     AND SC5010_SOURCE.MERGEKEY IS NOT NULL
                    THEN
```

Fonte: Autor.

Figura 43 – Parte 3 MERGE INTO Incremental.

```
                WHEN MATCHED
                 AND SC5010_TARGET.END_DATE = '9999-12-31T00:00:00.000+0000'
                 AND SC5010_SOURCE.KEY_SC5010 != SC5010_TARGET.KEY_SC5010
                 AND SC5010_SOURCE.MERGEKEY IS NOT NULL
                THEN
                    UPDATE SET SC5010_TARGET.END_DATE = CURRENT_TIMESTAMP()

                WHEN MATCHED
                  AND SC5010_SOURCE.KEY_SC5010 = SC5010_TARGET.KEY_SC5010
                THEN UPDATE SET *

                WHEN NOT MATCHED
                THEN
                    INSERT * EXCEPT(
                        KEY_SC5010,
                        MERGEKEY
                    )

                WHEN NOT MATCHED BY SOURCE
                 AND SC5010_TARGET.IS_DELETED = 'false'
                 AND TO_DATE(SC5010_TARGET.C5_EMISSAO, 'yyyyMMdd') >= DATE_TRUNC('day', DATEADD(MONTH, getArgument("meses_retroativos_pedidos"),
                 CURRENT_TIMESTAMP()))
                THEN
              UPDATE SET SC5010_TARGET.IS_DELETED = True,
              SC5010_TARGET.END_DATE = current_timestamp()
            """)

    return _do_work
```

Fonte: Autor.