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

Fellipe Domingues Fernandes

Implementação de uma arquitetura de processamento distribuído de grandes volumes de dados em uma empresa do varejo

Florianópolis
2023

Fellipe Domingues Fernandes

Implementação de uma arquitetura de processamento distribuído de grandes volumes de dados em uma empresa do varejo

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. Ricardo José Rabelo, Dr.
Supervisor: Felipe Ferraz Leonardo, Eng.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Fernandes, Fellipe Domingues
Implementação de uma arquitetura de processamento distribuído de grandes volumes de dados em uma empresa do varejo / Fellipe Domingues Fernandes ; orientador, Ricardo José Rabelo, 2023.
111 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Big Data. 3. Data Warehouse. 4. Computação Distribuída. 5. Análise de Dados. I. Rabelo, Ricardo José. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Fellipe Domingues Fernandes

Implementação de uma arquitetura de processamento distribuído de grandes volumes de dados em uma empresa do varejo

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, 15 de Dezembro de 2023.

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

Banca Examinadora:

Prof. Ricardo José Rabelo, Dr.
Orientador
UFSC/CTC/DAS

Documento assinado digitalmente
Ricardo Jose Rabelo
Data: 15/12/2023 16:34:32-0300
CPF: ***.802.119-**
Verifique as assinaturas em https://v.ufsc.br

Felipe Ferraz Leonardo, Eng.
Supervisor
BIX Tech

Documento assinado digitalmente
FELIPE FERRAZ LEONARDO
Data: 15/12/2023 17:33:45-0300
Verifique em https://validar.iti.gov.br

Prof. Rodrigo Lange, Dr.
Avaliador
IFRS/Campus Ibirubá

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

Dedico este trabalho aos meus amados pais, cujo apoio
e encorajamento foram cruciais para sua conclusão.

AGRADECIMENTOS

Agradeço profundamente aos membros da minha família, que nunca deixaram de estar ao meu lado, que me levantavam enquanto eu caía, e que me motivavam nos muitos momentos em que pensei em desistir. Aos meus pais, Patricia Domingues Fernandes e Reinaldo Arcendino Fernandes, sou grato por proporcionarem um lar acolhedor, por jamais deixarem faltar o que comer, por priorizarem a educação dos seus filhos, pela paciência que tiveram com meus momentos de estresse e sobretudo, por demonstrarem a importância do respeito, da integridade e da moralidade.

Ao meu irmão, Reinaldo Domingues Fernandes e sua esposa Juliana Bezerra por estarem presentes e sempre dispostos à ajudar ao longo desse trajeto, pelos bons jantares compartilhados e as boas conversas.

À minha namorada, Isabela Morgerot Ceni, por me motivar a ir além, me desafiar, sair da zona de conforto. Também pela paciência e compreensão nos momentos em que ficava mais distante e pelas memórias que construímos juntos ao longo dos anos.

Ao meu avô Valberto Antônio Domingues, pelo encorajamento, pelos sorvetes e ótima companhia, à minha madrinha Jussara Domingues por sempre acreditar em mim, pelas risadas e brincadeiras compartilhadas, e à todos os demais membros da minha família que contribuíram de uma forma ou de outra com essa jornada.

Também gostaria de deixar meus agradecimentos aos professores da ECA/UFSC, em especial ao meu orientador Ricardo José Rabelo, que foi uma figura chave no meu desenvolvimento pessoal, acadêmico e profissional enquanto graduando do curso de Engenharia de Controle e Automação. Sua orientação, lições e ensinamentos formaram grande parte do que sou hoje e abriram muitas portas na minha vida. Serei eternamente grato por tê-lo tido como professor, mentor, orientador e amigo.

Estendo meus agradecimentos à BIX Tech e seu corpo de consultores e executivos, em especial ao Felipe Santos Eberhardt, egresso da ECA/UFSC por supervisionar uma empresa tão acolhedora, humana e justa. Ao Matheus Kjellin, também egresso da ECA/UFSC por ser um exímio líder técnico, por me receber tão calorosamente na BIX Tech e por tudo que me ensinou. Ao meu supervisor Felipe Ferraz Leonardo por ser essa pessoa carismática, aberta, empática e extremamente competente, pelas conversas técnicas, pelos momentos de descontração e também os de trabalho. Sou grato à todos os outros colaboradores que ajudaram no meu crescimento profissional e que contribuíram direta ou indiretamente com a realização deste trabalho.

À Universidade Federal de Santa Catarina e seus servidores, por proporcionarem um corpo docente capacitado, um ambiente inclusivo, por estimularem a racionalidade e proporem o questionamento como arma contra a desinformação.

Por fim, aos amigos e colegas de curso que, unidos, tornam os momentos durante a graduação tão memoráveis.

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 15 de Dezembro de 2023.

Na condição de representante da BIX Tech 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
FELIPE FERRAZ LEONARDO
Data: 15/12/2023 17:29:21-0300
Verifique em https://validar.iti.gov.br

Felipe Ferraz Leonardo
BIX Tech

RESUMO

Dada a alta competitividade do setor de varejo com o advento dos mercados eletrônicos (*e-commerce*), as empresas buscam otimizar seus processos de venda, atração e retenção de clientes com o uso da análise de grandes volumes de dados históricos almejando se destacar frente aos seus concorrentes. Entender o comportamento do consumidor e agilizar a tomada de decisões são elementos chave para o sucesso de um negócio. Esse projeto aborda os desafios de uma grande empresa do varejo na implementação de uma arquitetura de engenharia de dados que se utiliza de tecnologias modernas como a replicação de bases de dados em tempo real, processamento distribuído de altos volumes de dados e disponibilização democrática de informação para os *stakeholders*, analistas de inteligência de negócio e para modelos de Inteligência Artificial e Aprendizado de Máquina na forma de tabelas agregadas com dados de vendas, fretes, parceiros, notas fiscais, remessas, entre outras a partir de bases históricas existentes, representando um complexo problema de computação dado o volume de dados. A plataforma construída será utilizada pela empresa cliente do projeto de forma a concedê-la ganhos significativos em performance de consultas, rapidez na tomada de decisões de negócios, eficiência operacional, cálculo de indicadores chave como a taxa de *churn* de clientes, valor vitalício por cliente, comissão de parceiros, tendências de venda e muitos outros. Esse estudo se propõe como um guia para empresas que buscam aprimorar suas estratégias e cultura de dados não só no contexto do varejo, mas em qualquer área de negócio que possa se beneficiar da tomada de decisão baseada em dados, abrangendo conceitos, dificuldades, limitações e custos envolvidos na implementação de uma arquitetura moderna de dados.

Palavras-chave: Big Data. Computação distribuída. Computação em nuvem. Data Lakehouses. Orquestração. Replicação de bancos de dados.

ABSTRACT

Given the high competitiveness of the retail sector with the advent of electronic markets (e-commerce), companies are seeking to optimize their sales processes, attraction and retention of customers through the analysis of large volumes of historical data, aiming to stand out from their competitors. Understanding consumer behavior and speeding up decision-making are key elements for the success of a business. This project addresses the challenges of a large retail company in implementing a data engineering architecture that makes use of modern technologies such as real-time database replication, distributed processing of high volumes of data, and democratic availability of information for stakeholders, business intelligence analysts, and Artificial Intelligence and Machine Learning models in the form of aggregated tables with data on sales, freight, partners, invoices, shipments, among others from existing historical bases, representing a complex computing problem given the volume of data. The proposed platform will be used by the client company to grant it significant gains in query performance, speed in business decision-making, operational efficiency, calculation of key indicators such as customer churn rate, lifetime value per customer, partner commission, sales trends, and many others. This study serves as a guide for companies looking to improve their data strategies and data culture not only in the retail context but in any business area that can benefit from data-driven decision-making, covering concepts, difficulties, limitations and costs involved in implementing a modern data architecture.

Keywords: Big Data. Distributed Computing. Cloud Computing. Data Lakehouses. Orchestration. Database replication.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
API	Application Programming Interface
B2B	Business-to-Business
B2C	Business-to-Consumer
BI	*Business Intelligence* - inteligência de negócios
BQ	BigQuery
CDC	*Change Data Capture* - captura de mudança de dados
DAG	Directed Acyclic Graph
DE	*Data Engineering* - engenharia de dados
DS	*Data Science* - ciência de dados
DW	*Data Warehouse* - armazém de dados
ERD	Entity-Relationship Diagram
ERP	Enterprise Resource Planning
ETL	Extract, Transform and Load
FSM	Finite State Machine
GCP	Google Cloud Platform
GCS	Google Cloud Storage
GKE	Google Kubernetes Engine
HDD	Hard Disk Drive - disco rígido
IAM	Identity and Access Management
KPIs	Key Performance Indicators
ML	Machine Learning
MQTT	MQ Telemetry Transport
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PK	Primary Key
SD	*Software Development* - desenvolvimento de software
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
SSD	Solid State Drive - unidade de estado sólido
TI	Tecnologias da Informação
UML	Unified Modelling Language
vCPU	Virtual Central Processing Unit

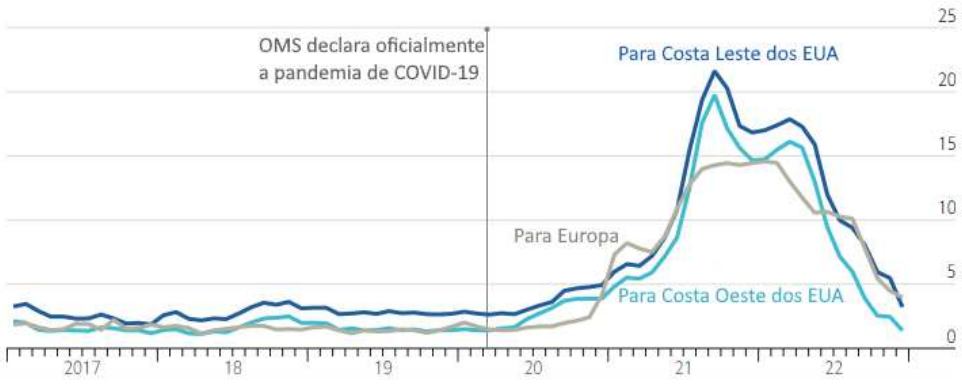
SUMÁRIO

1 INTRODUÇÃO

As relações de compra e venda entre consumidor e mercado vêm sofrendo drásticas mudanças com a digitalização e automatização de processos do comércio de artigos de consumo. A expansão do *e-commerce* (comércio digital) é um fenômeno mundial e está impactando diretamente nas decisões de compra e no comportamento dos consumidores. E não apenas da população mais jovem, mas também para as pessoas que não necessariamente cresceram tendo o comércio digital como sendo a norma. O consumidor tem a possibilidade de, com alguns cliques, procurar as melhores ofertas de produtos específicos do seu interesse que agora são vendidos por lojas ou comerciantes de outras cidades, estados, países e até até mesmo continentes.

Essa explosão de popularidade do *e-commerce* se tornou viável com a ajuda de vários fatores, incluindo um forte crescimento da frota marítima de transporte (TRADE; DEVELOPMENT, 2022) e uma forte redução nos custos do transporte transoceânico após a pandemia de COVID-19 (ECONOMIST, 2023), retornando aos níveis pré-pandêmicos e acumulando uma queda relativa de cerca de 85% para algumas regiões, dados constatados na Figura 1:

Figura 1 – Custo de transporte de um contêiner de 40 pés com origem na China para diversas regiões do planeta, em milhares de dólares.



Fonte: Adaptado de (ECONOMIST, 2023).

Com isso, os consumidores estão munidos de arma poderosa: agora têm a oportunidade de escolha, pois os comerciantes estão concorrendo com o mundo inteiro. Se o consumidor encontrar uma oferta de um produto que deseja, porém vendido por uma loja distante, ele ainda assim pode efetuar a compra, apenas aguardando alguns dias para a sua chegada, existindo casos de plataformas que oferecem fretes para o mesmo dia, dada uma compra efetuada até determinado horário.

Isso gera, conforme citado anteriormente, uma competitividade nunca antes vista entre as empresas de comércio e varejo, assim como outras empresas de áreas

correlatas. Portanto, estas empresas estão passando a reagir perante este movimento. Estão percebendo a necessidade de coletar e fazer uso da maior quantidade de informação possível sobre as compras já realizadas pelos seus clientes (MUNHOZ DE MEDEIROS; HOPPEN; MAÇADA, 2020, p. 2–3). Quem quer manter as portas abertas precisa saber o que seu concorrente está fazendo, o que seu cliente está querendo e para onde o dinheiro e a economia estão indo.

Assim, de forma genérica, surge a necessidade de uma gestão mais inteligente, que objetiva melhorar a experiência de compra oferecendo preços mais competitivos e ofertas relevantes para o perfil de cada consumidor, também almeja otimizar a cadeia produtiva e de logística para eliminar gargalos ou situações de estoque excessivo ou insuficiente para uma demanda variável. Precisa impulsionar a eficiência operacional, e acima de tudo ampliar as suas margens de lucro. Cumprir com esses objetivos dá à empresa que bem utiliza os dados que coleta uma vantagem imensa sobre os concorrentes, dado que poderá oferecer melhores condições de compra para seus clientes com ofertas vantajosas tanto para o cliente (no sentido de comodidade, interesse nos produtos e principalmente em preço) quanto para a empresa (menores custos operacionais, maior quantidade de vendas, mais engajamento, redução da taxa de desistência de compras (*churn*)) (MUNHOZ DE MEDEIROS; HOPPEN; MAÇADA, 2020, p. 7–8).

No escopo do projeto em questão, as necessidades giram em torno da eficiência operacional na realização de um conjunto de processos chamados corriqueiramente de *analytics*, que embora existente e funcional hoje no cliente, possui algumas limitações claras.

Na implementação atual (pré-projeto) a área de análise de desempenho de vendas é feita de forma informal e totalmente sob demanda, não havendo padronização nas rotinas, nos prazos e tempos de geração de relatórios e atualizações no painéis de inteligência de negócio. Todas as consultas necessárias para uma análise são exportadas manualmente em arquivos `.csv` dos sistemas transacionais ou outros sistemas legado completamente descontinuados e defasados. Esses arquivos são tratados à mão e inseridos nas aplicações desejadas ou avaliados caso a caso. Esse tratamento manual impossibilita, também, que a empresa tome passos em direção à um uso mais avançado dos seus dados na forma de modelos de predição e modelos estatísticos para avaliar o comportamento passado de vendas, dado o alto volume de informação necessária para bons resultados e inviabilidade de fazer isso à mão.

No contexto de mercado atual, a empresa cliente do projeto já possui uma relativa alta dominância no setor de varejo de moda, mas percebe forte competição por atores externos também no ramo da moda (tanto na forma de competitividade com outras empresas nacionais quanto, em especial, internacionais) que podem oferecer peças com melhor custo-benefício, melhores condições de compra, descontos atrativos e/ou coleções que chamam melhor a atenção do cliente final. Sendo assim, surge a

necessidade de tentar compreender melhor o perfil e comportamento do consumidor na tentativa de manter ou até mesmo expandir sua dominância e permanecer relevante no mercado brasileiro frente à crescente competitividade por outros atores no ramo.

1.1 OBJETIVO GERAL

O objetivo geral do projeto aqui descrito é de desenvolver uma plataforma de integração de dados de múltiplos sistemas heterogêneos no formato de um DW. Essa plataforma deve ser capaz de receber dados crus, processá-los (realizar agregações, cálculos, deduplicações e agrupamentos), disponibilizando os dados processados em tabelas para eventuais consultas (*ad hoc* (sob demanda)) e para integração com outros sistemas, especialmente os painéis de inteligência de negócio (*dashboards*) gerenciais e para modelos de aprendizado de máquina. A entrega referente aos painéis de inteligência de negócios é o objetivo chave da execução deste projeto, titulado BI Carteiras. O autor deverá buscar entender os requisitos, mapear fontes de dados e processá-los de forma recorrente até que atinjam um formato facilmente interpretável para ser consumido nestes sistemas ou pelas pessoas-chave do negócio.

Com a entrega do projeto, a intenção é permitir aos analistas e executivos o acesso às bases de dados e *dashboards* que resumam e apresentem informação do como e pelo quê é composto o faturamento da empresa cliente, com a possibilidade de avaliar quais são os *top performers* no quesito faturamento, seja na forma de avaliar os produtos mais bem-sucedidos, produtos com a maior margem de lucro, clientes com maiores *tickets*, comportamento de compra dos clientes (qual cliente tende a comprar qual tipo de produto), quais são as regiões que mais e menos faturam normalizados pela população, quais os melhores vendedores e outros indicadores derivados à partir dos dados transacionais do sistema Enterprise Resource Planning (ERP) SAP que operacionaliza a empresa. Esses indicadores, no momento inicial pré-projeto, são calculados manualmente de forma *ad hoc*, e portanto, costuma tomar até duas semanas até que um relatório requisitado esteja pronto.

No escopo inicial do projeto, foram determinadas algumas tabelas críticas para o atingimento desses objetivos, são elas: uma tabela fato ordens de venda, contendo todos os registros de vendas realizadas na empresa, e dimensões para clientes, representantes de venda, fretes, faturas, remessas, regionais, filiais e produtos que ajudam a caracterizar as vendas efetuadas. O objetivo final de entrega no escopo tratado é a disponibilização para consulta das informações contidas nesse contexto de vendas e faturamento, possibilitando a provação de hipóteses de negócio e habilitando a descoberta de novas correlações que ajudem a decidir quais passos tomar na expansão do negócio, no aumento do volume de vendas, na cativação de mais clientes e outros indicadores que possam ajudar a manter ou expandir sua participação de mercado frente à concorrência de outros grandes nomes nacionais e internacionais no varejo

de moda.

1.1.1 Caracterização do problema e resultados observados com a implementação do projeto

Com a implementação do projeto de criação desse DW analítico contendo dados de venda, os resultados atingidos incluem principalmente a agilização dos processos de tomada de decisão e de conhecimento intrínseco do *modus operandi* dos canais de venda da empresa, reduzindo o período necessário na obtenção de relatórios do desempenho de vendas de um intervalo de até duas semanas (com relatórios específicos para datas escolhidas), para se tornar um componente diário e completo, isto é, todos os dias as bases de dados analíticas entregues seriam repopuladas com a totalidade de dados coletados com um atraso máximo de 24 horas, e disponibilização imediata a qualquer momento.

Com essa infraestrutura de processamento em produção, o atrito gerado entre os times internos (executivos e analistas com o corpo técnico de dados) na requisição de excertos dos dados deve ser zerado, não mais sendo necessária a abertura de chamados demorados, nem a execução de tarefas manuais pela equipe técnica para extrair, processar e entregar bases resumidas. Outro ganho observado foi a redução nos erros encontrados pelos analistas e executivos pois os processos de extração e refinamento dos dados agora se torna automatizado, padronizado e validado.

Sendo assim, as entregas definidas neste projeto habilitam uma nova etapa na maturidade de dados do negócio (na forma de Artificial Intelligence (AI)) e automatiza, simplifica, melhora a frequência e habilita maior acurácia nos fluxos de análise previamente existentes.

1.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral proposto e entregar os resultados observados, definem-se os seguintes objetivos específicos:

1. Planejar os testes de carga e previsão de volumes esperados no DW;
2. Implementar todos os fluxos de ingestão dos bancos de dados de origem utilizando CDC, a ser discutido à frente, permitindo um registro histórico de cada transação de cada tabela de cada origem do sistema SAP, rastreando todas as operações que ocorrem na empresa;
3. Desenvolver todos os fluxos de dados (*pipelines*) necessários para o processamento de algumas tabelas críticas, como as tabelas de dados de ordens de venda, remessas, faturas e outras mencionadas na Seção 1.1;

4. Implementar uma lógica de carga incremental para o processamento das tabelas, permitindo que os relatórios que antes tomavam até duas semanas a partir da data de requisição sejam automatizados e consultáveis a cada dia, com no máximo 24 horas de atraso;
5. Disponibilizar dados processados em uma camada de consulta e disponibilização democrática para analistas;
6. Realizar o gerenciamento de acesso e controle de identidade dos usuários às tabelas processadas e;
7. Salvar registros (*logs*) de processamento para otimização de custos e tempo.

1.3 EQUIPE DE PROJETO

O projeto descrito neste PFC iniciou-se com algumas movimentações e negociações entre a BIX Tech e a empresa contratante (maiores detalhes no Capítulo 2) ainda em Novembro de 2022, com ajustes e fechamentos de contrato e negociação de preços relativos ao consultor (autor dessa monografia), além de organização de um time interno, composto primariamente pelo autor como engenheiro de dados responsável, um Product Owner (que atua como *scrum master*, organiza contatos com outros times e repassa atualizações aos gerentes de tecnologia e outros interessados) e um especialista em inteligência de negócios, ambos colaboradores internos ao cliente.

Externos ao time principal estão alguns executivos, um líder técnico e toda uma equipe de analistas de negócio focados no desenvolvimento de *dashboards* e no conhecimento disponibilizado a partir de consultas nas bases de dados.

1.4 ESTRUTURA DO DOCUMENTO

No Capítulo 2 é feita uma descrição da empresa na qual o PFC foi desenvolvido, a BIX Tech. Além disso, discute-se um pouco sobre o modelo de negócios da empresa, processos e ritos, além de brevemente apresentar, de forma anônima, a empresa cliente da BIX Tech requisitante do projeto aqui discutido e alguns dos seus processos internos que serão substituídos pela solução entregue.

No Capítulo 3 é apresentada uma fundamentação teórica, em que são discutidas as melhores práticas, conceitos da engenharia de *software* e especialmente da engenharia de dados que são a base sobre a qual o projeto é construído.

No Capítulo 4 são expostos diagramas e fluxos de informação usados no cumprimento dos objetivos gerais e específicos do projeto.

No Capítulo 5 o projeto em si é descrito em detalhes. São discutidos os maiores desafios de implementação, as decisões tomadas, consequências, limitações e custos envolvidos no projeto BI Carteiras.

No Capítulo 6 é feita uma análise crítica de resultados envolvendo tanto o quesito técnico de engenharia de dados, quanto o impacto da implementação do projeto no cliente.

No Capítulo 7 é feita uma conclusão do PFC, resumindo o que foi feito, propondo sugestões de melhoria e trabalhos futuros no âmbito de engenharia de dados.

2 EMPRESAS E SISTEMAS ENVOLVIDOS

Neste capítulo serão apresentadas as empresas envolvidas no projeto, assim esclarecendo um pouco mais do contexto em que o mesmo está inserido.

2.1 BIX TECH

A BIX Tech Corp é uma empresa estado-unidense registrada na Flórida. A BIX Tech atua como o braço internacional da empresa brasileira BIX Tecnologia, registrada em Florianópolis e fundada em 2014 por uma equipe de cinco sócios que incluem alguns egressos do curso de graduação em Engenharia e Controle e Automação da Universidade Federal de Santa Catarina. A BIX é uma empresa *startup* que presta serviços de consultoria para outras empresas em quatro grandes áreas chamadas internamente de *chapters*:

- *Business Intelligence* - inteligência de negócios (BI): área responsável pelo desenvolvimento de painéis analíticos (*dashboards*), a interpretação dos dados, auxílio na tomada de decisão, relatórios automáticos;
- *Data Science* - ciência de dados (DS): área referente ao uso de dados em conjunto com redes neurais, aprendizado de máquina, visão computacional e outros métodos estatísticos de inferência;
- *Data Engineering* - engenharia de dados (DE): que cuida da parte de infraestrutura, processamento, armazenamento, catálogo, disponibilização e produtização dos dados de forma apropriada;
- *Software Development* - desenvolvimento de software (SD): prestando consultoria para desenvolvimento de sistemas em geral, aplicativos de celular, interfaces de aplicações, sistemas *front-end* e *back-end*, websites e outros softwares.

Tendo as áreas de atuação em mente, para cada projeto em andamento em algum cliente da BIX, alocam-se consultores de uma ou mais áreas de atuação, a depender do tipo de contrato firmado com o cliente. De forma geral, os projetos da BIX Tech são regidos por contratos de duas categorias:

- Escopo fechado: em que o cliente busca a BIX com um escopo definido e relativamente fixo. As entregas são mapeadas com ampla antecedência e os prazos são mais rígidos. Contudo, o consultor não atua em outras áreas ou demandas eventuais ou;
- *Staff augmentation*: em que o consultor alocado atua como um colaborador da empresa cliente, como um verdadeiro aumento de equipe. Este tipo de contrato é

mais flexível, suscetível a renovações mais frequentes, e permite que o consultor atue em várias frentes ao mesmo tempo, seja resolvendo problemas, atuando em diversos projetos em paralelo dentro do cliente e/ou difundindo conhecimento para os colaboradores internos.

Por mais que o projeto em pauta neste trabalho possua um escopo relativamente bem definido, o contrato que rege a permanência do autor no cliente é do tipo *staff augmentation*, haja vista que embora bem definido, o escopo é muito amplo, os desenvolvimentos são relativamente lentos e complexos, o volume de dados é grande e a quantidade de problemas, regras de negócio e entrevistas a serem realizadas não são facilmente mapeáveis nem previsíveis.

2.2 A CONTRATANTE

Por restrições de um acordo de confidencialidade firmado entre o autor deste trabalho e a BIX Tech, alguns detalhes da empresa contratante não poderão ser compartilhados. Essa restrição se dará ao longo deste documento, porém sem prejudicar o entendimento completo dos assuntos tratados. As informações mais gerais podem ser compartilhadas para que haja uma boa contextualização do trabalho em si.

Dito isso, a contratante é uma empresa que incorpora algumas marcas do ramo do varejo de moda muito presentes no Brasil e relativamente presentes no exterior. Seu porte é grande, faturando mais de um bilhão de reais por ano e contando com um quadro extenso de mais de cinco mil funcionários.

2.2.1 Fluxos de análise atuais

A empresa contratante, dado seu tamanho considerável, é dividida em diversos times com diferentes áreas de atuação. Existem, de forma geral, dois grandes setores: uma área dedicada ao design e manufatura têxtil, e uma área dedicada exclusivamente ao varejo, contando com lojas parceiras, fraqueadas e lojas próprias da empresa cliente que fazem a revenda dos produtos de moda para o cliente final (consumidores pessoa física ou mesmo para outras lojas revendedoras). Como o enfoque do projeto, conforme discutido no Capítulo 1, é no desempenho da máquina de vendas, o projeto deve se atentar às fontes de dados e fluxos de análises que são realizados no âmbito do varejo de moda.

Por fluxo de análise, entende-se como sendo todas as etapas necessárias para se tomar uma decisão de negócio ou tomar conhecimento de um fato ocorrido a partir da avaliação criteriosa de dados coletados. Atualmente, na empresa cliente do projeto, esses fluxos de análise são feitos usando como base os dados contidos diretamente nos sistemas transacionais (diretamente de bancos de dados operacionais) ou então de um sistema de DW legado e descontinuado, que não mais atende à todas as análises

feitas e que serve apenas para alguns fluxos, e ainda assim, os dados lá contidos estão defasados em relação aos sistemas transacionais, invalidando conclusões que possam ser tomadas pois grande parte do que está contido nesse sistema está errado.

Com isso, atualmente quem deseja realizar qualquer tipo de análise de vendas (novamente, determinar informações como quais são os produtos que mais vendem, quais possuem a maior margem de lucro, preferência de compra dos clientes, saber determinar e quantificar a eficácia e viabilidade de promoções no faturamento da empresa, ranquear lojas em volume de vendas, determinar satisfação de clientes no pós-compra) são todas análises válidas e executadas por equipes internas de analistas, engenheiros e executivos, cada qual com um objetivo (times focados em vendas Business-to-Business (B2B), outros em Business-to-Consumer (B2C), outros focados no desempenho das coleções lançadas (primavera-verão, outono-inverno, edições limitadas, etc.). Essas análises são realizadas, atualmente, com a cooperação entre os analistas e as equipes internas de dados (administradores de bancos de dados, equipe de Tecnologias da Informação (TI)) que exportam manualmente os dados necessários para essas análises, tomando como fonte os sistemas transacionais e/ou o DW legado mencionado previamente.

Esse tipo de interação é feita a partir de chamados em um sistema interno de *tickets* de chamado, que gera uma forte burocracia na geração dos relatórios. Além do tempo dependido na emissão e despacho de chamados, existe a demora na entrega das bases de dados (arquivos exportados dos sistemas mencionados), fortes limitações técnicas no manejo desses dados (como por exemplo o limite de cerca de 1 milhão de registros do *Microsoft Excel*) e a demora da própria exportação dos dados, pois os sistemas transacionais não são otimizados para consultas necessárias para as análises desejadas.

2.2.2 Implementação de um novo DW como solução aos gargalos atuais

Como proposta de solução aos gargalos de integração de dados e na interação entre times atuais apresentados na Seção 2.2.1, a equipe da BIX Tech faz a proposta de alocar pelo menos um consultor de engenharia de dados, o autor, para atuar em conjunto com o cliente na criação de uma nova infraestrutura capaz de receber os dados relevantes dos sistemas transacionais, de processá-los e transformá-los em tabelas resumidas que estarão disponíveis para consulta em uma plataforma na nuvem. Detalhados ao longo deste relatório, os conceitos e práticas aplicados na construção de um DW farão com que os analistas e executivos (e até mesmo outros sistemas computadorizados) possam consultar uma única fonte de dados para extrair a informação desejada.

Isso remove a necessidade de interação entre a equipe analista e o corpo técnico para obtenção dos dados, agilizando consideravelmente (de até duas semanas

para quase instantâneo) o acesso aos dados desejados, bastando que se acesse e autentique-se na plataforma de consulta para visualizar, extrair ou até mesmo fazer análises utilizando a linguagem Structured Query Language (SQL) para manusear e controlar a informação que deseja se ver.

Assim, não mais será necessária a abertura de chamados, não mais será enfrentado um grande atraso na entrega das bases de dados, pois elas já estarão prontas para consulta diariamente, e todo o fluxo para entregar os dados à quem se interessa será feito de forma recorrente, automática e previamente validada. A expectativa, portanto, é aumentar a produtividade das equipes responsáveis pela tomada de decisões. Alguns relatos dados pelo próprio cliente à BIX Tech informam que alguns colaboradores-chave para o negócio despendiam, semanalmente, um dia inteiro dedicado apenas na operacionalização de alguns relatórios, isto é, um dia inteiro de trabalho gasto em atividades repetitivas, manuais e de baixo valor agregado que podem ser automatizadas com um DW (e a arquitetura que o suporta) bem implantado.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são discutidas as tecnologias utilizadas, conceitos teóricos e terminologias, elementos considerados cruciais para a compreensão plena das atividades realizadas na execução deste PFC.

3.1 DADOS, INFORMAÇÃO E TEORIA DO CONHECIMENTO

Ao longo deste documento, a palavra “dado” é referenciada constantemente, e evita-se o uso termos como “informação” deliberadamente. Isso se dá por uma clara distinção semântica que é muito bem explicada por (ACKOFF, 1989), e amplamente usada na indústria e academia:

> Dados são símbolos que representam propriedades de objetos, eventos ou ambientes em que estão inseridos. São produtos da observação. (...) Dados, como minérios brutos, não possuem valor agregado até que sejam processados [entenda-se, transformados] em uma forma usável. Assim, a diferença entre dados e informação é funcional, não estrutural, mas dados são normalmente reduzidos quando são transformados em informação. (...) Conhecimento, [por sua vez], pode ser obtido de duas formas: por transmissão de alguém que já o tenha, ou extraindo-o a partir de experiência. De qualquer forma, a aquisição de conhecimento é chamada de aprendizado. (ACKOFF, 1989, p. 3–4, tradução livre do autor, colchetes indicam palavras inseridas para melhor contextualização).

Essa definição, embora tendo mais de trinta anos, permanece até hoje caracterizando uma distinção clara entre dado, informação e conhecimento. Por dados, compreendem-se fatos conhecidos e/ou mensuráveis, com um significado implícito, mas que precisa de contexto para ser compreendido totalmente ou para ter um significado completo. Um dado pode existir, de forma geral, em duas formas: dados quantitativos (expressos em termos numéricos, como valores de medições) e dados qualitativos (de natureza descritiva, como textos, interpretações, propriedades de um elemento).

Já a informação é alguma conclusão ou suposição que pode ser provada ou testada a partir de um dado ou, comumente, um conjunto de dados. É bastante comum precisar utilizar uma grande quantidade de dados para alcançar uma única conclusão, e no contexto deste trabalho, uma conclusão significa uma decisão de negócios potencialmente impactante na competitividade, eficiência e faturamento de uma empresa.

O conhecimento, por sua vez, é algo muito mais intrínseco, que depende de contexto, repetibilidade e consistência de informações que concordam entre si. Para melhor compreender a diferença entre cada um desses termos, segue na Tabela 1 um exemplo de um conjunto de dados quantitativos:

Tabela 1 – Tabela com dados sem contexto para ilustração didática.

25.2
24.7
25.7
18.4
20.6
13.5
14.3
16.4
19.3
19.5
16.5
24.5

Fonte: Intencionalmente não divulgada neste momento.

Sem contexto, fontes nem unidades de medidas esta tabela não possui qualquer valor agregado, ainda que se possa palpitar quanto ao que está sendo representado nesses dados descontextualizados, seu verdadeiro significado só é revelado se soubermos um pouco mais sobre a tabela.

Explicita-se a diferença entre dados e informação quando se compara a Tabela 1 com a Tabela 2:

Tabela 2 – Temperatura média no primeiro dia de cada mês de 2022, de acordo com a estação meteorológica A806, em Florianópolis.

DATA	TEMPERATURA MÉDIA (°C)
01/01/2022	25.2
01/02/2022	24.7
01/03/2022	25.7
01/04/2022	18.4
01/05/2022	20.6
01/06/2022	13.5
01/07/2022	14.3
01/08/2022	16.4
01/09/2022	19.3
01/10/2022	19.5
01/11/2022	16.5
01/12/2022	24.5

Fonte: (INMET, 2022)

Evidenciando que a quantidade de detalhes conhecidos sobre um conjunto de dados é diretamente proporcional ao valor que se pode extrair do mesmo. Com um pouco de contexto, é possível saber não só o que está sendo dito, mas também potencialmente o como, quando e onde.

Agora, como discernir a informação do conhecimento? É uma pergunta um pouco mais complexa, mas ainda se pode refletir sobre quando nos perguntamos “o que realmente temos de informação quando analisamos completamente os dados da Tabela 2?”.

Sabemos que não se pode extrapolar os dados de apenas um ano para determinar o clima da cidade de Florianópolis pois existe muita variabilidade nas temperaturas ao longo do tempo. Eventos climáticos como a intensidade do El Niño, ondas de calor, aquecimento global, correntes marítimas e muitos outros afetam a temperatura local (CAI *et al.*, 2020, p. 215–231). Como mencionado anteriormente, o conhecimento é a aplicação da informação com o adicional de uma certa recorrência, possui consistência, e muitas vezes toma anos para ser construído. Assim, para uma informação se tornar conhecimento, ela precisa ser colocada à prova e se mostrar válida em múltiplos casos e/ou ao longo de um certo período de tempo, extraindo o conhecimento a partir da experiência, ou, em último caso, repassado por alguém que já o adquiriu (ACKOFF, 1989, p. 4).

3.2 BANCOS DE DADOS TRANSACIONAIS E *DATA WAREHOUSING*

Na Seção 3.1 foram vistas algumas tabelas, sendo a mais completa contendo registros (coloquialmente “linhas”), colunas, um cabeçalho, fonte e uma descrição. Não coincidentemente, nas tecnologias de informação de hoje existem ferramentas conhecidas como bancos de dados que catalogam, armazenam e permitem consulta à dados em tabelas muito semelhantes às apresentadas previamente. Assim, um banco de dados é um conjunto coesa de dados com um significado fornecido por contexto, o que invalida uma coleção de dados não correlatos como um banco de dados.

Introduz-se aqui também o termo SGBD, usado na indústria para se referir à toda uma categoria de software para sistemas que mantêm e operacionalizam bancos de dados. Um SGBD é um software de uso geral que atua na definição de um banco de dados, com tipagem de dados, estruturas e restrições a serem aplicadas aos dados. Também é responsável pelo gerenciamento da construção física junto com o sistema operacional (armazenamento em disco rígido ou dispositivos de estado sólido) e também na manipulação de dados, fornecendo uma ou mais linguagens/comandos que permitem que um usuário manipule a base de dados. Existem outros domínios como o gerenciamento de acesso, compartilhamento de informação com sistemas externos e automatização de rotinas que também estão sob responsabilidade do SGBD (ELMASRI; NAVATHE, 2015, p. 6).

Além disso, os bancos de dados podem ser categorizados em duas grandes formas: **transacionais** ou **analíticos**. Os da primeira categoria são pensados, modelados e construídos pensando em alta performance e escalabilidade no registro, modificação ou deleção de transações independentes, como um cadastro de um cliente e suas informações de contato, um registro de uma compra com o produto e valores associados, ou mesmo a gravação do horário de entrada de um funcionário em seu trabalho. Este tipo de operação é extremamente comum no mundo moderno, em que muitas das operações que são realizadas são de natureza transacional, em que um fato ocorre e

pode-se desejar registrá-lo para consulta futura.

Já os bancos de dados analíticos possuem um foco muito maior na consulta de altos volumes de dados e na realização de operações, agregações e filtragem de registros específicos. Embora um banco transacional seja capaz de agregações e um analítico também seja capaz de operar como um transacional, há uma grande diferença técnica na implementação desses dois tipos de bancos de dados e questões de desempenho que serão exploradas ao longo desta seção. Um tipo específico de banco de dados analítico que possua um alto nível de adoção em uma corporação, ou que agrega dados de muitos contextos e que é usado como fonte de dados para responder questionamentos de negócio é o que se chama armazém de dados, popularizado como DW.

Exemplos de SGBDs comuns no mercado de tecnologias da informação hoje incluem Oracle, MySQL, Microsoft SQL Server e PostgreSQL (KOLONKO, 2018), e todos esses bancos de dados possuem uma interface de interação com usuários na forma de SQL. Essa é uma linguagem interpretada pelo SGBD e converte comandos textuais que operacionalizam um banco de dados.

Os bancos operacionalizam sistemas ERP - sistema de gerenciamento de recursos empresarial, funcionando como a camada de persistência para as transações que ocorrem no dia a dia de uma grande empresa. Alguns comandos básicos em SQL são:

- `SELECT name, price FROM products WHERE category = 'Electronics';` usa a palavra-chave reservada da sintaxe SQL `WHERE` (usada para filtrar o resultado). Essa consulta retorna as colunas `name` e `price` de uma tabela `products`, porém apenas os produtos que possuem a coluna `category` igual à “Electronics”;

- `INSERT INTO customers (name, age) VALUES ('Fellipe', 24);` usa palavras-chave reservadas `INSERT INTO` (usada para inserir um ou mais registros em uma tabela) usada em conjunto com a cláusula `VALUES`. Essa consulta insere na base de dados, especificamente na tabela `customers` uma linha com dados referentes ao nome e idade do autor. Essa tabela deve ter sido previamente criada e bem definida com outros comandos SQL, como o `CREATE TABLE customers [...]`.

3.2.1 Bancos de dados transacionais relacionais

Dentro da categoria de bancos de dados transacionais existem diversas subdivisões (relacionais, em memória, grafos, distribuídos e muitos outros), cada qual especializada em resolver algum tipo de problema real, implementado por um software de SGBD diferente e que conta com vantagens e desvantagens para determinadas aplicações, não existindo uma solução unânime para qualquer situação. Ainda assim os bancos de dados relacionais dominam grande parte do mercado devido à alta apli-

cabilidade em muitos dos problemas de negócio: este tipo de banco de dados utiliza-se dos conceitos de tabelas que possuem relacionamentos entre si para organizar e armazenar dados de forma estruturada. Esta abordagem relacional foi proposta por (CODD, 1970) e permanece relevante desde então.

Os dados propriamente ditos são armazenados em diversas tabelas, com cada tabela possuindo um nome, uma estrutura de colunas (usualmente chamado *schema*) e um conjunto de registros (*rows*). Essas tabelas são projetadas pelos programadores e engenheiros de forma a representar entidades do mundo real, como cadastros de clientes, produtos, pedidos, fatos ocorridos, entre outros, e esta modelagem de entidades é parte crucial no desempenho e usabilidade do sistema.

Portanto, de forma geral, uma tabela em um SGBD relacional pode ser representada de forma muito similar à apresentada na Tabela 2. A Figura 2 exibe uma tabela criada em um banco de dados relacional, representando informações de produtos disponíveis para venda em uma empresa fictícia. Na modelagem apresentada, a tabela `products` possui diversas colunas, dentre elas:

- Uma coluna identificadora para cada produto, dada por `product_id`;
- Uma coluna para registrar o nome do produto, em `name`;
- Uma descrição para cada produto, em `description`;
- O preço de cada produto, em `price`;
- Outros metadados contendo as datas de registro e modificação nas colunas remanescentes.

Figura 2 – Tabela `products` contendo informações de produtos de uma empresa fictícia.

product_id	name	description	price	registration_datetime	last_updated_at_datetime
1	4K Ultra HD TV	55-inch 4K Ultra HD Smart TV with HDR	799.99	2023-10-06T17:00:00	2023-10-06T17:30:00
2	Bluetooth Speaker	Portable Bluetooth speaker with rich sound	59.99	2023-10-06T18:00:00	2023-10-06T18:30:00
3	Tablet Pro	High-performance tablet with 10.2" display	399.99	2023-10-06T19:00:00	2023-10-06T19:30:00
4	Coffee Maker	Automatic coffee maker with built-in grinder	89.99	2023-10-06T20:00:00	2023-10-06T20:30:00
5	Gaming Console	Next-gen gaming console with 4K gaming support	499.99	2023-10-06T21:00:00	2023-10-06T21:30:00
6	Wireless Mouse	Ergonomic wireless mouse with adjustable DPI	29.99	2023-10-06T22:00:00	2023-10-06T22:30:00
7	Smart Thermostat	Wi-Fi-enabled smart thermostat for home automation	149.99	2023-10-06T23:00:00	2023-10-06T23:30:00
8	Fitness Tracker	Fitness tracker with heart rate monitor and GPS	79.99	2023-10-07T00:00:00	2023-10-07T00:30:00
9	Robot Vacuum	Smart robot vacuum with mapping and navigation	249.99	2023-10-07T01:00:00	2023-10-07T01:30:00
10	Wireless Router	High-speed wireless router for seamless connectivity	129.99	2023-10-07T02:00:00	2023-10-07T02:30:00
11	Digital Camera	Advanced digital camera with 24MP sensor	699.99	2023-10-07T03:00:00	2023-10-07T03:30:00
12	Portable Charger	Compact portable charger for on-the-go charging	39.99	2023-10-07T04:00:00	2023-10-07T04:30:00
13	Smart Home Hub	Central smart home hub for controlling devices	149.99	2023-10-07T05:00:00	2023-10-07T05:30:00
14	Electric Scooter	Foldable electric scooter for urban commuting	299.99	2023-10-07T06:00:00	2023-10-07T06:30:00
15	Outdoor Grill	Stainless steel outdoor grill with multiple burners	399.99	2023-10-07T07:00:00	2023-10-07T07:30:00
16	Wireless Keyboard	Slim and stylish wireless keyboard with backlight	49.99	2023-10-07T08:00:00	2023-10-07T08:30:00
17	Noise-Canceling Headphones	Premium noise-canceling headphones for immersive aud	249.99	2023-10-07T09:00:00	2023-10-07T09:30:00
18	Home Security Camera	Wireless home security camera with motion detection	79.99	2023-10-07T10:00:00	2023-10-07T10:30:00
19	Smart Refrigerator	Smart refrigerator with touchscreen and app integration	1,499.99	2023-10-07T11:00:00	2023-10-07T11:30:00
20	Bluetooth Earbuds	Wireless Bluetooth earbuds with long battery life	69.99	2023-10-07T12:00:00	2023-10-07T12:30:00

Fonte: Acervo do autor.

3.2.1.1 Bancos de dados analíticos

Diferentemente dos bancos de dados transacionais, que são otimizados para rápidas escritas e concorrência de transações com garantia de estabilidade e integridade de dados, os bancos de dados analíticos possuem um foco muito maior no desempenho de consultas e no processamento de altos volumes de dados, e seu uso está muito mais relacionado à análise de dados e aplicações de inteligência de negócios do que estritamente na operação ou execução das regras de um negócio propriamente dito.

Em consequência dessa distinção, surgem dois termos para categorizar as tecnologias da informação que são focadas no processamento transacional e as focadas em análise exploratória e analítica de dados: OLTP é o termo técnico usado para identificar uma ferramenta focada no processamento de transações, enquanto OLAP é a terminologia para tecnologias analíticas (CONN, 2005, p. 1–3).

Também é importante ressaltar que um banco de dados analítico possui, de forma geral, as mesmas características de implementação de um banco de dados transacional relacional, isto é, também se utiliza dos conceitos de tabelas, de chaves primárias e estrangeiras na associação entre essas tabelas, também pode ser representado graficamente com um diagrama de entidades e relacionamentos e é gerenciado por um software de SGBD. Em sua essência, um banco focado em OLAP possui diferenças nas suas características construtivas quando comparado aos OLTP.

Uma característica particular é a modelagem dos dados dentro de um banco analítico, que consiste, usualmente, no modelo estrela. O modelo estrela nada mais é que uma forma de organizar as tabelas-resultado em tabelas fato (contendo as principais informações) e dimensões de suporte, que detalham ou complementam um registro na tabela fato. Seguindo esse raciocínio, pode-se exemplificar uma tabela fato como uma tabela de vendas, e cada venda possui um vendedor associado responsável pela efetuação ou conversão da venda, e detalhes sobre esse vendedor (cadastros) estariam em uma tabela dimensão. Cada venda também possui um cliente, que também é registrado em uma tabela dimensão.

Mas a principal diferença construtiva está no formato de armazenamento dos dados em si. Um banco costumeiramente OLTP é desenhado para funcionar, por detrás dos panos, como um banco linear. Isso significa que os dados são registrados linha a linha em disco, e uma mesma linha possui todos os seus dados (todas os valores das colunas desta linha) em um bloco contíguo no Hard Disk Drive - disco rígido (HDD) ou em um mesmo *chip* de memória *flash* nos Solid State Drive - unidade de estado sólido (SSD), o que torna a consulta à registros específicos uma operação relativamente rápida, dado que uma vez acessando o índice ou chave primária do registro, o restante dos dados estão em sequência no disco (ABADI; MADDEN; HACHEM, 2008, p. 1–4).

Enquanto isso, um banco OLAP registra seus dados em formato colunar, ou

seja, por mais que existam registros e colunas assim como em OLTP, as informações em disco estão agrupadas por coluna. Isso acaba facilitando e agilizando a realização de filtros e cálculos em cima de uma mesma coluna, como em agregações, somatórios, cálculos de mínimos, máximos, médias, agrupamentos e outros tipos de operações. Além disso, a maioria das tecnologias OLAP também permitem a indexação de registros e particionamento de tabelas (fisicamente separar partes de uma tabela em locais conhecidos em disco), o que também permite que essa tecnologia faça filtros e busque por registros específicas assim como em OLTP, com a desvantagem de ser mais lento na inserção de novos registros dado que como mencionado anteriormente, o SGBD teria que acessar diversos blocos em disco, visto que cada coluna está agrupada em um bloco diferente e não contíguo. Isso acaba invalidando o uso de bancos OLAP para aplicações transacionais devido à limitação de desempenho em tempo real, e portanto, ambas as tecnologias são necessárias e relevantes quando trata-se do mundo de dados e informação, cada qual com sua aplicação (RAMAKRISHNAN; GEHRKE, 2002, p. 677–679) e (PETROV, 2019, p. 38–40).

Para exemplificar as diferenças em disco de um sistema OLAP e OLTP, considera-se uma tabela de produtos que possui três colunas: um identificador único `product_id`, um nome de produto (`name`) e seu preço unitário (`price`), sendo que existem dois registros cadastrados no banco de dados, de acordo com o gabarito da Tabela 3:

Tabela 3 – Dados de exemplo para tabela de produtos.

product_id	name	price
17	Gaming PC	8999.90
42	Smartphone	2499.90

Fonte: Acervo do autor.

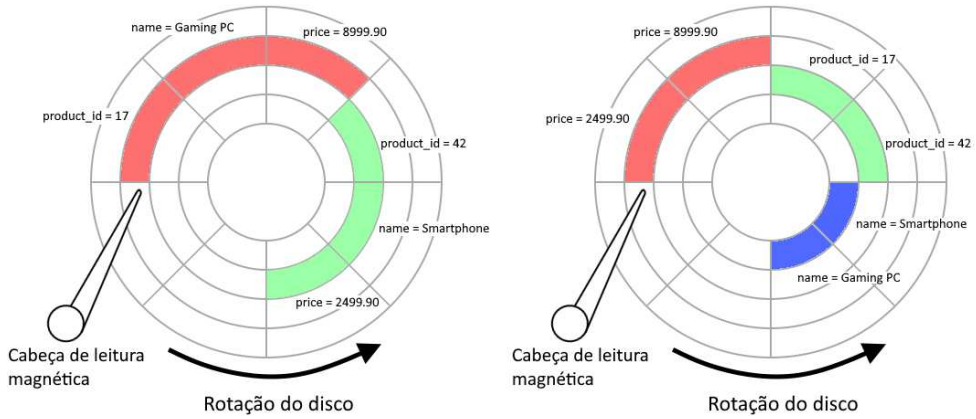
O autor propõe a seguinte representação em um disco rígido para ambas as categorias de sistemas, conforme apresentada na Figura 3.

Nesta representação, o caso da Figura 3a demonstra que todos os dados da primeira linha, referente ao *Gaming PC* estão contidos em sequência no disco (área pintada de vermelho da Figura 3a), e em seguida, os dados da segunda linha estão também contíguos do disco (área pintada de verde da Figura 3a), corroborando com o formato discutido anteriormente em que sistemas OLTP armazenam dados de forma linear (linha a linha). Percebe-se, portanto, que para realizar a pesquisa completa de um registro específico ou a inserção de um novo registro, a cabeça magnética do HDD precisa ler ou escrever apenas três setores que são vizinhos uns dos outros, o que é uma operação relativamente rápida para padrões de discos rígidos.

Já o caso apresentado na Figura 3b, percebe-se que estão povoadas também um total de seis setores do HDD, porém agora em três trilhas distintas. Isso acontece pois, como apresentado anteriormente, um sistema OLAP armazena dados de forma

colunar, otimizando operações em cima de colunas específicas. Tomando como exemplo um caso em que se deseja realizar uma operação simples para calcular o preço médio dos produtos cadastrados no banco de dados, no sistema OLAP, a cabeça magnética precisa apenas ler dois setores (os pintados em vermelho da Figura 3b) e computar a média. Caso essa mesma operação seja realizada em um sistema OLTP como o da Figura 3a, a cabeça magnética teria que recuperar os dados de preço em trilhas não-contíguas, pois para esse caso, os preços estariam em trilhas e setores (ou até em discos do HDD) diferentes e distantes entre si, o que faz com que o disco precise completar mais revoluções para recuperação dos dados, atrasando consideravelmente a consulta.

Figura 3 – Representação física em disco em sistemas OLTP comparados com OLAP.



(a) OLTP, trilhas contíguas do HDD contém registros completos.

(b) OLAP, trilhas contíguas do HDD contém colunas completas.

Fonte: Acervo do autor.

Por fim, na Tabela 4, um quadro de resumo comparativo entre ambas categorias de sistemas em alguns aspectos gerais:

Tabela 4 – Comparativo entre tecnologias OLAP e OLTP.

Aspectos	**OLAP (Online Analytical Processing)**	**OLTP (Online Transaction Processing)**
Objetivo geral	Análise e registro de dados históricos	Processamento de transações
Estrutura primária	Esquema estrela ou de fatos e dimensões armazenados fisicamente como colunas	Tabelas normalizadas e relacionadas entre si armazenadas em formato linear
Volume	Altos volumes de dados (muitos terabytes ou petabytes)	Usualmente menores volumes, até alguns poucos terabytes
Complexidade de consultas	Alta complexidade e custo	Baixa complexidade
Tempo de resposta	Usualmente mais longo	Rápido e ideal para aplicações de tempo real
Manipulação de dados	Foco primário em leitura, havendo escrita	Foco primário em escrita, mas há leitura
Casos de uso	Inteligência de negócios, relatórios, geração de *insights*, montagem de *data warehouses*, consultas gerais e fonte de dados para aplicações de AI - inteligência artificial e Machine Learning (ML) - aprendizado de máquina	Gerenciamento de transações e fatos ocorridos, aplicação de regras de negócio e operacionalização do negócio

Fonte: Acervo do autor.

3.2.2 Armazéns de dados

Finalizando essa subseção de metologias e tecnologias para armazenamento de dados tem-se um conceito relativamente recente (surgindo no início da década de 1990) quando comparado aos conceitos de bancos de dados clássicos que surgiram nos anos 1960. São mais de vinte anos de lacuna técnica, mas ainda assim os chamados armazéns de dados (tradução livre de DW - *data warehouse*) estão se popularizando, principalmente devido aos recentes avanços tecnológicos que possibilitaram um uso muito lucrativo na análise de dados, juntamente com avanços na computação em modelos de AI e ML (BEESETTY; PRAMOD; VINEET, 2021).

O termo DW foi usado pela primeira vez por Bill Inmon nos anos 1990, definindo-o como:

> Um Data Warehouse é uma coleção de dados orientada por assunto, integrada, não volátil e variável ao longo do tempo, em apoio às decisões da gerência. O armazém de dados contém dados corporativos granulares. Os dados no armazém de dados podem ser usados para muitos fins diferentes, incluindo ficar armazenados à espera de requisitos futuros que são desconhecidos hoje. (INMON, 2005, p. 29, tradução livre do autor, citação originalmente provinda da primeira edição do livro em 1990).

Os armazéns de dados, em sua essência, são plataformas abrangentes nas quais os usuários (sejam estes humanos ou outras ferramentas/softwares) podem consultar dados de forma performática, e portanto, utilizam-se de tecnologias OLAP

em sua implementação. Na construção de um DW, as equipes de engenharia de dados se reúnem com os *stakeholders* e outras partes interessadas nos dados para elaborar estratégias, prioridades e definir a modelagem do armazém, assim nascendo - por exemplo - tabelas que respondem de forma imediata algumas perguntas, como quantidade de vendas agrupadas por cada representante/coordenador de vendas, classificação de clientes que mais gastaram (com maior ticket médio, qual o maior ticket absoluto), frequência de compra (chamado reincidência) de um determinado cliente, custos com frete agrupados por unidade federativa, por tipo de encomenda, por peso da remessa e muitas outras perguntas que são do interesse da gerência em responder. Essas tabelas são criadas a partir do alto volume de dados dos bancos transacionais, e podem ser quebradas no DW em períodos arbitrários (fazendo as análises supracitadas mês a mês, ano a ano, semana a semana ou qualquer outro período).

Existem muitas formas de se implantar um DW, visto que existem diversas ferramentas que se encaixam e se relacionam para gerar a plataforma. Nomeadamente, esta plataforma necessita essencialmente de:

1. Uma camada de ingestão, que faz a replicação dos acontecimentos (operações de inserção, modificação e deleção de registros) dos bancos transacionais e disponibilização desses dados em uma camada de armazenamento para que possam ser processados;

2. Uma camada de armazenamento para todas as tabelas não processadas originadas do transacional e todas as tabelas processadas geradas pela camada de processamento (seja na forma de um servidor local com amplo espaço em disco ou armazenamento em nuvem);

3. Uma camada de processamento para transformar os dados crus do transacional em tabelas úteis para o negócio, que também pode ser hospedada localmente em um servidor, mas é costumeiramente usada na nuvem devido ao alto poder de processamento necessário;

4. Uma camada de disponibilização e acesso, que seja capaz de realizar o controle de acesso e permissões, e servir de interface para consulta aos dados guardados na camada de armazenamento (autenticação de usuários, gerenciar quem deve poder enxergar e consultar determinadas tabelas, controle de informações sensíveis e outras preocupações do gênero).

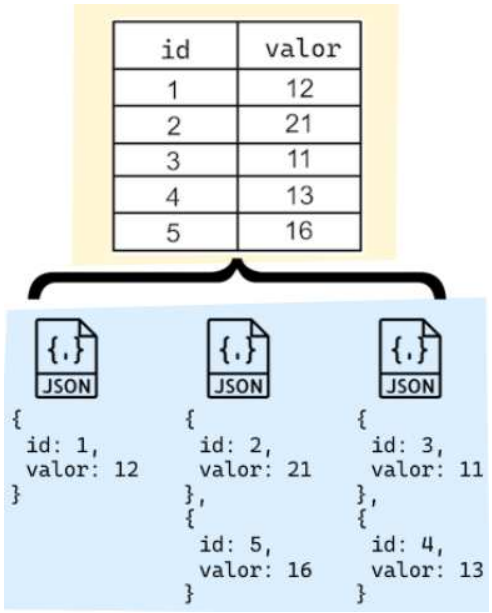
É a partir dessas quatro camadas principais que surgem conceitos como Extract, Transform and Load (ETL) - do inglês Extrair, Transformar e Carregar dados (discutido em detalhes na Seção 3.3), e algumas empresas desenvolvem e vendem ferramentas para atuar em cada um desses pontos, ou em algumas vezes, em todos esses pontos.

Ferramentas comerciais como Snowflake, QlikSense, Databricks e dbt são utilizadas no mercado para cuidar de uma ou mais camadas.

Além disso, existem variações no modelo de construção de um DW, a mais famosa sendo denominada *Data Lakehouse*. Este tipo de arquitetura é uma contração dos termos *Data Warehouse* e *Data Lake*, e surge como um híbrido entre estes tipos de formatos de armazenamento. Um *Data Warehouse* é, originalmente, tabular (assim como bancos de dados transacionais, envolvendo os chamados de dados estruturados), enquanto um *Data Lake* é a prática de se armazenar arquivos individuais (de diversos formatos como vídeos, imagens, texto, áudios, os chamados dados não-estruturados) em um grande repositório. Uma boa analogia para um *Data Lake* seria o próprio sistema de arquivos de um computador, capaz de armazenar diversos arquivos em pastas e permite o acesso à eles via uma interface gráfica, ou ainda, pode-se imaginar um grande repositório da ferramenta *Google Drive* como um *Data Lake*.

Sendo assim, um *Data Lakehouse* é um tipo de arquitetura de dados em que a camada de armazenamento é subdividida em uma camada física, onde se armazenam arquivos em algum formato predeterminado (usualmente `.parquet`, assunto da Seção 3.5) e uma camada de interpretação destes arquivos de forma inequívoca (ZAHARIA, M. A. *et al.*, 2021, p. 1–6). Assim, a Figura 4 exemplifica como esse tipo de formato pode funcionar, demonstrando como diversos arquivos de extensão `.json` da camada física (em azul), cuja formatação está exemplificada com valores de `id` e `valor` podem ser entendidos como a tabela da camada de interpretação demarcada em amarelo:

Figura 4 – Representação de um *Data Lakehouse*.



Fonte: Acervo do autor.

A maior vantagem deste tipo de arquitetura é que é muito fácil escrever diversos arquivos menores em um grande repositório compartilhado pois é possível paralelizar a escrita e leitura destes arquivos (mais detalhes na Seção 3.5). Além disso, os custos deste tipo de armazenamento são baratos se comparado ao custo de manter esses dados em disco acoplado à ferramenta OLAP, ao invés deste repositório separado. A depender da região e da latência desejada na leitura destes arquivos, os custos de armazenamento podem variar de US$ 0.0012 (um ponto dois milésimos de dólar) para as maiores latências até US$ 0.035 (três centavos e meio de dólar) para arquivos com recuperação imediata, valores esses cobrados para cada gigabyte por mês de armazenamento na provedora de serviços em nuvem Google Cloud Platform (GCP, 2023, preços de Out/2023). Outra vantagem é a inexistência de um limite prático para o volume de dados total em um armazenamento em formato de arquivos como o proposto pela arquitetura *Lakehouse* na nuvem, sendo possível armazenar exabytes de dados, se necessário e possuindo recurso financeiro para tal.

Algumas ferramentas que auxiliam na manipulação e implementação deste tipo de arquitetura são *Delta Lake* (usada no projeto), Apache Iceberg e Apache Hudi.

3.3 INTEGRAÇÃO DE DADOS E INTRODUÇÃO À ETL

Reiterando os quatro requisitos principais de uma arquitetura de dados no formato de DW (ou da variação *Data Lakehouse*), temos as camadas:

1. De ingestão ou replicação dos bancos de dados transacionais;
2. De armazenamento de dados, especificamente para *Data Lakehouse* no formato de arquivos em um repositório unificado;
3. De processamento e transformação de dados e;
4. De disponibilização e gerenciamento de acesso.

Estas camadas geram um conceito conhecido no mercado de engenharia de dados como ETL (INMON, 2005, p. 144–145, 420, 425), abreviação que engloba:

1. A extração (E - *Extract*) dos dados de suas fontes originárias (camada de ingestão), em que os dados crus são copiados ou extraídos dos sistemas transacionais e/ou quaisquer outros sistemas;
2. A transformação (T - *Transform*) dos dados de um estado inicial à um estado final (camada de processamento). Essa transformação se dá na forma de filtros, deduplicação, enriquecimento, limpeza, modelagem, validação, remoção de dados com potencial de identificação pessoal e até mesmo *compliance* com regulações de privacidade e;

3. A carga (L - *Load*) desse resultado em um sistema de persistência (camada de armazenamento) ou diretamente para ferramentas de inteligência de negócios ou qualquer outro consumidor de dados.

Quando se trata da construção de uma plataforma de dados, o ponto focal e palavra chave do problema é a **integração**. Em um armazém de dados, independente da variação a ser implementada, dados de diversos contextos devem ser integrados e relacionados entre si para gerar uma estrutura coesa e consistente capaz de responder às perguntas do negócio. Portanto, realizar as etapas de ETL acaba tornando-se grande parte dos desafios de engenharia de dados, tomando uma parcela significativa do tempo de desenvolvimento de um projeto desse escopo. Não só isso, mas também vale ressaltar que um mesmo conjunto de dados pode sofrer múltiplos processos de ETL, sendo processados ou carregado diversas vezes ao longo de sua existência.

A seguir, serão discutidas brevemente algumas técnicas, tecnologias e ferramentas modernas na integração de dados e execução dos processos de ETL, porém, detalhes avançados específicos como o processamento distribuído em si serão discutidos apenas na Seção 3.5.

3.3.1 Extração de bancos transacionais usando Change Data Capture (CDC)

No contexto de integração com bancos relacionais transacionais, existem diversas formas de realizar a replicação das operações de inserção, modificação e deleção que ocorrem internamente no banco para algum sistema externo, como uma camada de aterrissagem de dados crus (por vezes chamada *landing zone*) para posterior processamento. A forma mais simples e menos performática é a realização de consultas periódicas, por vezes selecionando todos os registros de uma tabela e gerando uma "foto" do banco de dados em uma data e substituindo a "foto" anterior, se houver. Essa é uma abordagem rudimentar e muito custosa para o banco transacional, e por esse motivo, outras técnicas são usadas na replicação.

Uma técnica que é suportada pela maior parte dos SGBDs é chamada de CDC, em que um sistema externo ao banco de dados é responsável por capturar as mudanças nos dados à medida que ocorrem, permitindo que essas alterações sejam replicadas para sistemas externos de maneira eficiente e em tempo real. A exemplo, um evento de inserção no banco de dados, além de inserir a linha propriamente dita também gera eventos de logs internos no SGBD, eventos esses que são consultados pelo software de captura de mudança de dados, repassados à uma fila de pré-processamento e depositados, em formato de arquivos, em algum repositório ou sistema de armazenamento. Se esse mesmo registro for posteriormente deletado, esse evento de deleção também é capturado e repassado para a camada de armazenamento. Na Figura 5 tem-se um arquivo de extensão `.json` com a representação de

eventos capturados em uma tabela de produtos.

No exemplo apresentado na Figura 5, o registro do produto de identificador com valor `4` sofreu uma criação às 08:03 da manhã do dia 28 de Outubro de 2023, inicializado com valor 39.99, depois sofreu uma atualização de valor que foi reduzido para 29.99 às 15:37 do dia 30 de Outubro de 2023, e posteriormente foi deletado da tabela do banco de dados no dia 31 de Outubro às 21:37, com datas e horas no horário oficial de Brasília.

Figura 5 – Exemplo de captura de logs de um SGBD.

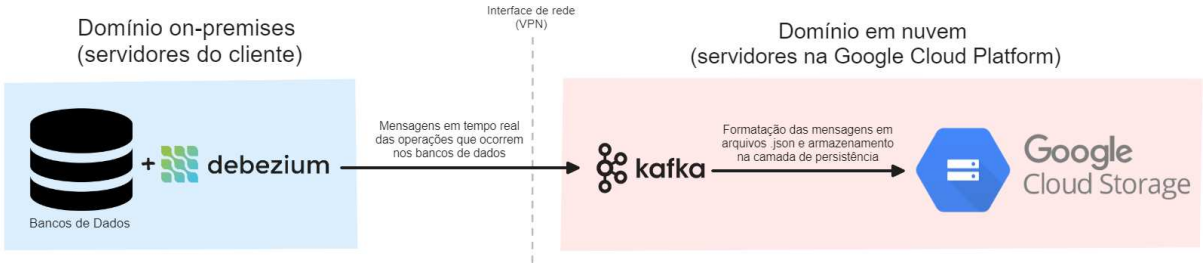

```
{
    "op": "c",
    "before": null,
    "after": {
        "product_id": "4",
        "price": "39.99"
    },
    "source": {···
    },
    "ts_ms": 1698491021323
}
{
    "op": "u",
    "before": {
        "product_id": "4",
        "price": "39.99"
    },
    "after": {
        "product_id": "4",
        "price": "29.99"
    },
    "source": {···
    },
    "ts_ms": 1698691021921
}
{
    "op": "d",
    "before": {
        "product_id": "4",
        "price": "29.99"
    },
    "before": null,
    "source": {···
    },
    "ts_ms": 1698799021537
}
```


Fonte: Acervo do autor, adaptado de dados reais do projeto desenvolvido neste PFC.

Este tipo de formato de captura é bastante eficiente e muito leve (em termos de carga de processamento) no banco de dados. Esse tipo de operação é definida em engenharia de software como um sistema orientado à eventos (CHANDY, 2009, p. 1040–1044).

A Figura 6 explicita um diagrama básico da arquitetura de ingestão discutida, consistindo na camada “E” da contração ETL, com a inclusão de textos explicativos e uma separação clara entre a infraestrutura que está em domínio do cliente da parte hospedada em nuvem.

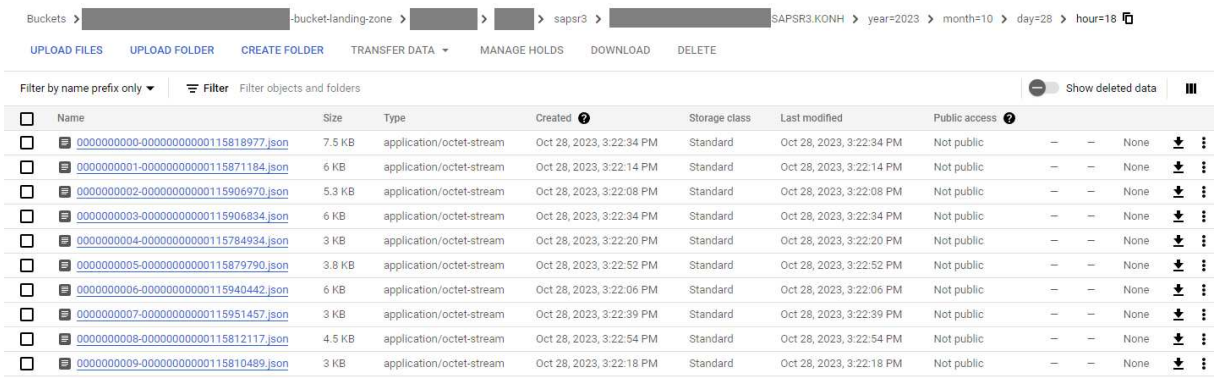
Figura 6 – Ilustração do processo de replicação de um banco de dados.



Fonte: Acervo do autor.

O resultado da correta configuração destas ferramentas está apresentado na Figura 7, com diversos arquivos de formato `.json` sendo disponibilizados e acessíveis em uma hierarquia de pastas similar ao sistema de arquivos de um sistema operacional como *Windows* ou *Linux*. Na parte superior da Figura 7 estão os nomes das pastas, e percebe-se que a hierarquia é particionada pelo ano, mês, dia e hora de captura dos registros, como explicitados pelos prefixos *year*, *month*, *day* e *hour*. Nesse caso, todos os registros contidos nos arquivos apresentados foram capturados no intervalo entre 18h e 19h (excludente) do dia 28 de Outubro de 2023. Alguns trechos da imagens foram censurados para que não seja possível a identificação do cliente.

Figura 7 – Resultado dos processos de replicação no ambiente do GCS.



Fonte: Acervo do autor.

3.3.2 Padrões de transformação de dados

Na etapa de transformação dos dados previamente extraídos geralmente ocorrem uma série de operações que objetivam preparar os dados para análises futuras e/ou condicioná-los para a tomada de decisão. Essas transformações são necessárias para garantir a qualidade, consistência e permitir uma usabilidade planejada dos dados.

Abaixo, alguns dos processos de transformação comuns aplicados: tipagem de colunas, padronização de textos, tratamento de valores nulos, reposições com estruturas condicionais, cálculo de medidas a partir de outros campos, deduplicação de registros e outros.

Estas transformações asseguram que os dados que fluem através dos *pipelines* são de alta qualidade e estão em um formato pronto para análise. Cada uma destas etapas deve ser cuidadosamente planejada e executada, pois transformações inadequadas podem levar a interpretações errôneas dos dados.

Após a aplicação dessas transformações, os dados podem ser novamente carregados para o repositório definido, costumeiramente em outras "pastas" que não a *landing zone*. Uma arquitetura bastante presente no mercado e a arquitetura medalhão, que separa as tabelas em camadas de agregação e processamento incrementais cujo nome é inspirado nas medalhas de olimpíadas, em que existem:

1. A camada de chegada dos dados crus dos sistemas transacionais, previamente mencionada e nomeada *landing zone* ou *landing layer*, onde há alta duplicidade de dados (pois armazena eventos sobre os registros, sendo que uma mesma linha pode sofrer múltiplos eventos) e nenhum nível de processamento agregado;

2. Uma camada chamada *bronze zone* ou *bronze zone*, em que ocorre um nível baixo de limpeza (apenas descarte de registros inválidos), tipagem de colunas e conversão de formatos de dados de `.json` para algo mais performático como `.parquet` (assunto da Seção 3.5) sobre os dados da *landing zone*;

3. Uma camada *silver zone* ou *silver layer*, montada a partir da *bronze zone*, em que há a deduplicação total dos dados, limpeza e tratamentos completos, que serve como uma réplica analítica OLAP otimizada para consulta do banco de dados transacional OLTP, com o adicional de estar completamente tratada e validada. É bastante usada para consultas *ad hoc* ou validações dos dados dos sistemas transacionais;

4. A última camada, *gold zone* ou *gold layer*, que utiliza-se dos registros da *silver zone* e os processa aplicando agrupamentos, agregações, aplicação de regras de negócio, filtragem e mais métodos de processamento como os descritos acimas para gerar tabelas com informações relevantes para o negócio. Aqui entram resultados como ordens de vendas abertas por período, por cliente, por ano-calendário, por região ou quaisquer outras agregações. Derivam-se dos dados da camada anterior as métricas e resultados que a equipe de coordenadores, gestores e executivos desejam observar, avaliar e tomar decisões sobre.

A transição dos registros para as camadas posteriores da arquitetura medalhão precisa ser feita de forma sistemática, consistente e eficiente, e portanto são emprega-

das ferramentas de gerenciamento de *pipelines* de dados para operacionalizar o ciclo de vida do DW.

3.3.3 Orquestração de rotinas de processamento

Entende-se como orquestração de rotinas de processamento como toda a instrumentação para gerenciamento de execução, agendamento, monitoramento, incrementalidade e coleta de *logs* de execução dos *pipelines*. Nesta etapa da arquitetura já existe um fluxo que replica os dados dos bancos transacionais e os coloca na primeira camada do DW, e faz-se necessário, portanto, executar a lógica de processamento que formata, limpa, filtra e agrega informação. Uma boa forma de modelar um orquestrador como esse é na forma de um grafo acíclico direcionado (do inglês DAG). É possível fazer um paralelo entre um DAG e uma máquina de estados finitos (Finite State Machine (FSM)), com a restrição de que não pode haver um ciclo fechado em um DAG, o que geraria um processo que não possui fim. Ambos possuem nós (ou estados) e setas direcionadas, com um nó inicial e um final, obrigatoriamente. Na Figura 8 há um exemplo de um DAG que poderia representar as tarefas de processamento desde sua chegada até a *gold zone*.

Figura 8 – Exemplo de DAG linear e sequencial.

Fonte: Acervo do autor.

Cada DAG é um conjunto de tarefas interdependentes, onde uma tarefa só é iniciada após a conclusão bem-sucedida das tarefas anteriores. Esta abordagem garante que os dados são movidos e transformados sequencialmente através das camadas do DW de forma organizada e sem redundâncias ou conflitos imediatos.

No mundo real, os DAGs podem assumir qualquer complexidade. É possível expandir o exemplo da Figura 8 em mais detalhes, quebrando elementos em ações mais granulares. Por exemplo, é possível descrever a tarefa `process_landing_to_bronze` em cada uma das etapas necessárias para transformar os dados da *landing zone* para *bronze zone*, porém, esta tarefa é delegada para algum *script* em alguma linguagem de programação. A tarefa do orquestrador é apenas gerenciar as dependências e conflitos que potenciais erros em uma dessas etapas acarretam, além de executá-las ou disparar sinais para que algum sistema externo as execute. Caso a etapa de processamento da *bronze zone* para a *silver zone* falhe, qualquer seja o motivo, então não faz sentido tentar executar a etapa de *silver* para *gold*, visto que são sequenciais e dependentes.

A Figura 9 demonstra um exemplo em que uma das tarefas não é executada com sucesso, interrompendo a execução do DAG:

Figura 9 – Exemplo de DAG com uma tarefa em estado de falha (em vermelho).

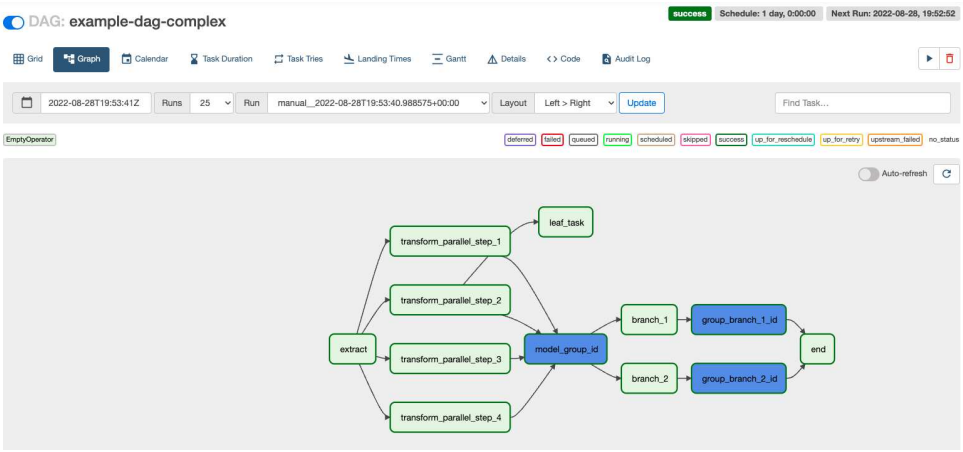
Fonte: Acervo do autor.

Além desse gerenciamento, faz parte do escopo do orquestrador o agendamento dessas rotinas, assegurando que as transformações de dados ocorram em momentos ideais sugeridos pelo engenheiro de dados (seja para minimizar carga no *cluster* de computadores executando o *pipeline*, seja por restrição da equipe de negócios, seja por otimização de custo ou qualquer outro motivo). Assim, orquestradores possuem internamente um componente de agendamento de tarefas similar ao *Windows Scheduler* ou ao *crontab*, definindo datas e horas de execução dos processos. A notação *crontab* é bastante famosa, consistindo em uma simples *string* de cinco valores separados por espaço. Cada um dos valores representa um intervalo na ordem: minuto, hora, dia do mês, mês e dia da semana. Alguns *crontabs* de exemplo são:

- `0 4 * * *`: todos os dias às 4 da manhã do horário local do computador;
- `30 9 * 1-6 *`: todos os dias às 9h30 da manhã, de primeiro de Janeiro até 30 de Junho, no horário local do computador e;
- `10 * * * 1-4`: a todo minuto 10 (de todas as horas), enquanto os dias da semana forem de segunda-feira até quinta-feira, no horário local do computador.

Na Figura 10, apresenta-se um exemplo de um DAG um pouco mais complexo na forma que é apresentado na interface de usuário do popular orquestrador Apache Airflow:

Figura 10 – Exemplo de DAG com tarefas-folha e dependências mais complexas.



Fonte: (ASTRONOMER, 2023).

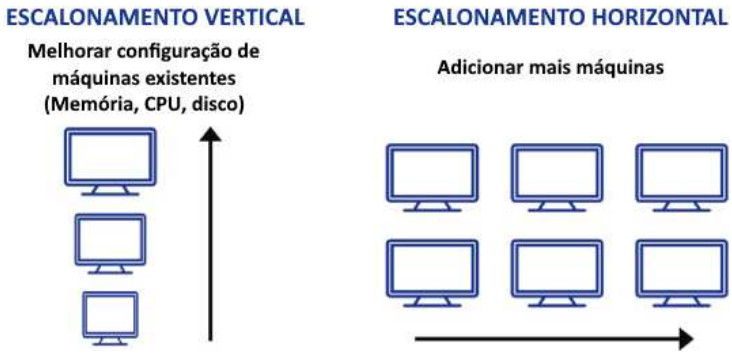
3.4 COMPUTAÇÃO EM NUVEM

Como introduzida previamente, a computação em nuvem permite que tarefas computacionalmente intensas, como é o caso do processamento de grandes volumes de dados ao mesmo tempo, sejam executadas em tempo hábil e de forma escalável. Caso uma tarefa seja única e singular, isto é, precise ser executada apenas uma vez, é irracional e ineficiente comprar um computador com muitos núcleos de processador e alta capacidade de memória para que após a execução da tarefa o mesmo se torne inútil para o negócio. A computação em nuvem permite que uma empresa “alugue” temporariamente ou permanentemente um ou muitos computadores e pague apenas pelo uso dos mesmos, e a partir do momento em que seu uso se faz desnecessário, são desalocados e não mais são faturados.

3.4.1 Escalabilidade em nuvem

Essa flexibilidade também permite um alto nível de escalabilidade horizontal. No contexto da computação, a escalabilidade vertical consiste na adição de mais recursos computacionais em um servidor ou computador existente, ou seja, adição de mais pentes de memória, mais espaço em disco, ou processadores com mais núcleos ou velocidade de *clock*. Essa abordagem, embora funcional, possui um limite técnico bem definido, além de se tornar relativamente cara conforme se aproxima das tecnologias de ponta. A escalabilidade horizontal, por sua vez, permite que sejam adicionados mais computadores comparativamente mais baratos, ao invés de melhorar as especificações de uma máquina já existente (SUBRAHMANYAM *et al.*, 2023, p. 2).

Figura 11 – Diferenciação entre escalabilidade vertical e horizontal.



Fonte: Adaptado de (SUBRAHMANYAM *et al.*, 2023, p. 4).

Este tipo de configuração é apenas um dos produtos oferecidos por empresas como a Google Cloud Platform (GCP) no âmbito de computação em nuvem. Existem diversos outros tipos de serviços como a configuração de redes privadas protegidas, hospedagem de bancos de dados, hospedagem de páginas na *web*, serviços de armazenamento de arquivos, modelos pré-treinados de AI generalistas, assistentes de conversão de texto-para-voz ou voz-para-texto de forma programática, redes de entrega de conteúdo e muitos outros serviços catalogados em (GOOGLE CLOUD PLATFORM, 2023c).

3.5 *BIG DATA*, COMPUTAÇÃO DISTRIBUÍDA E TECNOLOGIAS ASSOCIADAS

O termo *big data* é aberta e livremente usado coloquialmente na contextualização de altos volumes de dados. Porém, no que consiste um alto volume de dados? Em quanto tempo de processamento espera-se obter um resultado derivado de uma tabela de alto volume? A partir de qual tamanho um conjunto é considerado verdadeiramente *big data*? O próprio conceito de tamanho também está aberto à discussão: número de registros ou espaço que o conjunto ocupa em disco ou memória?

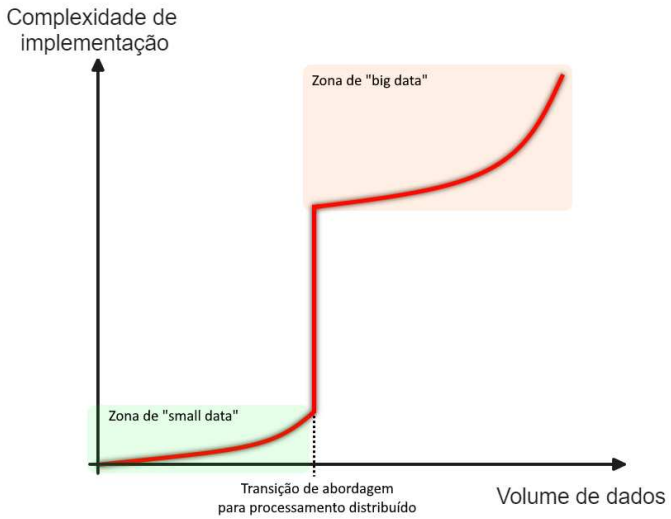
Embora aberta e com diversas interpretações, a definição de um conjunto de dados considerado verdadeiramente *big data* pode ser entendida, de acordo com referências acadêmicas, como sendo quando os métodos convencionais de análise e processamento não são cabíveis em tempo hábil na derivação de um resultado sobre o conjunto (HASHEM *et al.*, 2015, p. 1–2, 7–9). Essa definição, portanto, pressupõe que toda e qualquer base de dados de alto volume necessita, obrigatoriamente, ser operada por ferramentas ou técnicas distribuídas de processamento. Assim, se uma tabela pode ser carregada inteiramente em memória e ser processada em tempo hábil por um único computador, sem que haja a necessidade de repartição do conjunto, então essa tabela não consiste um conjunto grande o suficiente para ser considerado *big data*.

Em algumas literaturas, o termo *big data* vem associado à quatro “Vs” (HASHEM *et al.*, 2015, p. 3–4): volume, variedade, velocidade e valor. Essas propriedades caracterizam o significado e necessidade de se trabalhar com grandes conjuntos de dados, seja porque o volume é alto e requisita uma abordagem diferente no tratamento, seja porque é composto de grandes tabelas, muitas fontes e/ou possui grande potencial de escalabilidade. Ou então a variedade de diferentes informações coletadas é extensa, na forma de dados tabulares, vídeos, imagens, áudios. Às vezes a velocidade de processamento necessária é crucial para o caso de uso, ou ainda, por fim, o valor entregue pela informação é percebido como alto.

No projeto em questão, dois “Vs” se apresentam como críticos, o volume tratado e o valor agregado percebido possui alta relevância para o negócio.

Essa definição envolvendo escopos e métodos de processamento avançados é bastante apropriada pois, em termos de complexidade, o tratamento de conjuntos de dados pequenos e grandes não é linear, e possui um grande salto de complexidade (e evidentemente, custos), conforme representado na Figura 12:

Figura 12 – Representação da complexidade de implementação de *pipelines* de processamento para pequenos e grandes volumes de dados.



Fonte: Acervo do autor.

Dessa forma, assim que se cruza o limiar em que uma só máquina ou um só processo não é capaz de produzir os resultados esperados no tempo desejado, e não se faz possível ou viável a escalabilidade dessa máquina para que execute mais rapidamente, nem seja possível otimizar o processo além do que já está, então a solução plausível se torna a conversão da lógica de processamento para técnicas distribuídas. De acordo com Bill Chambers e Matei Zaharia, autores de uma forte referência no âmbito de processamento distribuído, o livro *Spark, The Definitive Guide*:

> Normalmente, quando se pensa em um "computador", imagina-se uma máquina única parada na sua mesa em casa ou no trabalho. Essa máquina funciona perfeitamente bem para assistir filmes ou trabalhar com planilhas. No entanto, como muitos usuários provavelmente já experimentaram em algum momento, existem algumas coisas que um computador não é potente o suficiente para realizar. Uma área particularmente desafiadora é o processamento de dados. Máquinas individuais não têm poder e recursos suficientes para realizar cálculos em enormes quantidades de informação (ou fazê-lo em tempo hábil). Um *cluster*, ou grupo de computadores combina os recursos de várias máquinas, dando a capacidade de usar todos os recursos cumulativos como se fossem um único computador. (CHAMBERS; ZAHARIA, M., 2018, p. 20, tradução livre do autor).

Existem ferramentas que abstraem muitas das complexidades que esse tipo de abordagem traz, sendo a mais popular chamada Apache Spark, ou apenas Spark. Essa ferramenta funciona como um motor analítico para processamento de dados em larga escala que é hospedado em um conjunto de computadores, e que disponibiliza uma interface de comunicação através de uma Application Programming Interface (API) para que algumas linguagens de programação como Scala e Python possam disparar comandos ao motor de processamento de forma rápida, essencialmente criando planos de execução que a motorização paraleliza no *cluster* de computadores.

Internamente, a implementação do Spark pode ser entendida como um sistema mestre-escravo na qual, para cada tarefa (terminologia oficial é *application*) deve existir uma instância controladora (chamada *driver*) e um ou mais executores (chamados *executors* ou *workers*). Assim, para realizar uma operação distribuída com Spark, deve-se submeter uma *application* com algumas configurações, entre elas as especificações desejadas de memória e núcleos de CPU para o *driver*, assim como a quantidade de *executors* e suas especificações. Além disso, existe uma camada supervisória por cima de tudo chamada de *cluster manager*, responsável por alocar corretamente os recursos do *cluster* para eventuais *drivers* e *executors* (CHAMBERS; ZAHARIA, M., 2018, p. 20–22).

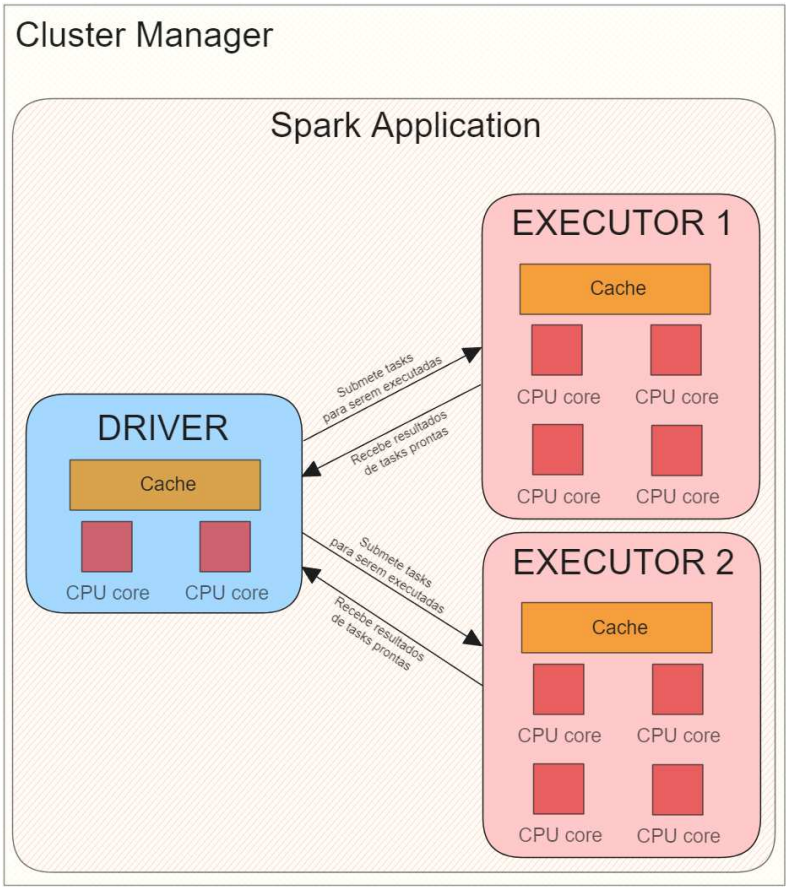
Cada *executor*, por sua vez, é responsável por receber uma parte do conjunto de dados a ser processado (chamado *dataframe*), realizar as operações necessárias (completando unidades de trabalho chamadas *tasks*) e devolver o resultado para o *driver*. Se cada *executor* for capaz de realizar a operação de forma independente, então os resultados parciais de cada *executor* são unidos e agregados rapidamente no *driver* e retornados ao usuário, seja na forma de salvamento em disco ou em algum repositório, ou como retorno de consulta (assim como são feitas operações em um SGBD comum) (CHAMBERS; ZAHARIA, M., 2018, p. 24–25). Isso permite, por exemplo, a persistência em disco de múltiplos arquivos ao mesmo tempo, cada um com uma fração dos dados completos, acelerando e efetivamente paralelizando o processamento do conjunto. Internamente, o Spark realiza a maior parte da operações, se possível, em memória, pois embora volátil, é a forma mais fácil de armazenar e

consultar dados em um sistema computacional. Todavia, até mesmo com alguns TB de memória, não seria possível armazenar todo o conjunto em memória ao mesmo tempo, e portanto, existe forma de persistir parte de um conjunto de dados em disco temporariamente. Essa operação é chamada de *spill* e prejudica consideravelmente o desempenho das rotinas caso ocorra, e assim se faz importante na otimização das rotinas de ETL o conhecimento intrínseco dos conjuntos de dados trabalhados e no dimensionamento das Spark *applications*.

Se a operação envolve múltiplos conjuntos de dados, o que é bastante comum especialmente em operações de `JOIN` ou ordenações, então pode ocorrer de um *executor* necessitar acessar dados que estão localizados em um outro *executor*, implicando em tráfego de rede entre *executors* e podendo ser necessária reordenação, filtragem ou outros métodos de efetivar a co-localização de dados relevantes em um mesmo *executor* para que a unidade de trabalho possa ser concluída. Essa operação de localizar dados relacionados entre si em um mesmo executor também é ponto focal da otimização a ser feita em uma *application* do Spark, pois uma má otimização pode fazer com que uma aplicação demore centenas de vezes a mais que o necessário.

Na Figura 13 está representada a arquitetura básica de uma implantação do Apache Spark em um *cluster*, que ilustra uma única *application*, e portanto apenas um *driver* operando com dois *executors*. Cada *executor* possui núcleos de CPU, uma camada de cache, memória e opcionalmente, disco (não representados). As tarefas são executadas pelos processadores e os resultados parciais são enviados ao *driver*. Caso um *executor* precise se comunicar com outro para compartilhar dados, esse fluxo se dá por rede interna através do *driver* e coordenado pelo *cluster manager*, que pode ser uma ferramenta de código aberto como Kubernetes ou algo gerenciado e providenciado por uma nuvem, como o GKE.

Figura 13 – Arquitetura de um *cluster* executando uma *application* do Spark.

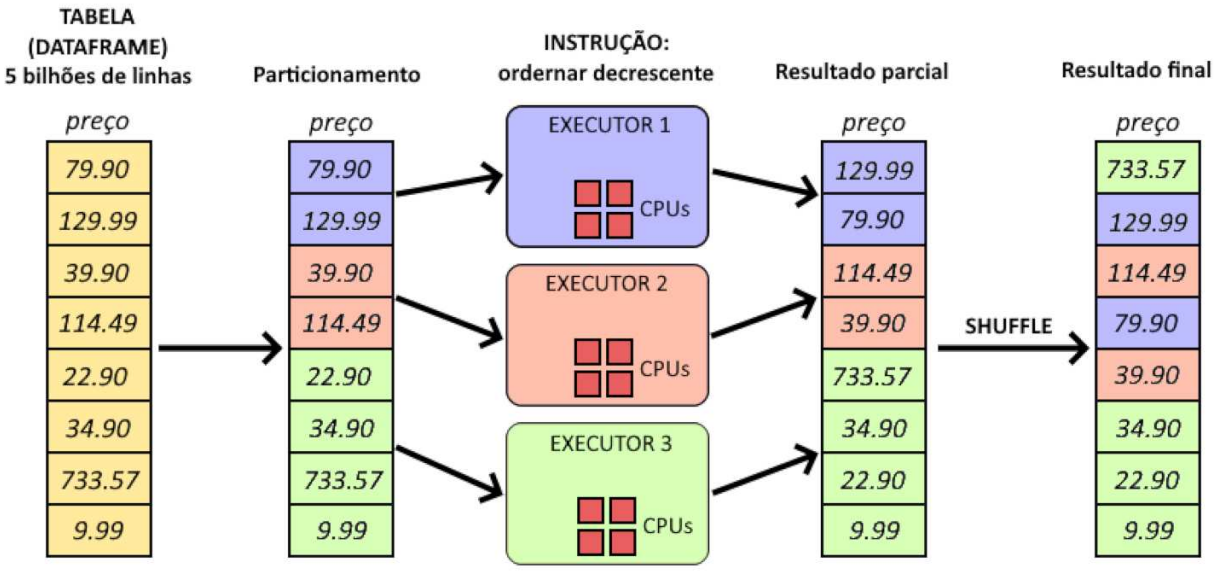


Fonte: Acervo do autor.

A forma com que o Spark distribui tarefas para os *executors* é controlada pelo engenheiro de dados, que define, entre muitas outras coisas, a forma de "quebrar" (terminologia oficial é particionar) uma ou várias tabelas grandes (por grande subentende-se um conjunto de dados que não cabe em memória em apenas um computador, ou seja, um conjunto *big data*) em fatias menores ou partições. Assim, cada partição pode ser operada individualmente por cada *executor*. A exceção para esse caso, como já mencionado, é quando um *executor* precisa de dados contidos em outro *executor*, o que introduz uma complexidade de compartilhamento chamada *shuffle* (CHAMBERS; ZAHARIA, M., 2018, p. 27–28). Essa necessidade está bem representada na Figura 14, que explicita uma operação de ordenação distribuída, na qual inicia-se com uma tabela exemplo com alguns valores que é particionada em fatias menores e distribuída para cada executor, instruídos para ordenar o conjunto de forma decrescente. O resultado individual de cada *executor* está com os preços ordenados conforme esperado, mas o conjunto, como um todo, ainda permanece desordenado. Nesse caso, a motorização do Spark dispara uma nova etapa de *shuffle* que compartilha os resultados parciais de cada *executor* entre si, realizando uma última ordenação final entre os resultados

parciais para obter o resultado final.

Figura 14 – Operação de ordenação distribuída.



Fonte: Acervo do autor.

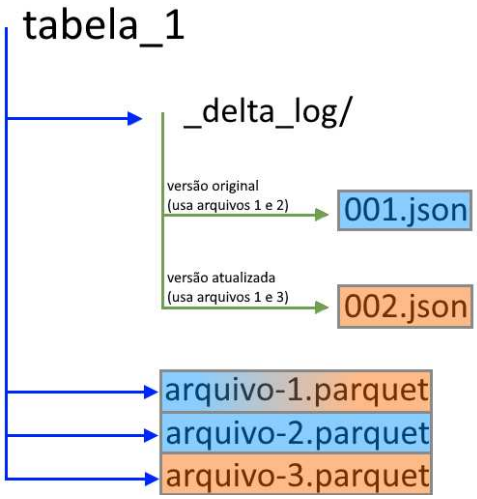
3.5.1 Delta Lake

Além disso, em conjunto com a motorização do Spark no paralelismo, deve-se atentar para o formato de arquivos usados na hora de leitura e escrita de tabelas segundo a arquitetura *Lakehouse*. Conforme mencionado anteriormente, é possível realizar a leitura de *n* diferentes arquivos como os de extensão `.json` e unir todos os dados contidos nesses arquivos em uma única tabela. Todavia, no âmbito de computação distribuída e *big data*, a quantidade de arquivos e seu tamanho se tornam relevantes na otimização de processos que leem ou escrevem para tais arquivos. Nesse contexto, um formato muito utilizado é o formato `.parquet`, projeto descrito e mantido pela fundação Apache, também. Esse formato é representado em disco de forma extremamente eficiente para esse tipo de aplicação pois é orientado à colunas (assim como sistemas OLAP), e possui um forte suporte para compressão e descompressão e tipagem de dados.

Esses arquivos `.parquet` são associados, intrinsecamente, com uma outra camada que sobrepõe a camada de persistência, implementada por uma ferramenta chamada Delta Lake, que armazena, versiona e controla a escrita e leitura de diversos arquivos `.parquet` de forma eficiente. Tomando-se um caso em que uma tabela é composta por dois arquivos `.parquet`, cada qual armazenando cinco registros, então tem-se um total de dez registros (*id* = [1, 2, ... 10]). Agora imaginando a situação em que se deseja modificar um dos registros, por exemplo o com *id* = 3, mas não

se deseja perder a versão anterior da tabela, então, cria-se um terceiro arquivo com os conteúdos originais do arquivo que continha o registro com *id* = 3, porém com a modificação em questão. Depois, versiona-se a tabela usando um arquivo de apoio de extensão `.json` que referencia quais arquivos `.parquet` de dados aquela versão necessita. Na Figura 15, uma representação na qual existem duas versões para uma mesma tabela: uma original e uma modificada, com reutilização do `arquivo-1.parquet` pois seu conteúdo é valido em ambas as versões (DELTA LAKE FOUNDATION, 2023).

Figura 15 – Representação de versionamento de tabelas com Delta Lake.



Fonte: Acervo do autor.

3.6 ARQUITETURAS DISTRIBUÍDAS E CENTRALIZADAS

A escolha entre uma arquitetura de processamento distribuído e uma arquitetura de processamento centralizado de dados é ditada, principalmente, pelo volume de dados a serem processados e pela necessidade de escalabilidade imposta por variações ou tamanho total das bases de dados consideradas. Enquanto a arquitetura centralizada opera em torno de um único sistema ou servidor que lida com todas as operações de dados, o processamento distribuído, como o nome sugere, divide a carga de trabalho entre vários sistemas interconectados via rede em um *cluster*, permitindo o processamento paralelo de grandes volumes de dados utilizando alguma tecnologia de paralelização (KSHEMKALYANI; SINGHAL, 2008, p. 1–5) como as apresentadas na Seção 3.5.

Em contraste, arquiteturas de processamento centralizado têm seu lugar em cenários onde o volume de dados é gerenciável e não há expectativa de crescimento rápido. Nestes sistemas, todos os dados são processados e armazenados em um servidor central, simplificando o gerenciamento e manutenção ao custo de algumas limitações significativas de escalabilidade e desempenho (MOFADDEL; TAVANGARIAN,

1997, p. 3). Existe um limite físico e tecnológico no quão grande e rápido um mesmo computador pode ser, seja por limitação de processador, memória, disco, gráficos ou outros recursos. À medida que o volume de dados a ser considerado cresce, o sistema sofrerá com gargalos de desempenho, dado que a capacidade de processamento e armazenamento é limitada.

Por outro lado, o processamento distribuído, empregado em arquiteturas modernas de *big data*, é projetado para superar essas limitações utilizando ferramental que permite que os dados sejam processados em paralelo, utilizando todo o poder de diversas máquinas no *cluster*, cada uma trabalhando em uma parte do problema. Isso não só acelera o processamento devido à execução simultânea de tarefas, mas também oferece uma escalabilidade horizontal quase ilimitada, visto que, por mais que um mesmo computador possui um limite claro e definido pelas tecnologias de fabricação de placas-mãe, processadores, pentes de memória, etc., ainda será possível adquirir um outro computador com as mesmas configurações. Dessa forma, à medida que a demanda por recursos cresce, novas máquinas podem ser adicionadas ao *cluster* (MOFADDEL; TAVANGARIAN, 1997, p. 4). Ainda assim, embora as arquiteturas distribuídas ofereçam vantagens significativas em termos de desempenho e escalabilidade, elas também trazem um nível de complexidade mais alto e considerações de custo. A infraestrutura necessária para suportar um sistema distribuído é mais complexa, requerendo não apenas mais *hardware*, mas também *software* para gerenciar a distribuição e coordenação das tarefas entre os nós (KSHEMKALYANI; SINGHAL, 2008, p. 39–42).

Por isso é de suma importância conhecer intrinsecamente os volumes, bases e conjuntos de dados a serem tratados, fazendo uma análise minuciosa das necessidades e requisitos de projeto. Para empresas que lidam com volumes de dados que estão na escala de *terabytes* ou *petabytes*, que precisam de análises em tempo real ou que preveem um crescimento significativo de dados, o processamento distribuído é quase sempre a escolha certa, visto que é inviável tentar processar tabelas que sequer cabem na memória de um computador sem um sistema de *software* capaz de lidar com *spill* para disco e que seja capaz de distribuir tarefas para serem executadas em múltiplos núcleos de processamento. Já pequenas e médias empresas com necessidades de dados moderadas podem operar eficientemente com arquiteturas centralizadas, pelo menos até que seu crescimento de dados justifique a transição para um modelo distribuído.

4 METODOLOGIAS E ARQUITETURA

Neste capítulo estão apresentadas algumas das metodologias de projeto usadas no desenvolvimento, a arquitetura geral do sistema envolvendo as fontes de dados relevantes para o projeto BI Carteiras, as rotinas de ETL, orquestração e disponibilização das tabelas processadas ao usuário final. Alguns diagramas e especificações em Unified Modelling Language (UML) também descrevem melhor a arquitetura proposta.

4.1 METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE PARA ÁREA DE DADOS

O desenvolvimento de soluções de software na área de dados requer metodologias e ferramentas que otimizem o processo de criação e manutenção de sistemas complexos, pois existe alta incidência de regras de negócio, adaptações, preferências e requisitos que só são descobertos com entrevistas com o cliente. Como em qualquer projeto, a implementação de práticas consistentes e ágeis é essencial para garantir a qualidade, a eficiência e a entrega contínua. Um projeto na área de dados está muito sujeito à mudanças de escopo, e portanto a gerência do projeto precisa ser criteriosa.

4.1.1 Versionamento de código com GitLab

Para realizar o controle de versão para os códigos das rotinas de ETL e gerenciamento de infraestrutura, o autor se utilizou do repositório GitLab existente na empresa cliente. Essa ferramenta disponibiliza outro recurso útil na forma da integração e entrega contínua (*continuous integration and continuous development*, ou CI/CD). As esteiras de CI/CD já estavam implementadas no repositório quando o projeto se iniciou, assim, quando novas alterações nos *scripts* eram feitas e confirmadas (operação de *merge* em Git), a atualização da base de códigos nas máquinas do *cluster* de computadores é automática.

4.1.2 Metodologias Ágeis e divisão de trabalho por *sprints*

As metodologias ágeis nos permitem responder de maneira flexível às mudanças de requisitos e prioridades constantes em projetos de dados, detalhando regras, ritos e padrões a serem seguidos para a entrega eficiente dos requisitos de projeto. O trabalho realizado neste projeto teve ciclos de entrega (*sprints*) quinzenais. Em cada *sprint*, planejado sempre às segundas-feiras, um dono de produto *product owner* da empresa cliente se reúne com o engenheiro de dados responsável (o autor), e são identificados os pontos em atraso, demandas pendentes e prioridades para a *sprint* corrente. Dessa forma, existe melhoria contínua nas entregas, como o processamento de novas tabelas, correção de problemas críticos, adição de novos campos nas tabelas

fato e dimensão, melhorias de desempenho na execução dos *pipelines*, destravamentos de acessos e outros.

Um ponto controverso que algumas metodologias pregam é o rito de realizar uma reunião diária com as partes interessadas de um projeto para alinhamento rápido, visto que coloca uma certa pressão nos desenvolvedores e muitas vezes tomam mais tempo do que deveriam. Ainda assim, ao longo do desenvolvimento do projeto, adotaram-se as *dailies* como forma de sincronização da equipe sobre pontos de travamento, tarefas para o dia e alinhamento de expectativas.

4.2 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Para a implementação bem-sucedida do projeto, os seguintes requisitos funcionais e não funcionais são considerados, de acordo com Tabela 5 e Tabela 6:

Tabela 5 – Relação de requisitos funcionais. O requisito F1 está fora do escopo desde trabalho pois é alvo de outro projeto.

ID	**REQUISITO**	**CATEGORIA**
F1	O sistema deve extrair dados de forma contínua e incremental via CDC das fontes	Operação
F2	Dados devem ser processados utilizando alguma ferramenta que permita realizar operações nos volumes tratados em tempo hábil	Operação
F3	A arquitetura medalhão deve ter suas camadas armazenadas em nuvem no GCS	Disponibilização
F4	A camada *bronze* deve fornecer um histórico de eventos para cada registro	Disponibilização
F5	A camada *silver* deve prover uma réplica otimizada para consulta do SGBD relacional	Disponibilização
F6	A camada *gold* gerar tabelas de fatos e dimensões para análises de inteligência de negócios ou consumo geral por quaisquer outras aplicações ou pessoas	Disponibilização
F7	O sistema deve utilizar-se de Delta Lake para controle transacional e gerenciamento de versões	Operação
F8	O sistema deve permitir atualizações e manutenção sem interrupção dos serviços	Manutenção
F9	O projeto deve entregar, ao todo, todas as tabelas do modelo estrela proposto na avaliação de escopo, contendo uma tabela fato de ordens de venda e sete dimensões para caracterizar produtos, clientes, representantes, filiais, regionais, remessas e faturas	Operação

Fonte: Acervo do autor.

Tabela 6 – Relação de requisitos não funcionais.

ID	REQUISITO	CATEGORIA
NF1	O sistema deve gerar as tabelas de interesse com intervalo menor que 24 horas em comparação ao sistema transacional	Desempenho
NF2	Deve responder à aumentos e diminuição de demandas e cargas de forma eficiente	Escalabilidade
NF3	Alta adesão às práticas de segurança para proteção e conformidade de dados, incluindo gerenciamento de acesso	Segurança
NF4	Deve possuir forma prática de realizar monitoramento das execuções de *pipelines*, com tempos e volumes processados	Monitoramento
NF5	Otimizado para a redução de custos de infraestrutura	Eficiência

Fonte: Acervo do autor.

4.3 ESPECIFICAÇÕES DE PROJETO

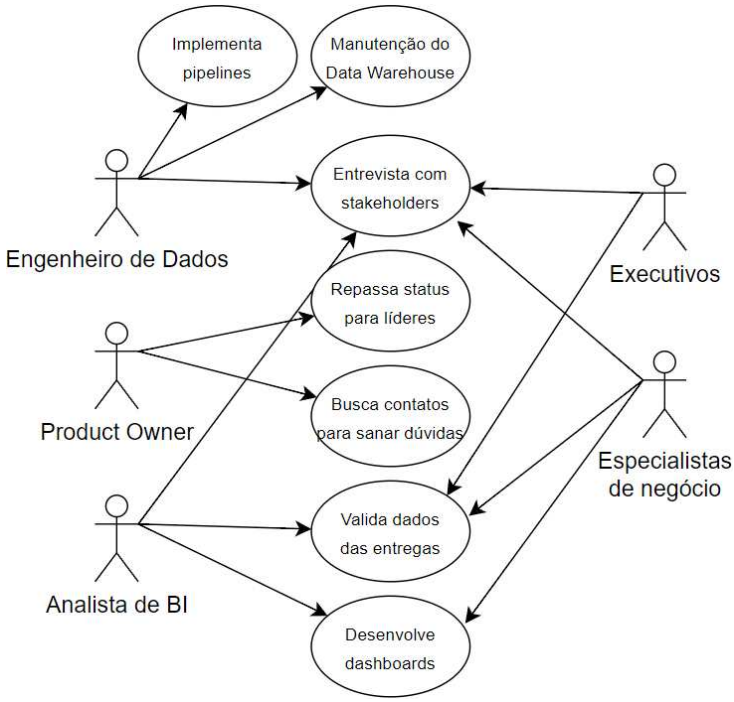
Nesta seção serão abordados alguns diagramas e representações em UML ou outras formas de especificar formalmente a arquitetura de sistema a ser implementado.

4.3.1 Atores envolvidos e diagrama de casos de uso

Dentre os atores mais importantes na execução do projeto estão a equipe diretamente ligada ao seu desenvolvimento, composta por um engenheiro de dados, um *product owner* e um analista de BI, e atores externos ao time, representados pelos executivos interessados e a equipe de especialistas de negócio que farão uso dos dados na geração de relatórios e análises.

A Figura 16 ilustra o diagrama de casos de uso, destacando as interações entre diferentes papéis dentro de um projeto de dados e as responsabilidades associadas a cada um deles. O engenheiro de dados é central para a implementação e manutenção de pipelines e do Data Warehouse e realiza entrevistas com os *stakeholders*. Executivos e especialistas de negócio estão envolvidos na definição de requisitos e objetivos estratégicos, colaborando com o engenheiro de dados por meio das entrevistas e fornecendo *feedback*. O *product owner* atua como um elo entre a equipe técnica e os stakeholders do negócio, responsável por comunicar o status do projeto, esclarecer dúvidas e buscar contatos importantes. Finalmente, o analista de BI valida os dados entregues e desenvolve, em paralelo, os painéis e visualizações para traduzir esses dados em insights acionáveis juntamente com os especialistas.

Figura 16 – Diagrama de casos de uso.



Fonte: Acervo do autor.

4.3.2 Serviços envolvidos e diagrama de *deployment*

Para efetivar a implantação da arquitetura de dados, foi necessário o uso de alguns serviços e ferramentas disponibilizados na GCP como o GKE na forma de *cluster* e GCS como camada de persistência. Enquanto isso, localmente em um servidor hospedado em computadores do próprio cliente existe um SGBD Oracle sendo executado, onde também fora instalado a ferramenta de captura em formato CDC Debezium. A escolha da provedora de nuvem GCP foi concretizada antes mesmo da concepção deste projeto, com uma parceria entre a empresa cliente do projeto e a GCP para outros projetos passados. Com isso, já havia uma infraestrutura implementada com essa provedora e foram feitas expansões em cima do contrato de prestação de serviços já firmado, descartando, portanto, o uso de outros providores como a Amazon Web Services ou Microsoft Azure.

De forma geral, o processo que executa Debezium está diretamente consultando os *logs* do banco de dados Oracle nas dependências do cliente, isto é, um servidor local cuja localização não pode ser divulgada. Essas informações de logs são enviadas para uma instância de Apache Kafka hospedada no *cluster* de computadores na nuvem. Esse envio é feito por protocolo MQ Telemetry Transport (MQTT), padrão amplamente difundido para envio e recebimento de mensagens em tempo real. O Apache Kafka, por sua vez, disponibiliza uma fila de eventos de *logs* que são imedia-

tamente transformados em requisições HTTP para as APIs oficiais do GCS com uma formatação específica. Cada projeto na GCP disponibiliza APIs para interação com seus múltiplos serviços, incluindo criação de objetos `.json` utilizando o verbo HTTP POST com parâmetros bem formatados, o que é feito nativamente dentro do Apache Kafka.

Sendo executados dentro do *cluster* ainda tem-se uma única instância da ferramenta Apache Airflow, que internamente dispara tarefas no formato de comandos para o próprio *cluster* para que sejam criadas instâncias e provisionados recursos para as Spark *applications*. Cada *application*, por si, é capaz de ler e escrever dados diretamente de/para o GCS utilizando a mesma API usada pelo Apache Kafka.

Por fim, um último serviço nativo da Google, o Google Cloud BigQuery, ou apenas BigQuery (BQ), se conecta diretamente e transparentemente aos arquivos guardados no GCS. O BigQuery é a camada de consulta que será a forma com que todos os analistas poderão consultar as tabelas geradas pelos *pipelines* com linguagem SQL. A partir dele também é possível realizar exportações de dados para planilhas ou outros sistemas, além de fazer toda a análise exploratória, computação e processamento adicionais (se necessário) e consultas sob demanda de forma rápida e eficiente.

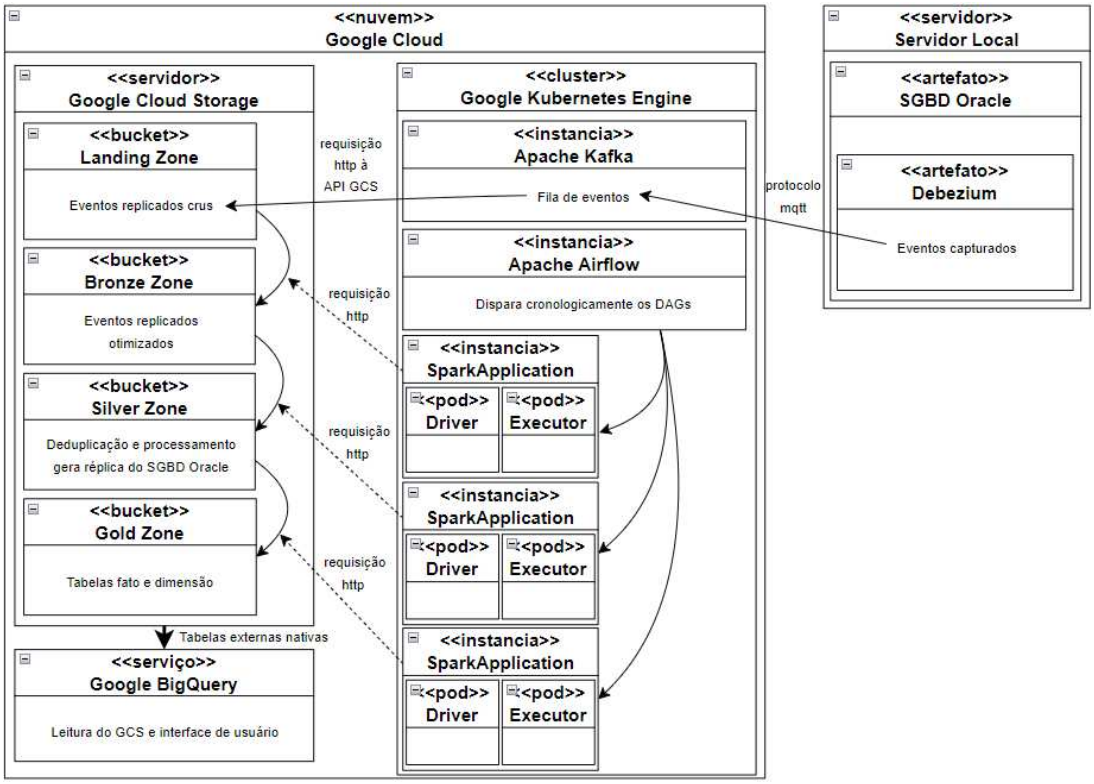
Esse produto funciona, por si só, como um DW, isto é, possui, internamente, uma motorização distribuída, máquinas virtuais escaláveis horizontalmente, armazenamento embutido e muitas APIs de interação (GOOGLE CLOUD PLATFORM, 2023a). Todavia, é um produto extremamente caro e gera um alto índice de *lock-in*: se o cliente compromete-se à usar todas as funcionalidades do BQ, não existe outra ferramenta no mundo que replica seu *modus operandi*, infraestrutura e APIs. Assim, se em algum momento a GCP desejar triplicar o custo de consultas nesse serviço, a empresa não possui alternativa a não ser acatar. O BQ, portanto, é usado como uma camada *web* na nuvem (ou seja, podendo ser acessado de qualquer lugar, desde que o usuário se autentique) que permite a consulta livre aos dados armazenados externamente ao próprio BQ, e sim no GCS. Conforme mencionado, o BQ também disponibiliza APIs nativas da para interação com as suas *features*, permitindo realização de consultas de forma programática (para alimentar sistemas ou modelos de AI e ML) ou consultas *ad hoc* por quem desejar, efetivamente resolvendo os problemas apresentados pelo cliente no início do projeto, fazendo dos dados um produto *self-service* para todo o corpo de analistas, automaticamente atualizado, limpo e bem caracterizado.

Na GCP, no momento de criação de qualquer recurso, seja uma pasta (chamadas *buckets*) no GCS para alguma das camadas da arquitetura medalhão ou a criação do *cluster* propriamente dito, o usuário é capaz de informar em qual região deseja instanciar o recurso, mas dentro de uma região, por questões de segurança, a localização exata do *data center* não é informada pela provedora de nuvem. Assim, devido à empresa cliente ser nativamente brasileira, mesmo com os custos dos serviços em nuvem

serem um pouco mais caros quando hospedados no Brasil (questão de custo de hardware e energia elétrica), a região definida na realização do *deployment* de todos os serviços é *southamerica-east1*, com *data center* localizado em algum lugar no estado de São Paulo. É de extrema importância que, uma vez decidida a localização de um recurso, todos os outros estejam, quando possível, na mesma localização para reduzir custos de tráfego de rede entre os mesmos e também a latência de comunicação entre eles.

Dessa forma, o diagrama de *deployment* da arquitetura é apresentado na Figura 17:

Figura 17 – Diagrama de *deployment*.



Fonte: Acervo do autor.

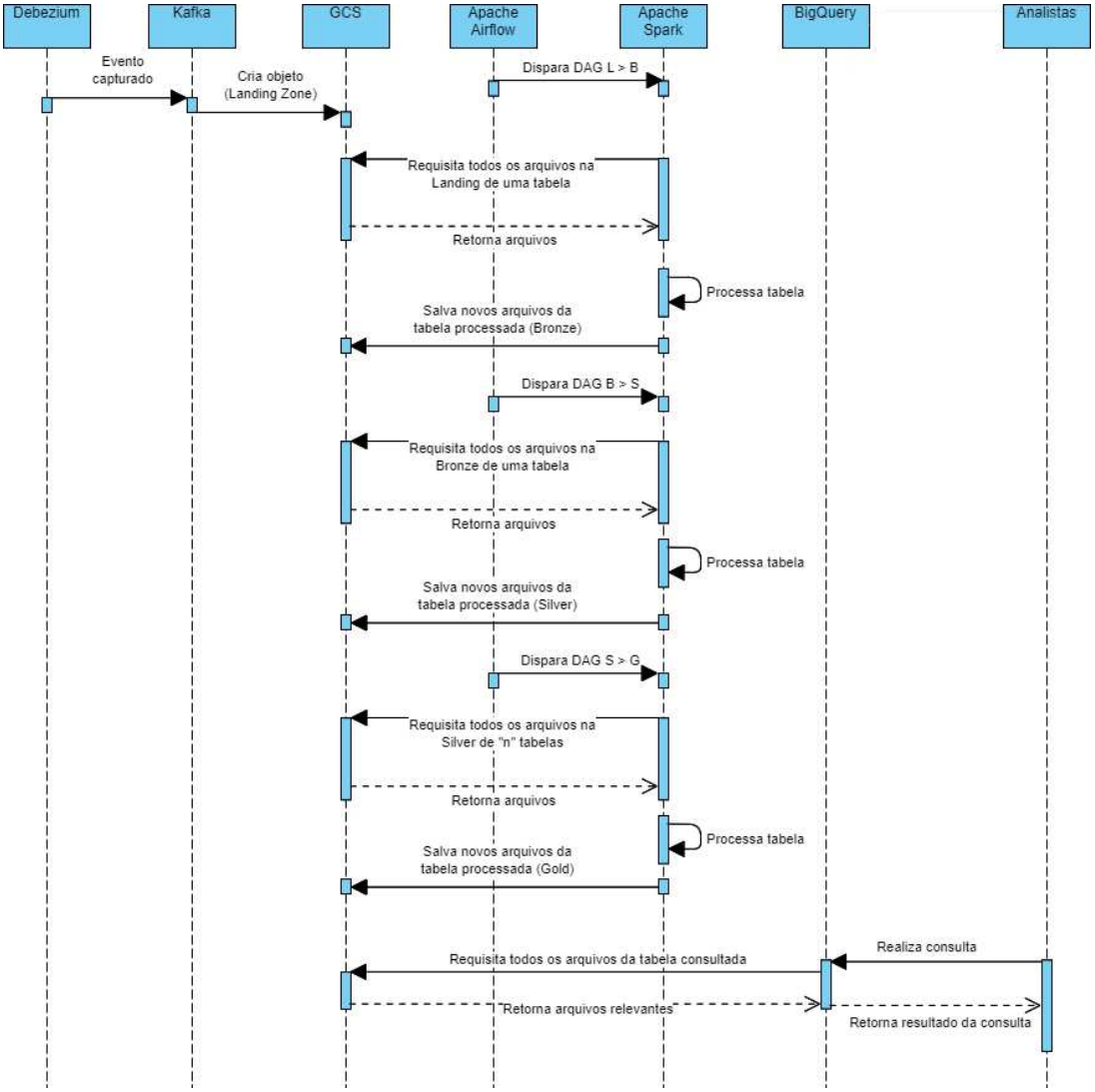
4.3.3 Integração dos serviços utilizados e diagrama de sequência

Um diagrama de sequência é usado para descrever como os componentes do sistema interagem entre si e trocam mensagens/requisições ao longo do tempo para realizarem as tarefas esperadas. No caso da arquitetura proposta, percebe-se que idealmente, para cada registro capturado via CDC pelo Debezium, transmitido para o Kafka e replicado como objeto na *landing zone*, idealmente se dispararia o DAG responsável pelo processamento desse registro da *landing* para *bronze* (L > B), *bronze* para *silver* (B > S) e eventualmente *silver* para *gold* (S > G), realizando a

computação necessária e salvando os resultados diretamente no GCS. Todavia isso requereria que a infraestrutura estivesse ligada 24 horas por dia processando eventos para todas as tabelas, o que é inviável dada a própria quantidade de tabelas a serem processadas (inicialmente foram 53 tabelas do ERP SAP), e portanto, dados alguns requisitos entendidos com entrevistas aos usuários (mais detalhes no Capítulo 5), o processamento é feito em lotes, executados a cada 24 horas no período da madrugada no horário de Brasília, que processam ao mesmo tempo todos os eventos ocorridos no último dia que passou.

De acordo com o diagrama de sequência dado na Figura 18, portanto, pode-se perceber que a maior interação acontece entre três elementos: o Airflow disparando as DAGs que executam os processos no Spark, e o Spark requisitando ou salvando arquivos no GCS, seguindo a arquitetura de um *Data Lakehouse* mencionado na fundamentação teórica, isto é, com cada tabela sendo composta por múltiplos arquivos. Por mais que não esteja representado no diagrama, a interação entre Debezium e Kafka é bastante recorrente, dependendo apenas da execução de eventos no SGBD. Enquanto isso, um usuário (analista) pode fazer consultas à qualquer momento na interface do BigQuery, que, novamente, integra diretamente com os arquivos guardados no GCS.

Figura 18 – Diagrama de sequência.



Fonte: Acervo do autor.

4.3.4 Fluxo de informação pelas camadas da arquitetura medalhão

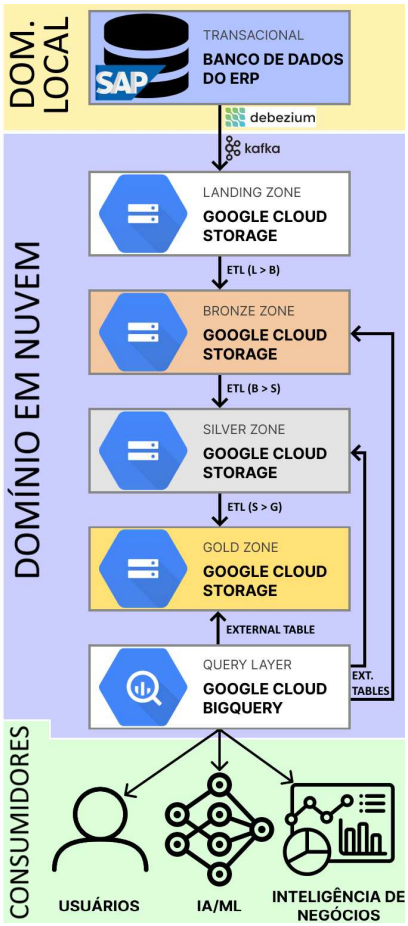
Uma forma de grosseiramente simplificar as especificações dadas nesta seção é na forma de um diagrama de fluxo de dados, que indica como os registros capturados por eventos do sistema transacional do ERP SAP vão transitando pelas camadas da arquitetura medalhão. O ponto de partida é o banco de dados transacional, esse banco de dados central é um banco Oracle, e mantém todos dados operacionais da empresa cliente do projeto. Conforme previamente mencionado, a captura de eventos desse sistema é realizada através de uma abordagem usando a replicação de eventos via CDC feita a partir da combinação de ferramentas Debezium, para a captura propriamente dita, e Kafka, na replicação das mensagens capturadas em arquivos de dados na *landing zone* em tempo real.

Verifica-se que uma vez capturados e replicados na *landing zone*, os eventos sofrem os processos de transformação até a camada *gold*. Cada uma destas etapas envolvem um certo nível de limpeza, deduplicação, filtragem e agrupamento dos dados, como tipagem de colunas e conversão de formatos de arquivos. O objetivo final, portanto, é ter na *gold zone* uma camada refinada de dados, com a aplicação de regras de negócio e conversão das muitas tabelas transacionais em poucas tabelas fato e dimensão, que oferecem métricas e valor aos executivos e tomadores de decisão e não apenas contém registros "jogados".

Todas as camadas que sofrem algum tipo de processamento, isto é, *bronze* até *gold* são consultadas diretamente pelo BigQuery usando sua funcionalidade de **external table**, desenvolvida especificamente pela Google como forma de integrar seus serviços e permitindo a utilização de linguagem SQL para consulta aos múltiplos arquivos `.parquet` tabulares que compõem uma tabela, arquivos esses salvos no GCS e assim implementando a arquitetura de um *Data Lakehouse* no DW. Internamente, o BigQuery faz a leitura de todos os vários pequenos arquivos salvos no GCS e os concatena em tempo de execução, para cada tabela definida pelo engenheiro de dados.

O diagrama de fluxo está apresentado na Figura 19:

Figura 19 – Diagrama de fluxo de dados no projeto.



Fonte: Acervo do autor.

4.4 CONFIGURAÇÕES DA ARQUITETURA GERAL DO SISTEMA

Congregando a teoria apresentada no Capítulo 3 envolvendo a fundamentação teórica necessária para pleno entendimento das atividades realizadas no projeto, a arquitetura geral do projeto está apresentada na Figura 24, com o escopo do projeto alvo deste PFC demarcado em vermelho claro, de acordo com o contexto do título deste trabalho (implementar uma arquitetura de processamento de dados distribuída).

No desenvolvimento do projeto na empresa cliente, a infraestrutura está hospedada quase totalmente em um *cluster* de computadores na GCP utilizando um serviço chamado GKE.

A vantagem de se ter uma arquitetura totalmente hospedada em um serviço como esse é que é usada em conjunto com software de código aberto gratuito, então não existem limites de licenciamento ou precificação arbitrária de ferramentas, com a desvantagem de requerer uma equipe técnica competente para desenvolver e manter toda a infraestrutura em plena operação. Alguns serviços são ofertados por grandes

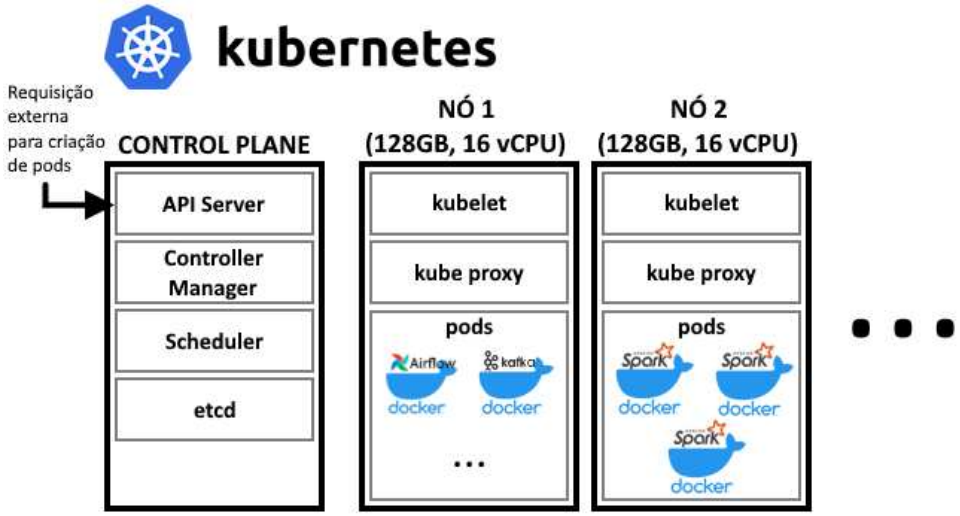
empresas, como Databricks ou Snowflake, que permitem a criação de DWs e toda a arquitetura de processamento de forma gerenciada, mas os custos dessas ferramentas costumam ultrapassar os custos de manter uma arquitetura própria. Um grande problema de utilizar-se dos serviços gerenciados, por exemplo, é o chamado *vendor lock-in*, no qual você passa a depender exclusivamente daquele provedor de serviços e está sujeito à quaisquer práticas abusivas e aumentos de preço que precisam ser acatados pois a arquitetura está inteira contida naquele provedor.

4.4.1 GKE como *cluster* de computadores

O GKE possui algumas características muito relevantes na otimização de custo. O GKE é capaz de auto-escalabilidade baseado na demanda, assim, se os recursos de memória e processamento no *cluster* como um todo estão subutilizados, algumas das máquinas que compõem o *cluster* (chamados nós ou *nodes*) são desligados e deixam de ser faturados. A configuração implementada permite que o *cluster* escalone automaticamente entre cinco e vinte nós, cada um com 16 Virtual Central Processing Unit (vCPU) e 128GB de memória RAM, totalizando até o máximo de 320 núcleos de processamento e 2.5TB de memória ao mesmo tempo. O *cluster* também possui espaço em disco para persistência e *spill* dos conteúdos de memória em disco, e, conforme previamente mencionado, está hospedado na região GCP *southamerica-east1*.

O GKE nada mais é que uma forma escalável de hospedar um *cluster* de computadores cujo gerenciador é composto pela ferramenta de código aberto Kubernetes. Essa ferramenta possui, de forma geral, um módulo chamado *control plane*, que funciona como um grande gerenciador de recursos, que disponibiliza APIs de comunicação que permitem que outros serviços (como o orquestrador Apache Airflow) interajam com os nós e criem contêineres Docker (análogos à uma máquina virtual) dentro de cada nó. No diagrama de *deployment* (Figura 17) fora mencionado que o *cluster* hospeda diversas instâncias (Kafka, Airflow, *applications* do Spark) e justamente essas instâncias que são as máquinas virtuais (na realidade, contêineres Docker, mas são praticamente análogos) que realizam todo o trabalho de replicação, orquestração e processamento, respectivamente. A arquitetura do Kubernetes, conforme está configurada para o projeto está representada na Figura 20:

Figura 20 – Arquitetura de um *cluster* no Kubernetes usando GKE.



Fonte: Acervo do autor.

Ressalta-se que nem todos os nós do cluster são utilizados a todo momento, com o uso médio girando em torno de dez a doze nós em regime permanente, e do total, parte está dedicada na hospedagem do Apache Kafka (responsável pela replicação dos sistemas transacionais), parte para o Apache Airflow (orquestrador), parte para aplicações do Apache Spark sob demanda (inclusive aplicações referentes à outros projetos de dados na empresa), e uma outra parte para outras aplicações diversas do cliente que estão fora do contexto desse trabalho.

4.4.2 Apache Airflow como orquestrador sendo executado dentro do *cluster*

De acordo com o conteúdo apresentado na Seção 3.3.3 e a arquitetura interna do Kubernetes sendo executado no GKE da Figura 20, o Apache Airflow, ou simplesmente Airflow está sendo executado em um contêiner Docker dentro de um dos *pods* em um nó. Esse contêiner fica ligado 24 horas por dia, acompanhando o tempo e disparando requisições ao próprio *cluster* do GKE (via seu **API Server** do *control plane*) para inicializar outros *pods* e contêineres para executar as Spark *applications*, tudo de acordo com uma temporização predefinida por notação *crontab* e com as regras de execução do DAG definido pelo engenheiro de dados (sequências de tarefas, pré-requisitos, o que fazer se alguma etapa falhar, etc.).

4.4.3 Apache Spark como motor de processamento sendo executado dentro do *cluster*

Similarmente ao já apresentado, e tomando como base as teorias explicitadas na Seção 3.5, o Apache Spark, ou simplesmente Spark é instanciado no *cluster* do

GKE através de tarefas do Airflow, assim, se o orquestrador faz uma requisição específica para o **API Server** do GKE (com parâmetros de configuração bem definidos), então um novo *pod* e novos contêineres Docker serão criados em algum dos nós do *cluster*, operação gerenciada pelo *kubelet* que vive dentro de cada nó (faz a verificação de quanto de memória e vCPU alocar).

Cada *application* do Spark irá criar contêineres para o *driver* e quantos *executors* forem necessários de acordo com a especificação da requisição feita pelo Airflow ao **API Server**.

O Spark, por sua vez, irá realizar as tarefas definidas em um arquivo Python (extensão `.py`) que contém efetivamente a lógica dos *pipelines*, escritos usando a biblioteca PySpark nativa do Spark. Essa biblioteca possui métodos como `df1.join(df2)` para realizar operações de `JOIN` de SQL diretamente entre dois *dataframes* (um *dataframe* é análogo à uma tabela). Existem diversos métodos que serão usados nos *pipelines* como `df.distinct()` para remover registros duplicados, `df.withColumnRenamed('col', 'colNew')` para renomear colunas em um *dataframe* e muitos outros métodos de **programação funcional** que serão melhor abordados no Capítulo 5.

Assim que uma *application* finaliza seu processamento, ela é terminada e os recursos que seu *driver* e *executors* são liberados no *cluster*, podendo ser alocados para outros serviços ou até mesmo, se inutilizados, pode ter o nó inteiro desligado pelo GKE, economizando na fatura da nuvem no fim do mês.

4.4.4 GCS como camada de armazenamento

O Google Cloud Storage, ou apenas Cloud Storage é um serviço repositório de arquivos escalável e barato. A organização interna do GCS dentro do projeto da GCP consiste na criação de grandes "pastas" chamadas *buckets*, e dentro de cada *bucket* é possível armazenar, novamente numa estrutura de um sistema de arquivos similar aos de sistemas operacionais padrão como *Windows*.

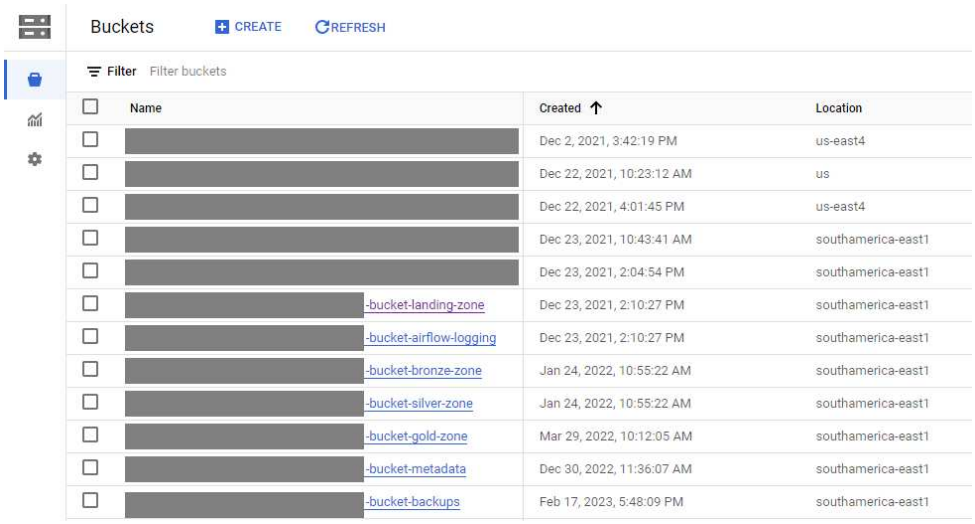
Na arquitetura do projeto, cada camada do medalhão possui um único *bucket* dedicado, dentro do qual uma estrutura de pastas interna categoriza cada tabela, por exemplo:

- Para armazenar a tabela LIKP (tabela cabeçalho de ocorrências de frete) na *landing zone*: fica no *bucket* dedicado para *landing*, em uma pasta com nome LIKP;
- Para armazenar a tabela LIKP (tabela cabeçalho de ocorrências de frete) na *bronze zone*: fica no *bucket* dedicado para *bronze*, em uma pasta com nome LIKP, e assim por diante;

No caso de tabelas da *gold zone*, o elas ficam armazenadas no *bucket* dedicado para *gold* em pastas com o nome da fato ou dimensão que representam (como *fato_ordens_de_venda* ou *dim_clientes*).

Na Figura 21 a representação dos *buckets* criados na interface gráfica *web* do GCS:

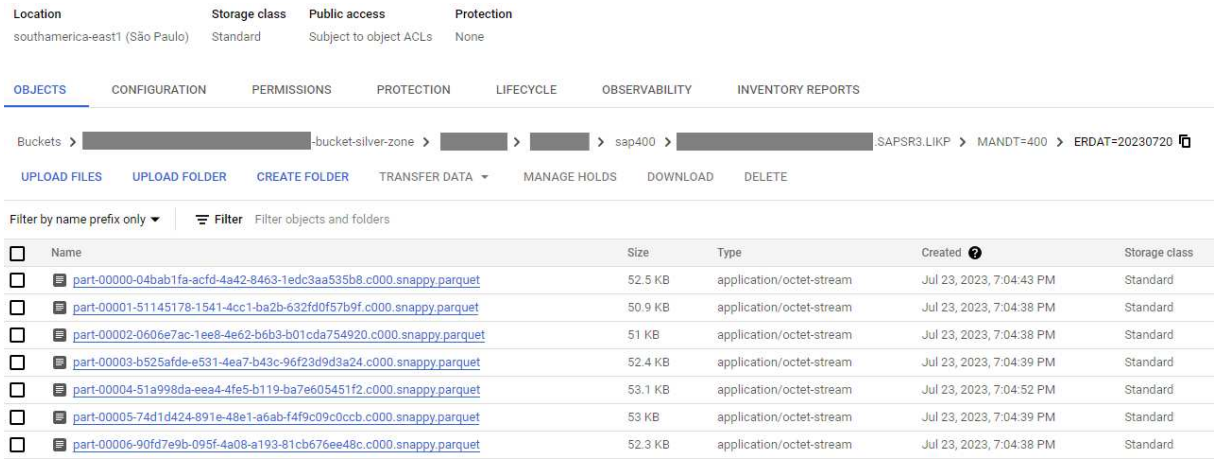
Figura 21 – *Buckets* criados no GCS.



Fonte: Acervo do autor.

Já a Figura 22 tem-se exemplos de registros para a camada *silver* da tabela LIKP, também na interface gráfica *web* do GCS e demonstrando diversos arquivos de extensão `.parquet` que compõem parte da tabela:

Figura 22 – Alguns arquivos da tabela LIKP *silver* do GCS.



Fonte: Acervo do autor.

Finalmente, na Figura 23, uma ilustração dos conteúdos de um desses arquivos `.parquet` que possui 83 registros contendo parte dos dados dessa tabela. A tabela completa seria a leitura de todos os `.parquet` associados à versão mais atual da tabela concatenados.

Figura 23 – Conteúdos de um dos arquivos `.parquet` da tabela LIKP na *silver*.

File Edit Tools Help

Filter Query (?): Execute Clear Record Offset: 0 Record Count: 1000

_OP	_TIMESTAMP_KAFKA	_SCN	_COMMIT_SCN	_PK_COMPOSITE	_LAST_UPDATED_AT	_YEAR	_MONTH	_DAY	_HOUR	/BEV1/RI	TRAGR	LPRIO	VOLUM	GEWEI
u	21-Jul-23 2:59 PM	1032538668458	1032538666766	400|0806942785	23-Jul-23 9:59 PM	2023	7	21	14	0	0001	01	3.840	KG
u	21-Jul-23 12:00 PM	1032512226340	1032512226487	400|0807957092	23-Jul-23 9:59 PM	2023	7	21	12	0	0001	01	4.698	KG
u	21-Jul-23 2:14 PM	1032531972197	1032531972460	400|0807981280	23-Jul-23 9:59 PM	2023	7	21	14	0	0001	02	48.567	KG
u	20-Jul-23 11:50 PM	1032407600433	1032407600142	400|0807986362	23-Jul-23 9:59 PM	2023	7	20	23	0	0001	01	3.840	KG
u	20-Jul-23 11:41 AM	1032274938577	1032274938530	400|0808005271	23-Jul-23 9:59 PM	2023	7	20	11	0	0001	01	3.189	KG
u	21-Jul-23 3:22 PM	1032543155399	1032543155430	400|0808010765	23-Jul-23 9:59 PM	2023	7	21	15	0	0001	02	4.698	KG
u	20-Jul-23 9:36 PM	1032383482391	1032383469104	400|0808015786	23-Jul-23 9:59 PM	2023	7	20	21	0	0001	01	3.840	KG
u	20-Jul-23 10:35 PM	1032395493500	1032395493399	400|0808016930	23-Jul-23 9:59 PM	2023	7	20	22	0	0001	01	3.840	KG
u	21-Jul-23 10:19 AM	1032498044339	1032498029529	400|0808021584	23-Jul-23 9:59 PM	2023	7	21	10	0	0001	01	16.467	KG
u	21-Jul-23 3:21 PM	1032543046056	1032543045854	400|0808024892	23-Jul-23 9:59 PM	2023	7	21	15	0	0001	01	16.467	KG
c	20-Jul-23 11:34 AM	1032274160261	1032274156935	400|0808027683	23-Jul-23 9:59 PM	2023	7	20	11	0	0001	02	0.000	

Showing: 83 Results Loaded: 0 to 83 Out of: 83

Fonte: Acervo do autor.

4.4.5 BigQuery como camada de disponibilização *self-service* de dados

O serviço BQ descrito previamente na Seção 4.3.2 é usado como uma camada *web* na nuvem (ou seja, podendo ser acessado de qualquer lugar, desde que o usuário se autentique) que permite a consulta livre aos dados armazenados no GCS. Conforme mencionado, o BQ também disponibiliza APIs nativas da para interação com as suas *features*, permitindo realização de consultas de forma programática (para alimentar sistemas ou modelos de AI e ML) ou consultas *ad hoc* por quem desejar, efetivamente resolvendo os problemas apresentados pelo cliente no início do projeto, fazendo dos dados um produto *self-service* para todo o corpo de analistas, automaticamente atualizado, limpo e bem caracterizado para as entidades de negócio envolvidas.

4.5 ARQUITETURA FINAL

Com isso, estão especificados todos os componentes principais implementados. Um último diagrama ajuda a entender melhor o escopo do projeto no contexto da implementação da arquitetura de processamento distribuído de dados, no qual é possível visualizar que muito embora algumas ferramentas estejam incluídas no escopo do projeto (como o Airflow e o Kafka), estes, em específico, não foram implementados ou configurados pelo autor, e são apenas utilizados diretamente (no caso do Airflow) ou indiretamente (no caso do Kafka) na execução dos processamentos dos dados.

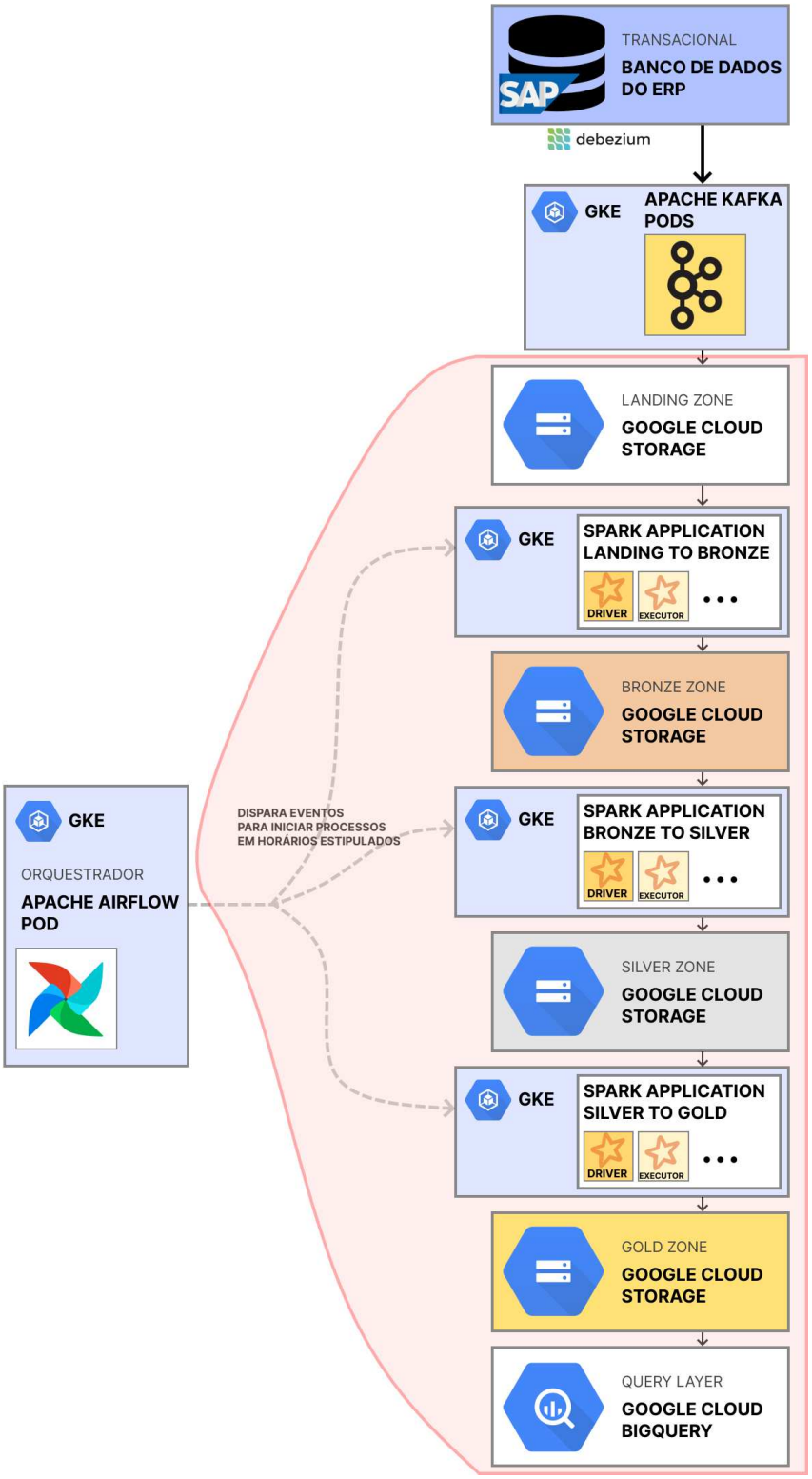
Assim, claramente percebe-se que o projeto em questão inicia-se fazendo as transições dos conjuntos de dados a partir da *landing zone* para a *bronze zone* a

partir de Spark *applications* executadas por DAGs configuradas no Airflow. Tudo isso está dentro do escopo do projeto, e o mesmo repete-se na transformação de dados da *bronze zone* para *silver zone* e da *silver zone* para *gold zone*, ambos também utilizando DAGs do Airflow.

A última etapa no fluxo da arquitetura é a criação das **external tables** no BigQuery, sendo este o ponto de contato entre consumidores e o produto entregue no projeto, conforme discutido na Seção 4.4.5.

O diagrama está apresentado na Figura 24.

Figura 24 – Infraestrutura do projeto. Área delimitada em vermelho indica o escopo de atuação.



Fonte: Acervo do autor.

5 IMPLEMENTAÇÃO

Neste capítulo serão descritos os detalhes de implementação do projeto, envolvendo o entendimento do seu escopo, desenvolvimento dos fluxos de dados e ETL, orquestração, dimensionamento de infraestrutura e disponibilização dos dados.

5.1 ENTENDENDO O ESCOPO DO PROJETO

As primeiras etapas de um projeto no escopo de dados consistem na integração entre consultor e os sistemas internos da empresa cliente, além da familiarização do mesmo com o modelo de negócios praticado. Esses são pontos relevantes pois o consultor possui a liberdade (e o dever) de opinar e discutir requisitos e objetivos junto ao cliente, fornecendo uma abordagem técnica crítica acoplada ao mínimo de conhecimento sobre o negócio em si, os quais resultaram nos objetivos apresentados no início desta monografia, no Capítulo 1.

Dessa forma, as semanas iniciais de projeto foram repletas de conversas, integrações e apresentações de equipes e times. Foram definidas as metodologias de projeto aplicadas conforme discutidas no Capítulo 4, foram requisitados e validados os acessos à repositórios, documentações internas e sistemas internos necessários (por exemplo, acesso e permissões ao projeto da GCP da empresa cliente). Além disso, algumas apresentações foram feitas apenas referentes ao sistema ERP e o SGBD relacionado.

Ao total, foram mapeadas inicialmente cinquenta e três tabelas residentes no SGBD Oracle responsável pela persistência e operacionalização do ERP SAP do cliente, e foram ativadas as rotinas de extração e replicação das operações ocorrentes nessas cinquenta e três tabelas diretamente para uma “pasta” (*bucket*) do GCS pertencente à *landing zone*. Essa etapa não está no domínio do projeto e foi realizada por uma outra equipe de consultoria. Dessas cinquenta e três, uma parte seria usada nas rotinas de processamento para eventualmente gerar as tabela da *gold zone* (fatos e dimensões), enquanto outras seriam parte de outros projetos ou usadas apenas para consultas *ad hoc* na *silver zone*.

Com a replicação em tempo real dos fatos ocorridos nestas tabelas, foi possível aprofundar o conhecimento técnico e de negócios por detrás das tabelas do SAP. Esse sistema é mantido pela empresa SAP SE, multinacional alemã que possui uma forte padronização e tipagem de dados, visto que é difundida no mundo inteiro em diversas áreas de negócio como na indústria de óleo e gás, varejo, manufatura e tecnologia. Isso acarreta em um alto degrau técnico a ser abordado pois, por exemplo, para obter registros referentes à ordens de venda (uma única entidade), precisa-se atentar e saber relacionar registros em várias tabelas, nomeadamente VBAK (cabeçalho de ordens de venda), VBAP (dados de itens de ordens de venda), VBEP (dados de frete para itens de

uma ordem) e VBKD (outras informações de negócio para uma ordem de venda), sendo que cada uma dessas tabelas possui n colunas (muitas vezes com $n > 300$), nomes de tabelas e colunas consistindo de acrônimos em alemão e documentação praticamente inexistente ou defasada. Além disso, dependendo da tabela e sua granularidade, seu volume passa da faixa de bilhões de registros, consistindo vários conjuntos de *big data* não documentados, não normalizados, sem um ERD bem definido e com muitas regras de negócio individuais não explícitas.

Um levantamento foi inicialmente feito pelo consultor em busca de melhor ambientar com as nomenclaturas, regras de negócio e particularidades dos sistemas implementados pelo ERP (ajudando também a ambientar ou repassar conhecimento com as equipes internas), elaborando um documento de consulta com traduções e nomes descritivos exportados dos sistemas SAP, conforme mostra a Figura 25. Apresentadas estão as descrições de colunas para a tabela VBAK, referente aos dados de cabeçalho de ordens de venda, isto é, as informações principais de cada ordem, evidenciando campos como o número do documento de vendas em si, datas de remessa, escritórios, equipes e organizações responsáveis pela ordem, moeda e muitos outros que não estão apresentados.

Figura 25 – Excerto de uma planilha de mapeamento dos nomes de tabelas e colunas.

1	Tabela	Campo	Chv Primária	Descrição breve	Nome breve	Título
1856	VBAK	TRVOG		Grupo p/operação de transação	GrpOpTrsaç	GOT
1857	VBAK	UPD_TMSTMP		Registro hora UTC forma descritiva (JJJJMMTThhmmssmmmuuun)	Reg.hora	Registro da hora
1858	VBAK	VBELN	X	Documento de vendas	Doc.venda	Doc.venda
1859	VBAK	VBELN_GRP		Nº do contrato de grupo	ContrGrupo	Contr.grupo
1860	VBAK	VBKLA		Sistema de origem com release e controle de operações	Origem	Origem
1861	VBAK	VBKLT		Classificação do documento SD	Código	C
1862	VBAK	VBTYP		Ctg.documento de vendas e distribuição	Ctg.doc.SD	CgDSD
1863	VBAK	VDATU		Data desejada de remessa	DtDesjRem.	DtDesjRem.
1864	VBAK	VGBEL		Nº documento do documento de referência	Doc.ref.	Doc.ref.
1865	VBAK	VGTYP		Ctg.documento de venda e distribuição (SD) precedente	Ctg.doc.SD	CgPrc
1866	VBAK	VKBUR		Escritório de vendas	EscrVendas	EscrV
1867	VBAK	VKGRP		Equipe de vendas	Eq.vendas	EqVs
1868	VBAK	VKORG		Organização de vendas	Org.vendas	OrgV
1869	VBAK	VPRGR		Período proposto para a data	Período	Per.
1870	VBAK	VSBED		Condição de expedição	CondExped.	CE
1871	VBAK	VSNMR_V		Nº versão do documento de vendas	Versão	Versão
1872	VBAK	VTWEG		Canal de distribuição	CanalDistr	CDst
1873	VBAK	VZEIT		Hora proposta de divisão da remessa (local c/ref.org.venda)	HrDesForn.	HorDesForn
1874	VBAK	WAERK		Moeda do documento SD	Moeda	Moeda
1875	VBAK	XBLNR		Nº documento de referência	Referência	Referência
1876	VBAK	XEGDR		Código: negócio triangular no âmbito da UE?	NegTriang.	
1877	VBAK	ZBLOCO		Bloco de Atendimento	BlocoAtend	Bloco de Atendimento
1878	VBAK	ZCODLINX		Código LINX	Cod. LINX	Código do LINX
1879	VBAK	ZCOPI		Envia Cópia Pedido	Env Cop Pe	Envia Cópia Pedido
1880	VBAK	ZCOVEX		Cód. Comissão Vendedor Ext.	C.ComVExt	Cód. Comissão Vendedor Ext.
1881	VBAK	ZCOVIN		Cód. Premiação Vendedor Interno	C.Pre.VInt	Cód. Premiação Vendedor Interno
1882	VBAK	ZDIAD		Dia D	Dia D	Dia D
1883	VBAK	ZDOCREFAT		Elemento de dados para Documento de Refaturamento	Doc. Refat	Documento de Refaturamento
1884	VBAK	ZDREF_RFT		Doc. Referência de Refaturamento		
1885	VBAK	ZEXRAT		Código de exceção rateio	Exc.Rateio	CodExRat

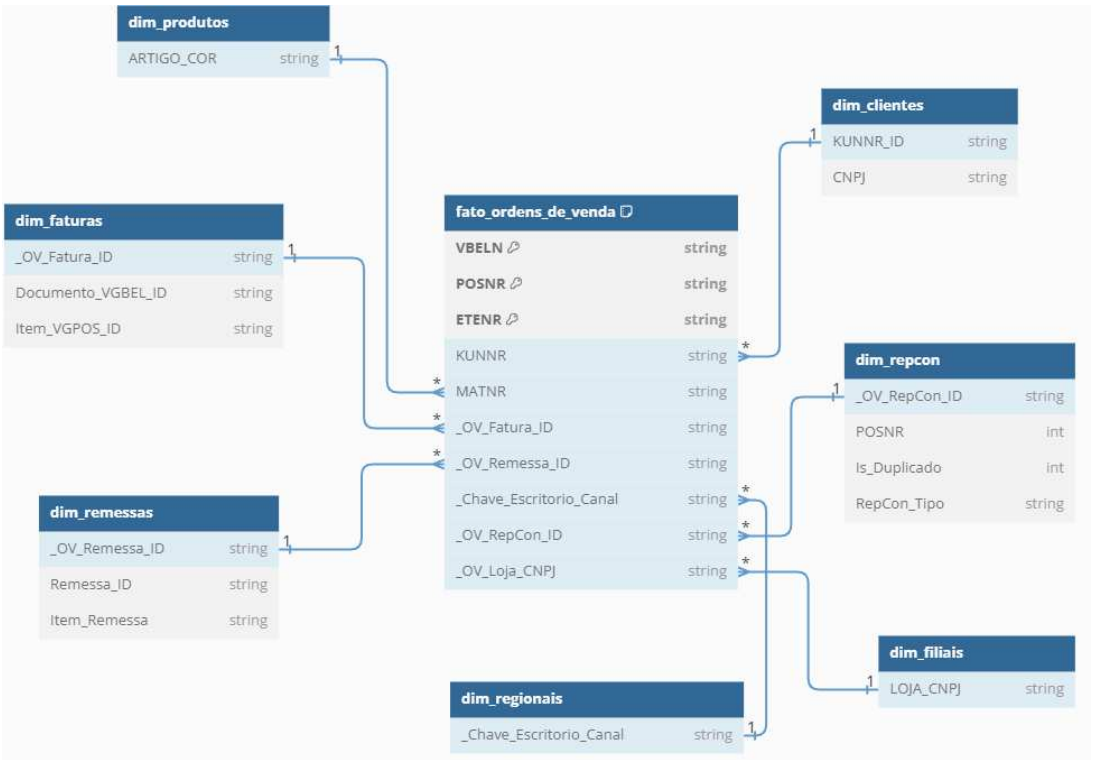
Fonte: Acervo do autor.

Com esse levantamento em mãos e a ajuda da equipe interna responsável pelos sistemas do SAP, foi possível mapear uma versão inicial do modelo de dados final desejado na *gold zone*, assim como priorizar corretamente quais tabelas do SAP seriam as cruciais para o andamento do projeto BI Carteiras e quais poderiam esperar um momento futuro. Essa priorização é crítica para qualquer tipo de projeto, mas em

especial os com escopo aberto e amplo como o BI Carteiras pois é bastante comum esse tipo de integração levar um bom tempo para ser concluído, e se as tarefas não forem bem definidas com entregáveis claros e plausíveis, a gerência e líderes técnicos podem ficar muito tempo sem atualizações, causando desgaste no relacionamento e da parceria entre cliente e consultoria.

A partir disso, foram mapeados alguns campos inicialmente considerados prioritários no modelo estrela de dados. A Figura 26 retrata, de forma resumida (algumas simplificações foram feitas na representação como a redução do número de colunas demonstrado), o modelo de dados a ser desenvolvido a partir das tabela do sistema transacional. Foram suprimidas muitas das colunas presentes pois a fato, sozinha, possui atualmente um total de 254 colunas, não sendo possível uma boa representação em um relatório.

Figura 26 – ERD para o modelo estrela inicial a ser desenvolvido na *gold zone*.



Fonte: Acervo do autor.

5.2 IMPLEMENTAÇÃO DA ARQUITETURA DE PROCESSAMENTO EM APACHE SPARK PARA CAMADAS *BRONZE* E *SILVER*

Com o modelo de dados em mãos e as tabelas-base dos sistemas transacionais já tendo seus eventos capturados e replicados, o consultor pôde atuar mais especificamente em cima das demandas do projeto. Na priorização determinada na avaliação de

escopo, as primeiras entregas consistem no processamento dos eventos registrados na *landing zone* até que se obtenha uma réplica idêntica e fiel das bases transacionais na camada de armazenamento do DW na *silver zone*, passando pela *bronze zone*.

5.2.1 Implementando a orquestração dos *pipelines* com Apache Airflow

A nível de código, a infraestrutura implementada permite utilizar-se da ferramenta de orquestração Airflow para inicializar os *pods* contendo os contêineres no *cluster* de computadores rodando no serviço GKE. Assim, quando uma tarefa existente no Airflow for disparada, ela imediatamente dispara a execução de uma Spark *application* no GKE, que por sua vez instancia contêineres para o *driver* e um ou mais *executors*. Cada tarefa do Airflow é configurada com alguns parâmetros como o número desejado de *executors*, a quantidade de memória e vCPUs para cada, além de alguns metadados como o nome da tabela a ser processada, locais de armazenamento no repositório para a tabela e alguns campos para realizar o particionamento na hora de processar e paralelizar as tarefas do Spark. Basta definir uma tarefa do Airflow com o operador `SparkKubernetesOperator` (nativo da biblioteca Python do Airflow), configurar os parâmetros desejados e disparar a execução de um *pipeline* como apresentado abaixo, nesse exemplo para processar a tabela VBAK da *landing zone* para *bronze zone* com 8 executores de 8GB de memória e 4 núcleos de processador cada:

```
# Instanciar um Spark job inicializando pods no GKE
spark_job = SparkKubernetesOperator(
            # string de conexao do Airflow com o cluster
            kubernetes_conn_id='gke_conn_id',
            # dicionario de parametros para os pods a serem criados
            params=dict(
                app_name='SAP-landing-to-bronze-VBAK',
                table='VBAK',
                spark_file='spark-scripts/etl/sap/landing-to-bronze.py'
                landing_path='gs://path/to/landing-zone/VBAK/',
                bronze_path='gs://path/to/bronze-zone/VBAK/',
                partition_fields=['ERDAT'],
                executor_instances=8,
                executor_memory='8g',
                executor_cores=4,
                driver_memory='4g',
                driver_cores=4
                ... # Outras configuracoes
            ),
            do_xcom_push=True,
            ... # Outros parametros
        )
```

A partir disso, basta configurar essa tarefa do Airflow para ser executada em intervalos regulares de tempo para que a rotina definida pelo arquivo `spark-scripts/etl/sap/landing-to-bronze.py` definido no trecho de código seja executada, usando a notação *crontab*. A mesma lógica pode ser replicada para execução de processos para

as camadas *silver* e *gold*, bastando que se alterem as configurações, caminhos e particularidades de cada tabela, além de principalmente o arquivo definido em `spark_file`, no qual as operações e lógica de processamento são efetivamente escritas usando a notação da API de Spark para Python, PySpark, assunto das próximas subseções.

É de importância ressaltar que, para essa abordagem funcionar, ou define-se um *script* do Spark para cada tabela (o que não é factível, dado que são pelo menos 53 tabelas a serem processadas no total), ou generaliza-se e parametriza-se o script para funcionar com todas as tabelas. Isso é possível para as camadas *bronze* e *silver* pois todas as tabelas devem sofrer as mesmas operações sobre os eventos capturados (deduplicação, formatação básica, etc.), que foi a abordagem adotada para o projeto. Com isso, não precisou-se escrever um arquivo Python `.py` para processar cada tabela, padronizando transformações e operações para todas as tabelas. Essa implementação efetiva a completude do requisito NF1 descrito na Tabela 6, garantindo que os *pipelines* serão executados pelo menos diariamente.

5.2.2 Interpretação dos eventos de CDC na geração da *bronze zone* a partir da *landing zone*

A etapa de transformação dos dados da *landing zone* para a *bronze zone* é fundamental para assegurar a rastreabilidade das informações dentro do processo de ETL. Na arquitetura medalhão implementada, a camada *bronze* é uma réplica otimizada da *landing zone*, isto é, a informação contida ainda permanecerá sendo a nível de eventos ocorridos no SGBD Oracle, porém com todas as vantagens de migração de formato de `.json` para `.parquet`, isto é, facilmente compressível e descompressível, rápido para ser escrito e ainda otimizado para leitura.

Na execução dos *scripts* responsáveis pelo processamento em si (que são disparados pelo Airflow) uma camada extra de interpretação de armazenamento é usada: Delta Lake, que assegura uma boa interpretação das tabelas a partir dos vários arquivos `.parquet` gerados e permite aos usuários consultarem versões prévias da tabela com o versionamento de arquivos. Um pseudocódigo para as etapas realizadas pelo *script* de processamento da *landing* para *bronze* é fazer a leitura da captura do CDC, filtrar apenas o que faz sentido para o negócio (por exemplo, operações de deleção vêm com dados na estrutura “after” nulos, dado que após a deleção o registro teoricamente não existe mais, assim, para operações de deleção precisa-se olhar para a estrutura “before”) e salvar no caminho para a tabela na *bronze zone*:

```
# inicializar spark e suas dependencias
spark = start_and_configure_spark()

# ler dados da landing zone que estao em formato .json
raw_data = read_records_from_gcs(spark, LANDING_ZONE_PATH)

# se estivermos lendo eventos (operacao) de delete, dados estarao nulos em 'after',
```

```
# pois apos delecao nao existe registro, portanto devemos ler estrutura em 'before'
# para saber qual registro foi deletado no banco transacional.
delete_ops = raw_data.where('__op' == 'd')['before']

# se estivermos lendo outras operacoes, usar dados da estrutura em 'after'.
other_ops = raw_data.where('__op' != 'd')['after']

# unir todas as operacoes em um unico dataframe
all_ops = delete_ops.union(other_ops)

# salvar dados na camada bronze
save_data_to_gcs(spark, BRONZE_ZONE_PATH, all_ops)
```

Essa abordagem, entretanto, é extremamente ineficiente pois esse *pipeline* estaria reprocessando todos os eventos a cada execução. Se um evento fora processado no dia de ontem, nada está impedindo de fazer a leitura e reprocessamento no dia de hoje de forma totalmente desnecessária, visto que o evento já ocorreu e já fora processado. Para tal, implementam-se lógicas de processamento incremental, muito utilizadas em engenharia de dados para que não haja retrabalho desnecessário, sendo que a primeira vez em que a tabela é processada ocorre uma carga completa, e a partir desse ponto apenas a diferença (novos registros/eventos) são considerados.

5.2.2.1 Cargas incrementais e cargas completas na etapa de *landing* para *bronze zone*

A lógica de implementação de uma carga incremental é relativamente simples, mas possui um requisito específico: a existência de um campo incremental não repetível para cada registro único de uma tabela, e que esse campo seja preferencialmente particionado ou rápido para ser filtrado. Normalmente utiliza-se um campo dos próprios sistemas que fazem a captura CDC, que automaticamente, junto aos dados propriamente ditos, trazem alguns metadados para cada evento incluindo a data e hora de captura utilizando um campo de `timestamp`.

Com isso, para cada registro único da tabela (determinado pela Primary Key (PK)), é possível filtrar e ordenar pela sua data de captura, assim, pode-se montar a cronologia de eventos que afetaram um dado registro, assim como visto na Figura 5 e explicado na Seção 3.3.1.

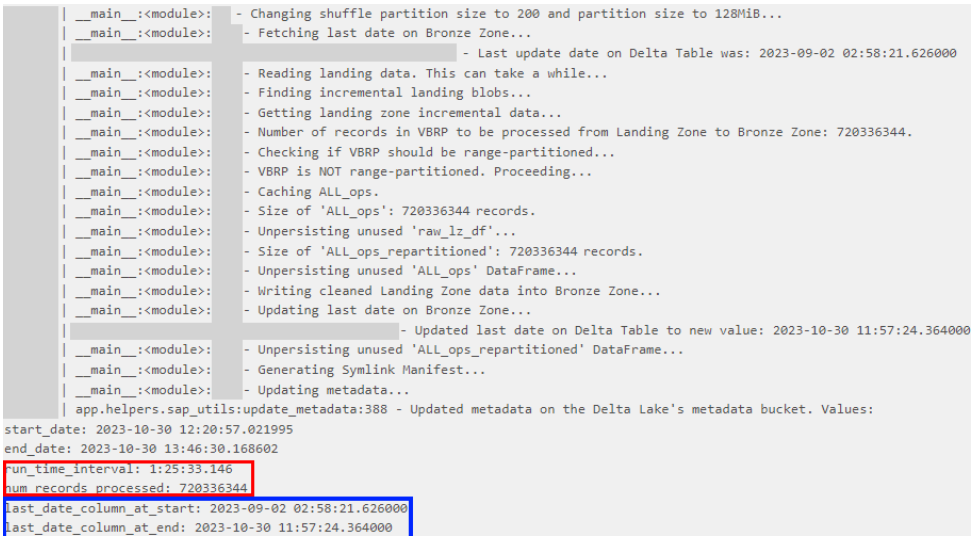
Para os *pipelines* de processamento, basta que se realize um filtro nos dados da *landing zone* para que sejam lidos apenas registros com a data de captura maior que a maior data de captura da execução passada. Dessa forma, é preciso manter, externamente, *logs* com a última data de captura para cada execução do *pipeline*, algo como o apresentado no pseudocódigo abaixo:

```
# inicializar spark e suas dependencias
spark = start_and_configure_spark()

# pegar o maior timestamp de processamento da ultima execucao salvo em um arquivo
```

```
# de metadados como 'last_date.json' para esta tabela
LAST_EXECUTION_TIMESTAMP = read_last_execution_max_timestamp(BRONZE_ZONE_PATH)

# leia registros da landing com timestamp maior que o maximo da ultima execucao
raw_data = read_records_with_filter(spark, LANDING_PATH, LAST_EXECUTION_TIMESTAMP)

# se estivermos lendo eventos (operacao) de delete, dados estarao nulos em 'after',
# pois apos delecao nao existe registro, portanto devemos ler estrutura em 'before'
# para saber qual registro foi deletado no banco transacional.
delete_ops = raw_data.where('__op' == 'd')['before']

# se estivermos lendo outras operacoes, usar dados da estrutura em 'after'.
other_ops = raw_data.where('__op' != 'd')['after']

# unir todas as operacoes em um unico dataframe
all_ops = delete_ops.union(other_ops)

# salvar dados na camada bronze
save_data_to_gcs(spark, BRONZE_ZONE_PATH, all_ops)

# pega o maior timestamp da execucao atual a ser salvo no novo 'last_date.json'
CURRENT_EXECUTION_MAX_TIMESTAMP = all_ops.orderby(timestamp, 'DESC')[0]

# salvar um novo arquivo de metadados como last_date.json com novo valor
update_last_timestamp_metadata(CURRENT_EXECUTION_MAX_TIMESTAMP)
```

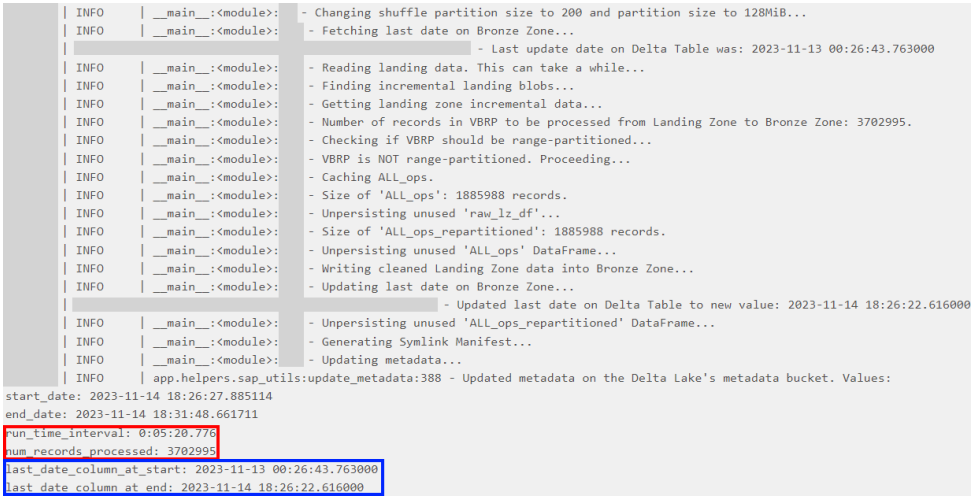
Assim, não mais é necessário reprocessar todo o conjunto da *landing zone* todos os dias, tornando a etapa de transformação muito mais rápida e eficiente, em alguns casos de tabelas relativamente grandes, como apresentado na Figura 27, cargas completas (evidenciado em azul) podem demorar até algumas horas para serem finalizadas em conjuntos com cerca de 720 milhões de registros (em vermelho), isso sabendo que da etapa *landing* para *bronze* não é feito nenhum processamento pesado, apenas leitura de `.json` não otimizados, pequenas conversões e condicionais (tratamento para operações de deleção) e salvamento. A carga incremental é usualmente muito mais rápida, conforme mostra a comparação da Figura 27 com Figura 28, essa última que demonstra que 3.7 milhões de registros foram processados em cerca de 5 minutos (demarcado em vermelho) utilizando o filtro incremental de data de captura maior que a maior data da última execução (evidenciado em azul), no intervalo dos dias 13 de Novembro e 14 de Novembro de 2023. Alguns trechos foram censurados para preservar a identidade do cliente.

Figura 27 – Exemplos de *logs* da aplicação em produção com carga completa.



Fonte: Acervo do autor.

Figura 28 – Carga incremental feita ao filtrar a *landing zone*.



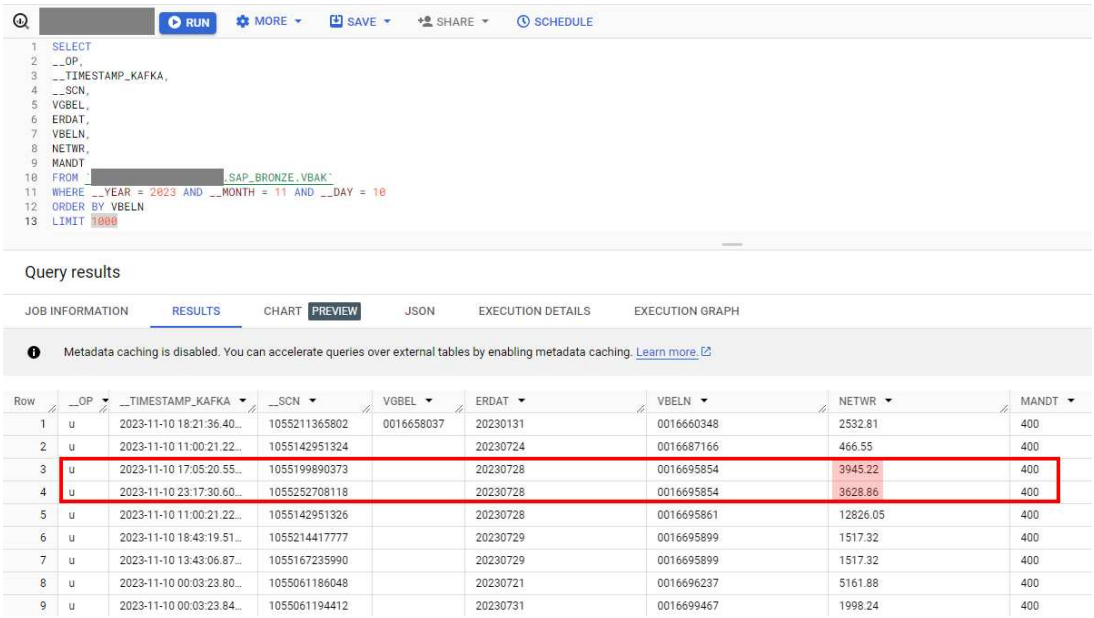
Fonte: Acervo do autor.

Esse tipo de carga incremental é extremamente eficaz (na questão de necessidade de recursos do *cluster* e tempo de processamento) para tabelas que funcionam como *append-only*, isto é, registros nunca são modificados ou deletados da tabela, apenas inseridos. Como a *landing zone* é uma tabela que remete à eventos, ela é do tipo *append-only* (um evento nunca é “desfeito”, mas pode-se ter um evento que desfaça o que um evento anterior fez). A *bronze zone*, portanto, como se trata de uma otimização para consulta desses eventos, também é *append-only*. Algumas complexidades se dão quando se deseja realizar uma carga incremental na qual um registro

com uma dada PK deve ser atualizado e não apenas colocado na tabela (como é o caso da *silver zone*), que serão abordadas na Seção 5.2.3.

O resultado esperado no fim do processamento das tabelas da *bronze zone* são um compilado de eventos cuja cronologia é verificável e consultável para eventuais análises de ciclo de vida de pedidos. Imaginando que uma ordem de venda possa ser criada, modificada (valores alterados) por diversos motivos (cancelamentos parciais, descontos, atualizações de cadastro, câmbio ou outros motivos), é possível utilizar as tabelas do ERP replicadas com seus eventos e filtrar por um registro específico de interesse, ordenando os eventos que ocorreram para este registro de forma cronológica para observar as mudanças que o dado sofreu ao longo do tempo. Uma avaliação como essa está representada na Figura 29, na qual é possível identificar em qual horário e quais modificações um determinado registro sofreu no sistema transacional, evento esse replicado na *landing* e disponibilizado na *bronze zone*. Nessa representação, a PK desta tabela é o campo VBELN, e em destaque é possível observar que um mesmo registro (um mesmo valor de VBELN) sofreu duas operações no período filtrado: uma às 17h05m da tarde, operação de atualização que possuía campo NETWR (valor de ordem) definido como 3945,22, e outra atualização às 23h17m da noite, que atualizou o mesmo campo NETWR para um valor de 3628,86. Sem avaliar outros campos, pode-se presumir a partir de um conhecimento do negócio que houve uma operação de cancelamento parcial da ordem de venda (a compra de alguns dos produtos daquela ordem específica pode ter sido cancelada pelo cliente, causando redução do valor da ordem).

Figura 29 – Excerto de eventos na *bronze zone* para a tabela VBAK com dados de cabeçalho de ordens de venda no BigQuery.



Fonte: Acervo do autor.

Sem um registro histórico dos *logs* de transação em um DW bem estruturado, esse tipo de evento de cancelamento parcial de ordens de venda seria perdido, pois não seria possível recuperar de forma fácil o valor de ordem anterior ao cancelamento. Uma métrica derivável dessa tabela, portanto, seria obter a taxa de cancelamento parcial de pedidos, juntamente com o valor total cancelado, tornando possível uma análise de perda de faturamento por cancelamentos de ordens por parte dos clientes, metrificando o quão crítico isso é para a empresa e dando um norte às decisões a serem tomadas, como escalar as análises para validar o por quê dessas ordens estarem sendo parcialmente canceladas na tentativa de mitigar ou evitar que isso ocorra.

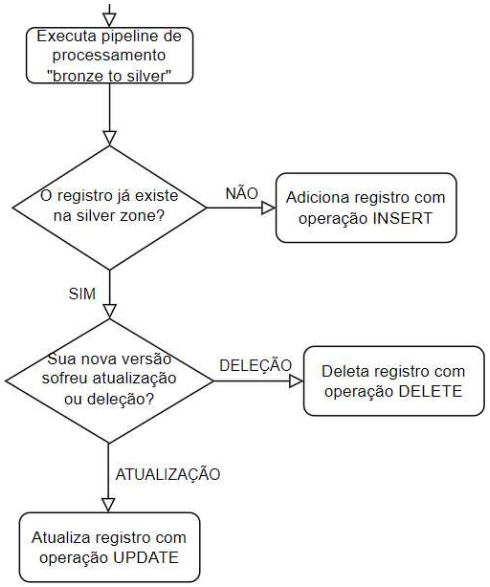
Conclui-se que, muito embora o objetivo primário das tabelas *bronze* (nem mesmo das tabelas *silver*) seja fazer análises complexas, isso não as inviabiliza para tal, e por mais que o objetivo final do projeto é a entrega da camada *gold*, as outras camadas podem ser tão importantes quanto para uma boa análise de faturamento. Além disso, esse objetivo é caracterizado pelo requisito F4 da Tabela 5, cumprindo a entrega das tabela *bronze*.

5.2.3 Condensação de eventos da *bronze zone* em registros de-duplicados para a *silver zone*

Conforme mencionado no parágrafo anterior e na Figura 30, a partir da camada *silver* as complexidades de processamento aumentam consideravelmente, pois, para

esse caso em que a tabela não é *append-only*, cada registro a ser processado de forma incremental precisa consultar a tabela *silver* atual e fazer uma comparação, de tal forma que se o registro não existe (condição determinada com base na existência de uma PK) então é inserido. Se ele existe, deve-se decidir o que fazer com o registro com base no tipo de evento sendo avaliado: se for uma deleção, deve-se deletar o registro e for uma atualização, deve-se atualizá-lo. A árvore de decisão que representa essa lógica está na Figura 30. Caso essa validação não seja feita, podem ocorrer situações de inserção de um mesmo registro múltiplas vezes, o que gera duplicação de dados e fere totalmente o objetivo de manter uma réplica do banco transacional na *silver zone*. Fazer a verificação de "O registro já existe na *silver* zone" é uma operação custosa de se fazer registro a registro da carga incremental, pois precisa-se encontrar um registro específico no meio de potencialmente bilhões de outros registros. Ainda assim, essa abordagem é muito mais rápida que fazer cargas completas.

Figura 30 – Árvore de decisão para operação de carga incremental quando uma tabela não é *append-only*.



Fonte: Acervo do autor.

Essa operação de comparação com múltiplos resultados possíveis a depender do estado inicial da tabela alvo é chamada de *merge*, e existem comandos e sintaxe em SQL para descrever as operações a serem feitas caso o registro exista e algumas condições sejam satisfeitas. Essa operação é utilizada nos *scripts* do Spark para realizar o processamento de dados da camada *bronze* para *silver* e estará descrita adiante nesta seção.

Antes de tudo, porém, é crítico que se entenda como, a partir de *logs* de eventos ocorridos no banco transacional (replicados na *landing* e trazidos até a camada *bronze*)

que se deriva uma réplica exata do sistema transacional. Essa é uma operação de condensação, na qual um registro qualquer deve possuir o estado e valores associados ao último evento ocorrido.

Sendo assim, se um registro originalmente criado com PK = 10 e valor 50 sofreu, cronologicamente, sua criação original e duas atualizações (uma primeira atualização para o valor 40, e uma segunda para o valor 45), então se a segunda atualização é o último evento associado à esse registro de PK = 10, então pode-se assumir que seu estado atual no banco de dados relacional é que sua PK continua valendo 10 (pois é imutável), e seu valor atual é 45. Se desejamos obter, portanto, uma réplica do banco, precisa-se, para cada registro único, ordená-los pela data de captura dos eventos e utilizar o valor contido no evento mais recente.

A exceção à essa regra seria, teoricamente, nos eventos de deleção. Para esses eventos seria necessário deletá-los da tabela ao invés de atualizar ou criar um novo registro. Porém, uma outra funcionalidade interessante que será implementada para as tabelas que não são *append-only* é a **deleção lógica ou virtual**. Uma deleção lógica nada mais é que tratar os eventos de deleção dos registros do banco de dados como apenas uma atualização qualquer, com a diferença de que inclui-se um campo booleano (verdadeiro ou falso) que dita se o registro está ou não está deletado, no lugar de fisicamente deletá-los da tabela na *silver zone*. Essa é uma ótima prática para manter registros históricos a longo prazo, para em caso de falha ou perda de dados no sistema transacional, o histórico de registros - até mesmo os deletados - se preserve, ou para repetir análises em registros deletados.

Essa funcionalidade de deleção virtual é implementada diretamente na lógica de execução da operação de *merge* realizada na *silver zone*, de tal forma que a árvore de decisão para casos de operação de deleção não exclua fisicamente o registro associado, apenas o marque como deletado.

Em suma, na *silver zone*, os seguintes processamentos são feitos para todas as tabelas:

1. Geração de um campo de PK único pois, nos sistemas SAP, especialmente para as 53 tabelas originalmente tratadas, todas possuem sua chave primária no modo composto, isto é, mais de um campo funciona como chave primária para identificar unicamente um registro. Assim, concatena-se, para cada registro de cada tabela, os campos que são integrantes de sua PK. Um exemplo disso é a tabela VBAK de cabeçalho de ordem, cuja PK é a concatenação dos campos MANDT e VBELN e;

2. Reordenação de eventos ocorridos para cada registro único, de forma a obter qual evento é o mais recente, e portanto, qual o estado atual de cada registro no sistema transacional. Isso garante que a *silver zone* será uma réplica exata do

SGBD Oracle. Essa é uma operação bastante custosa mesmo para a arquitetura distribuída implementada.

O primeiro processamento, na forma da geração da PK única é relativamente simples. Para cada tabela definimos uma lista de campos que compõem a sua PK e utilizamos um método nativo do PySpark, que é a API de comunicação que tem-se para executar comandos distribuídos em Spark usando Python chamado `concat_ws`, significando concatenação com separador. Assim, cria-se um novo campo `__PK_COMPOSITE` cujo valor é dado pela decomposição da lista, com cada elemento separado por um caractere padrão, no caso, o escolhido foi o *pipe operator* (|). Em PySpark, essa transformação é dada por `df_bronze = (df_bronze.withColumn('__PK_COMPOSITE', f.concat_ws('|', ['MANDT', 'VBELN']))` para uma tabela cujos campos de PK são a composição dos campos MANDT e VBELN. A distribuição desse comando para os *executors* pode ser feita de forma 100% paralela, sem necessidade de comunicação entre cada *executor*, visto que o resultado dessa computação em um *executor* não depende do resultado de nenhum outro *executor*, e portanto é uma operação relativamente rápida.

Já o segundo processamento é bem mais complexo. Assim como explicitado na Figura 14, a ordenação completa de um *dataframe* requer *shuffle*, pois a ordenação local em cada *executor* não garante que quando os resultados parciais forem unidos, o conjunto total estará ordenado. Ainda assim, a operação continua sendo possível, apenas é uma etapa mais demorada. O método de programação funcional disponibilizado em PySpark envolve a utilização de uma função janela de SQL. A expressão completa é dada por `df_bronze = df_bronze.withColumn("__ROWNUM", f.row_number().over(Window.partitionBy('__PK_COMPOSITE').orderBy('__TIMESTAMP_KAFKA').desc()))`. Essa expressão cria uma coluna nova `__ROWNUM` que, para cada valor de PK criado previamente, ordena os registros com base no seu tempo de captura de forma decrescente. Assim, para cada valor de PK, e portanto para cada registro único, o evento mais recente é colocado primeiro e recebe `__ROWNUM = 1`. O segundo mais recente recebe `__ROWNUM = 2` e assim por diante. Após isso, basta filtrar o *dataframe* resultante buscando apenas os registros com `__ROWNUM = 1`, este sendo o evento mais recente.

Por fim, pega-se esse *dataframe* e realiza a operação de *merge* previamente mencionada, inserindo todos os registros que não existiam previamente na tabela e atualizando todas as operações de *update* corretamente, lembrando também de tratar as operações de deleção como um mero *update* e marcando o registro em um campo específico `__IS_DELETED` como verdadeiro, assim fazendo a deleção lógica e atingindo o requisito F5 de entrega das tabelas *silver* descrito na Tabela 5.

De forma geral, um pseudocódigo para as operações de processamento para *silver* está abaixo:

```
# inicializar spark e suas dependencias
spark = start_and_configure_spark()

# pegar o maior timestamp de processamento da ultima execucao salvo em um arquivo
# de metadados como 'last_date.json' para esta tabela
LAST_EXECUTION_TIMESTAMP = read_last_execution_max_timestamp(SILVER_ZONE_PATH)

# ler dados da bronze zone em .parquet em formato incremental
bz_df = read_records_from_gcs(spark, BRONZE_ZONE_PATH, LAST_EXECUTION_TIMESTAMP)

# gera campo __PK_COMPOSITE como sendo concatenacao dos campos de PK composta
# como por exemplo 400|12345678 (MANDT|VBELN).
bz_df = bz_df.withColumn('__PK_COMPOSITE', f.concat_ws('|', ['col1'], ['col2']))

# realiza ordenacao por timestamp de captura para cada registro
bz_df = (bz_df.withColumn("__ROWNUM", f.row_number().over(Window
    .partitionBy('__PK_COMPOSITE').orderBy(f.col('__TIMESTAMP_KAFKA').desc()))))
# filtra para pegar apenas __ROWNUM == 1 (apenas o evento mais recente)
bz_df = bz_df.where("__ROWNUM == 1")
# remove coluna __ROWNUM que agora e inutil
bz_df_processado = bz_df.drop('__ROWNUM')

# salvar dados processados na camada silver com operacao merge, algo como:
# fazer merge do dataframe de updated em cima da origem, quando for operacao
# r(ead), d(elete) ou u(pdate) atualiza todos os campos (pode-se usar o campo
# de operacao do proprio kafka para definir a delecao logica)
# se for c(reate) insere normalmente

# assim a funcao abaixo merge_into implementa um SQL do tipo:
# MERGE INTO orig USING upd ON (orig."__PK_COMPOSITE" = upd."__PK_COMPOSITE")
# WHEN MATCHED AND (upd.{op_column_name} = "u"
# OR upd.{op_column_name} = "r"
# OR upd.{op_column_name} = "d") THEN
# UPDATE SET *
# WHEN NOT MATCHED THEN
# INSERT *;
merge_into(spark, SILVER_ZONE_PATH, bz_df_processado)

# pega o maior timestamp da execucao atual a ser salvo no novo 'last_date.json'
CURRENT_EXECUTION_MAX_TIMESTAMP = bz_df_processado.orderby(timestamp, 'DESC')[0]

# salvar um novo arquivo de metadados como last_date.json com novo valor
update_last_timestamp_metadata(CURRENT_EXECUTION_MAX_TIMESTAMP)
```

5.2.4 Visualização do DAG responsável pelo agendamento de tarefas na interface do Airflow para *bronze* e *silver*

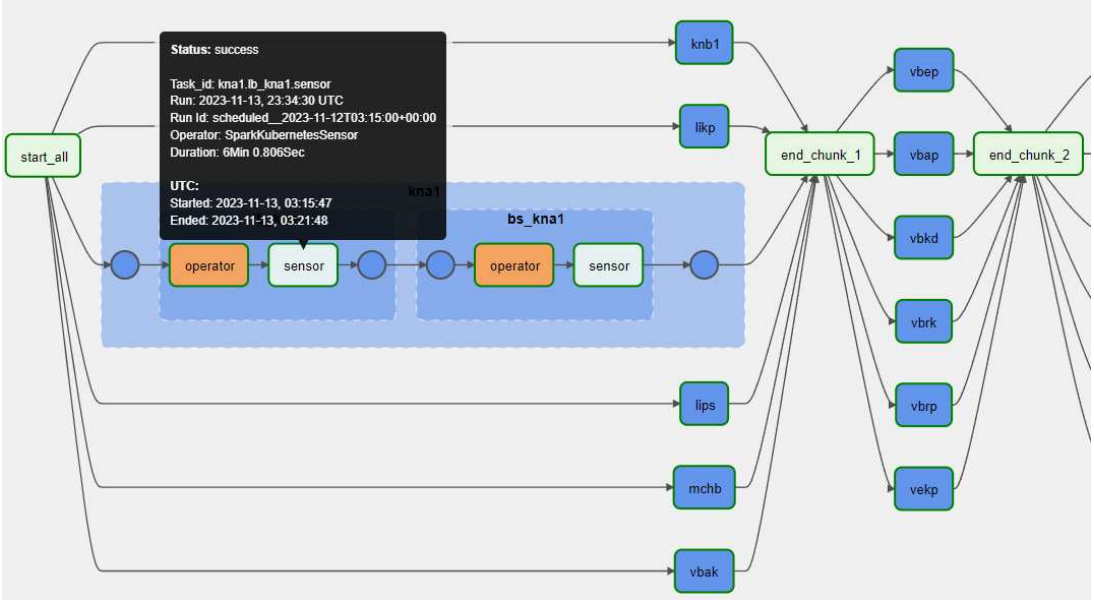
Com os processamentos *LB* (*landing* para *bronze*) e *BS* (*bronze* para *silver*) prontos, pode-se montar o primeiro DAG. Como esses processamentos são elementalmente sequenciais (ou seja, depois de um evento capturado e colocado na *landing* é processado para a *bronze*, ele pode imediatamente ser processado também para a *silver*), então o DAG deve sequenciar, para cada tabela processada, duas tarefas:

uma para realizar a etapa *LB*, e outra para a etapa *BS*. A Figura 31, que demonstra um DAG de processamento concorrente de seis tabelas de forma incremental, passando pelas etapas em sequência. Em destaque, algumas métricas do próprio Airflow sobre a execução da etapa *landing* para *bronze* da tabela KNA1 (cadastro de clientes no SAP), com incremental levando cerca de seis minutos para completar na última execução. Na forma apresentada, esse DAG dispara o processamento simultâneo das seis tabelas em destaque (KNB1, LIKP, KNA1, ...), e assim que todas as etapas dessas tabelas (*LB* e *BS*) estão completas, então a tarefa `end_chunk_1` é marcada como sucesso, e as próximas seis tabelas serão processadas (VBEP, VBAP, ...). Essa abordagem é feita para não executar todas as 53 tabelas em paralelo de uma vez, certamente utilizando mais recursos que o *cluster*, mesmo com vinte nós, possui.

Percebe-se também que existem, na verdade, duas tarefas para cada etapa de cada tabela, uma denominada *operator* e uma *sensor*. Essa distinção é feita apenas porque na realidade quando instancia-se um *pod* do Kubernetes via Airflow utilizando o `SparkKubernetesOperator`, essa tarefa é quase imediata (criação do *pod*), sendo necessária a instanciação de um *sensor* que "observa" e aguarda o *pod* terminar seu processamento. Isso tudo é feito de forma transparente ao usuário.

Esse DAG está sendo executado, em notação *crontab*, por `30 2 * * *`, isto é, todos os dias às 2h30 da madrugada em Brasília e leva, em sua totalidade e no modelo incremental cerca de 90 minutos para completar todas as tabelas, visto que algumas tabelas possuem, mesmo no incremental, um alto volume de dados. Como discutido previamente, é custoso operacionalizar um *merge* de altos volumes em cima de altos volumes pois, para cada registro (que são muitos) precisa-se tentar encontrá-lo na tabela atual (que é grande) para decidir qual operação fazer (inserção, atualização ou marcação lógica como deletado).

Figura 31 – DAG em produção que realiza o processamento



Fonte: Acervo do autor.

5.3 PROCESSAMENTO DAS TABELAS DA *GOLD ZONE*

O desenvolvimento das rotinas de processamento para a geração das tabelas analíticas fato e dimensão para a *gold zone* seguiu um padrão de implementação semelhante à de camadas anteriores, com a clara distinção de objetivos desejados com o uso e análise dos dados dessas tabelas em comparação à *silver* e *bronze*. Assim, existe um segundo DAG responsável apenas pela geração destas tabelas, e este DAG aguarda, a cada dia, que o DAG responsável pelos processamentos *LB* e *BS* terminem com sucesso. Assim que isso ocorre, disparam-se as tarefas para geração das tabelas *gold*, sem que haja um *crontab* definido visto que depende da execução do DAG anterior.

Existem quatro grandes diferenças na geração das tabelas *golds* em relação às outras camadas:

- Uma *gold* geralmente consulta de várias tabelas da *silver* e até mesmo outras *golds*, ao invés de consultar apenas a tabela referente da camada anterior. Assim, para gerar uma *gold*, pode ser necessário ler diversas *silver* e *golds* e salvá-las em memória ou disco (caso não caiba em memória) nos *executors*;
- Cada *gold* é única e possui um *script* em Python com instruções em PySpark únicas, ou seja, não há forma de reaproveitar ou generalizar *scripts* como é feito nas camadas anteriores e;

- Não se usam instruções nativas do PySpark usando *method chaining* (métodos como os apresentados previamente como `.union()`, `.partitionBy()` ou ainda `.withColumn()`, mas sim utiliza-se apenas do método `df.sql(<comandos sql>)` por motivos de que é muito mais simples definir regras de negócio, cálculos, agregações e diversas outras operações diretamente em SQL nativo. PySpark é capaz de ler SQL diretamente e converter nas sequências de comandos necessários para paralelizar e distribuir as operações entre os *executors*.
- Por questões de qualidade de dados e para evitar duplicações, nenhuma das tabelas *gold* são incrementais. Todos os dias, quando a tarefa referentes à uma *gold* é executada, ocorre uma operação de *overwrite* por sobre a tabela anterior. Ainda assim, é possível consultar versões anteriores de uma *gold* graças ao versionamento dos arquivos que compõem a tabela da ferramenta *Delta Lake*.

Por questões de confidencialidade, não é possível explicitar com muitos detalhes nenhum dos *pipelines* de processamento de tabelas *gold*, visto que elas contém muitas regras de negócio e particularidades do cliente. Todavia, um pseudocódigo que bem representa os passos, em PySpark na geração de uma tabela como a fato de ordens de venda é:

```
# inicializar spark e suas dependencias
spark = start_and_configure_spark()

# ler dados da silver zone em .parquet de algumas tabelas
vbak_df = read_records_from_gcs(spark, VBAK_SILVER_ZONE_PATH)
vbap_df = read_records_from_gcs(spark, VBAP_SILVER_ZONE_PATH)
faturas_df = read_records_from_gcs(spark, FATURAS_GOLD_ZONE_PATH)

# executa comando diretamente em sql para geracao da gold fato ordens de venda
fato_ov = spark.sql("""
WITH CTE_FATO AS (SELECT DISTINCT
    CAST(VBAK.ERDAT AS Date) AS Data_Criacao_OV,
    CAST(VBAP.VDATU AS Date) AS Data_Embarque_Pedido,
    CAST(FT.Data_Faturamento AS Date) AS Data_Faturamento,
    ...,
    CASE
        WHEN VBAK.VKORG = 'MI' THEN 'Mercado Interno'
        ELSE 'Mercado Externo'
    END AS Organizacao_Vendas
FROM vbak_df AS VBAK
LEFT JOIN vbap_df AS VBAP ON (VBAK.__PK_COMPOSITE = VBAP.__PK_COMPOSITE)
LEFT JOIN faturas_df AS FT ON (CONCAT(VBAK.VBELN, '|', VBAK.POSNR = FT.Chv_Fatura)
""")

overwrite_data_to_gcs_gold(spark, FATO_OV_GOLD_PATH, fato_ov)
```

Esse processo é repetido, caso a caso, com bastante esforço para cada tabela *gold*, com o intuito de cumprir o requisito F6 referente à entrega das tabelas *gold* da Tabela 5. A dificuldade da geração dessas tabelas está, além do volume, na quantidade

e complexidade das regras de negócio envolvidas. Existem muitos filtros arbitrários, campos desconhecidos até mesmo para a equipe interna responsável pelo ERP SAP, condicionais específicos e outras especificidades de negócio que não podem ser mencionadas a serem levadas em consideração. Grande parte das *golds* foi entregue em uma versão inicial, validada pelos atores responsáveis, retrabalhadas, corrigidas, entregues novamente, e assim se deu um ciclo repetitivo de entregas e validações. Algumas entregas demoraram mais de dez ciclos de validações até que pudessem ser consideradas prontas.

5.4 DIMENSIONAMENTO DAS SPARK *APPLICATIONS*

Parte do desafio na otimização de custos e eficiência da arquitetura está no bom dimensionamento das aplicações do Spark que executam os processos de transformação, seja da *landing* para *bronze*, *bronze* para *silver* ou *silver* para *gold*. Não existe receita ou padrão ótimo no dimensionamento das *applications* pois o comportamento (uso de memória, vCPU) e tempos de processamento variam muito a depender do conjunto de dados tratado e da chave de particionamento usada.

Como o Spark internamente distribui tarefas para os *executors* fazendo repartições e distribuindo essas fatias do conjunto de dados total, cada conjunto pode ser distribuído de diversas maneiras. O ideal é fazer com que cada *executor* tenha um número semelhante de registros em seu domínio, para que, de forma geral, todos terminem de realizar os cálculos e processamentos ao mesmo tempo ou perto disso.

Se uma aplicação distribui o conjunto entre dois *executors*, mas coloca 95% dos dados em um *executor 1* e os 5% restantes em um *executor 2*, e ambos possuem a mesma velocidade de clock, memória e demais variáveis idênticas, então no momento em que o segundo *executor* tiver terminado suas tarefas, o primeiro ainda estará trabalhando em cima de 90% do conjunto de dados. Esse fenômeno é chamado, na computação distribuída, de *data skew*, e faz com que algumas tarefas possam demorar até centenas de vezes mais que o esperado quando um *executor* recebe muito mais "trabalho" que outro.

Assim, a nível de cada tabela processada, o engenheiro de dados deve avaliar algumas características básicas do conjunto de dados que compõe tal tabela, entre elas:

1. O volume total da tabela, em número de registros, para entender a ordem de grandeza trabalhada;

2. O volume total em fator de tamanho em disco e memória da tabela para saber bem dimensionar a quantidade de *executors* e suas configurações como memória e número de vCPUs para finalizar o processamento em tempo hábil;

3. Características de distribuição estatística das colunas de uma tabela, com o intuito de informar à motorização do Spark quais são bons campos para repartir o conjunto de dados. A forma em que isso é implementada internamente em Spark consiste na execução de uma função de *hash* em cima dos valores de uma ou mais colunas, e cada *hash* único será enviado para um único *executor*, mas um mesmo *executor* pode receber mais de um *hash*. Assim, se um conjunto de dados possui 10 bilhões de registros e uma coluna booleana que pode ser *true* ou *false*, se 9 bilhões dos registros possuem valor *true* e 1 bilhão possuem valor *false*, caso o engenheiro de dados tente repartir esse conjunto por essa coluna em uma aplicação com dois *executors*, um deles receberia 90% dos dados enquanto o outro apenas 10%, o que é extremamente ineficiente. A solução aqui seria buscar um outro campo que bem distribua o conjunto entre todos os *executors*.

Isso requer um estudo analítico forte sobre as particularidades de cada conjunto de dados envolvendo uma análise exploratória. Ao longo do desenvolvimento do projeto, para otimizar o dimensionamento das aplicações, as cargas iniciais foram feitas com *executors* sobre-dimensionados (algumas rotinas utilizavam 20 *executors* com 32GB de memória e 4 vCPUs cada, totalizando 640GB e 80 núcleos) para uma primeira carga bem sucedida, e a partir desse ponto, utilizava-se o BigQuery para selecionar colunas específicas e a contagem de registros para cada valor único dessa coluna com o intuito de encontrar bons campos de particionamento e melhor dimensionar os *executors* a partir dessa avaliação. O bom dimensionamento dessas *applications* completa o requisito NF5 da Tabela 6.

5.5 CAMADA DE DISPONIBILIZAÇÃO COM O GOOGLE CLOUD BIGQUERY

Aproveitando-se do ecossistema disponibilizado na GCP, a ferramenta Google Cloud BigQuery é uma solução de ponta a ponta para consulta analítica de dados, oferecendo serviços de armazenamento e processamento com tecnologia OLAP em um único produto. Uma grande desvantagem de utilizar-se esse tipo de produto no lugar de uma arquitetura como a implementada no projeto é a perda das capacidades que a junção de Apache Spark e Delta Lake proporcionam (*timetravel*, controle sobre dados e infraestrutura) e principalmente o fator preço, visto que o mau uso do BigQuery pode custar muito caro, e portanto, deseja-se deixar o processamento pesado para a infraestrutura, enquanto no BigQuery limita-se à consultas e análises em cima de dados previamente processados.

Assim, é possível utilizar-se da ferramenta sem necessariamente realizar o processamento pesado nela ou sequer o armazenamento embutido que a ferramenta proporciona. No BigQuery, é possível criar uma entidade chamada **tabela externa** que, quando configurada, é capaz de ler todos os arquivos de um determinado *path*

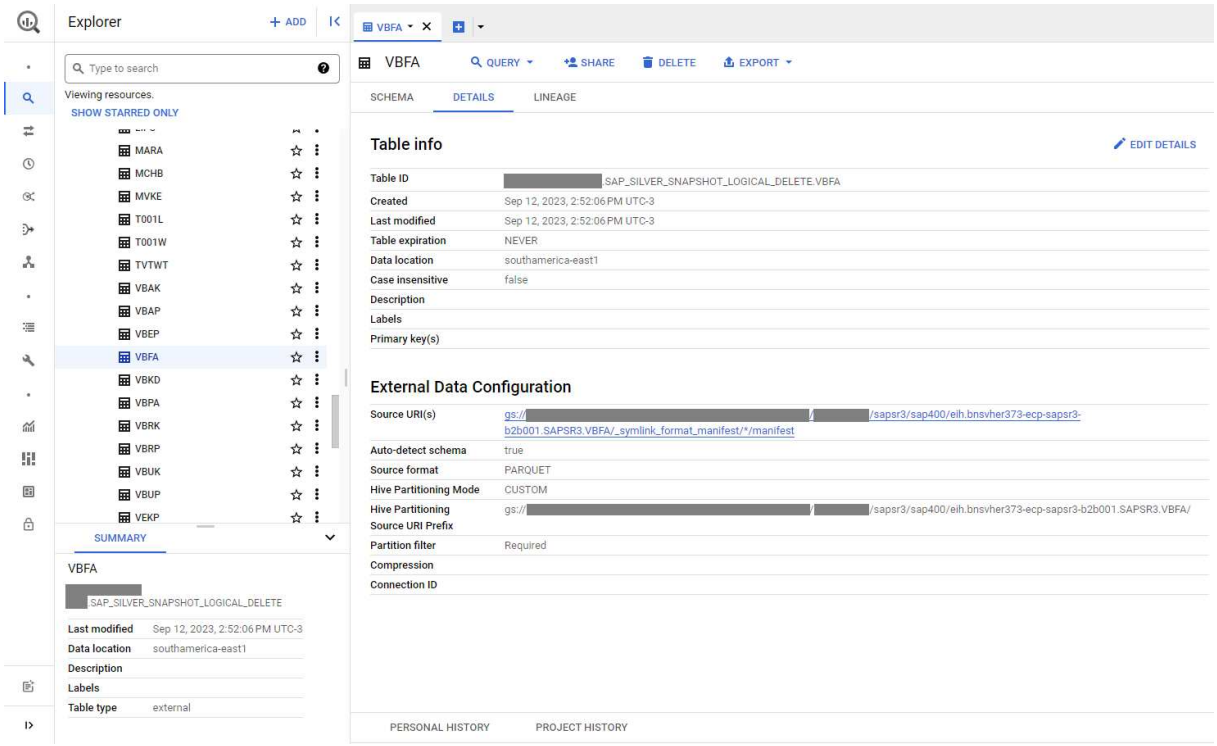
em alguns repositórios, incluindo a camada de armazenamento do DW na forma do GCS. Uma limitação do BigQuery, entretanto, é que embora planejada, no momento da escrita e do desenvolvimento do projeto, não há integração nativa para leitura de caminhos escritos utilizando a ferramenta Delta Lake. Isto é, uma tabela externa do BigQuery não saberia interpretar os `delta_logs/` e ler apenas os arquivos relevantes para a versão mais atual da tabela, conforme explicitado previamente na Figura 15.

A solução para esse problema de integração está na implementação de uma rotina de geração de um arquivo chamado **Symlink Manifest**, uma funcionalidade padrão da biblioteca Python do Delta Lake em conjunto com Spark que gera, a partir dos `delta_logs/` (que o Spark sabe interpretar corretamente), um novo arquivo textual que contém os caminhos de todos os arquivos `.parquet` de dados que compõem a versão atual da tabela. Basicamente, é uma forma de terceirizar a interpretação do formato Delta Lake para o Spark, e gerar uma lista de arquivos a serem lidos.

Com essa lista de arquivos em mãos, basta criar uma tabela externa do BigQuery com as instruções em pseudocódigo como: “leia todos os arquivos da pasta *k* do GCS cujos nomes estão no arquivo *z*, sendo *z* o arquivo gerado pelo Symlink Manifest”. Assim, qualquer usuário autenticado na plataforma da GCP e que tenha as devidas permissões configuradas na própria plataforma (usando o gerenciador de acessos nativos da GCP) poderá consultar e realizar operações em cima dos dados contidos na camada de persistência no GCS. Outra vantagem dessa abordagem é que não é possível modificar os arquivos originários da tabela de forma direta, ou seja, esta é uma forma de democratizar o acesso à informação sem que seja possível que um usuário quebre ou corrompa os dados contidos na camada de armazenamento, pela natureza externa das *external tables*.

Na Figura 32, um exemplo da interface gráfica do BigQuery demonstrando alguns parâmetros da criação da tabela externa na *silver zone* para a tabela VBFA que remete ao fluxo dos documentos de venda (etapas que um documento passa ao longo de sua existência como pendente, ativo, emitido, cancelado, etc.).

Figura 32 – Interface do BigQuery na aba de detalhes de uma tabela externa criada.



Fonte: Acervo do autor.

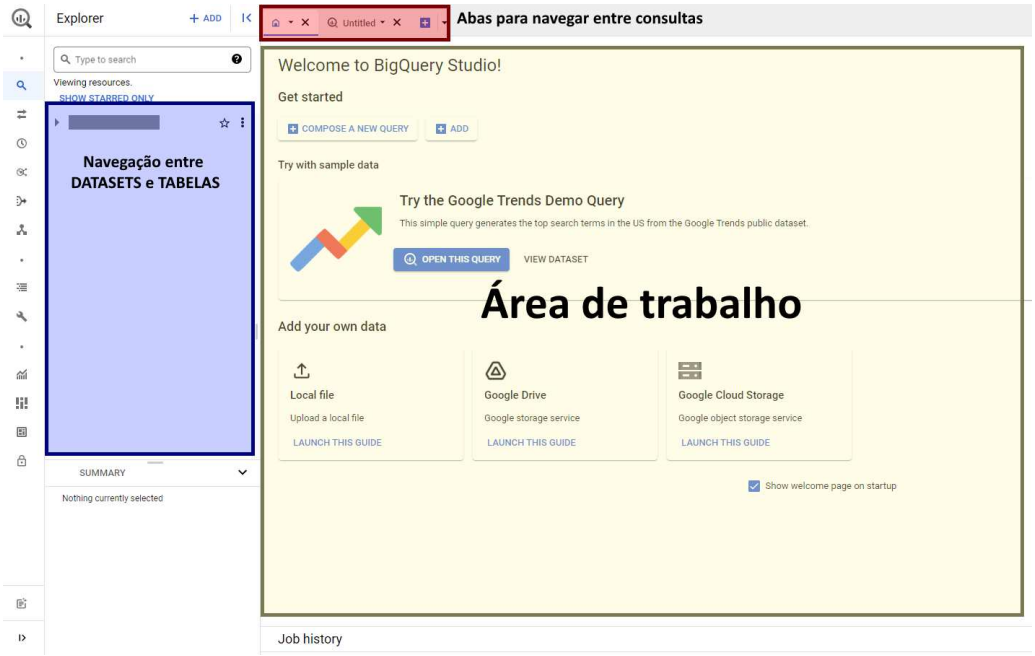
5.6 INTERFACE DE USUÁRIO E OPERACIONALIZAÇÃO DO DW PARA O CLIENTE

Tendo as tabelas processadas através de todas as camadas da arquitetura, com incrementais ligados para tabelas *bronze* e *silver*, fluxos bem definidos e entregas validadas na *gold*, e o BigQuery corretamente configurado para ler as tabelas externas, falta entender como é a interação dos usuários da plataforma de consulta com os dados propriamente ditos que são entregues e atualizados diariamente.

De acordo com o apresentado, o entregável para o projeto nada mais é que uma interface de interação que permite que usuários façam consultas aos dados do repositório no GCS. Essa interface é disponível para todos que acessarem a plataforma da GCP (GOOGLE CLOUD PLATFORM, 2023b) e estiverem suficientemente autenticados com suas contas no **projeto** (nomenclatura da GCP para organizar e faturar seus produtos) da empresa cliente, e que seu usuário tenha as devidas permissões nesse projeto para acessar o serviço BigQuery. Todo o gerenciamento de acesso é feito diretamente pela equipe de TI da empresa utilizando as ferramentas de Identity and Access Management (IAM) da GCP.

Quem visita o BigQuery pela primeira vez é recebido com uma interface amigável, separada em alguns quadrantes, conforme explicita a Figura 33

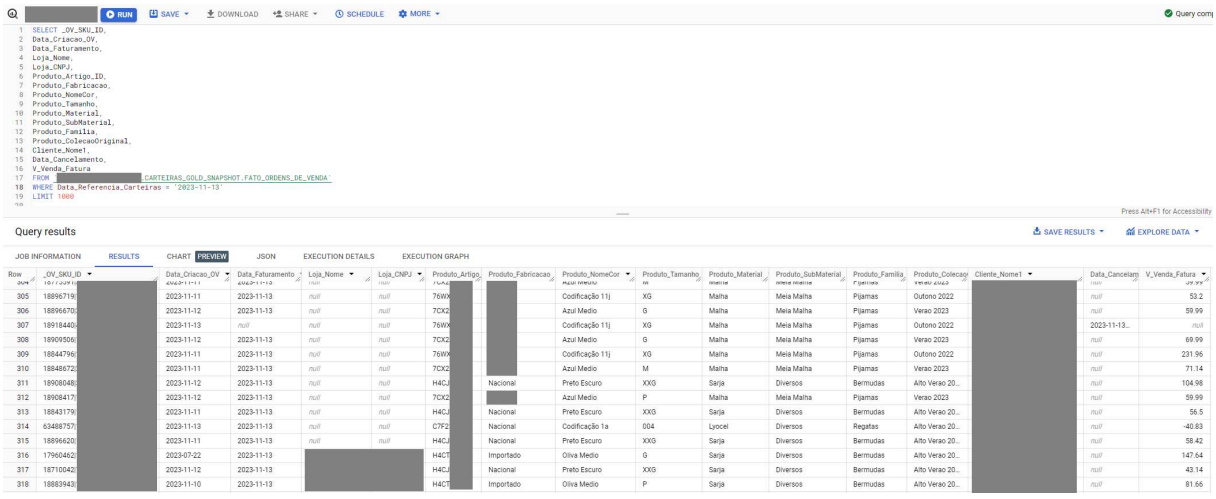
Figura 33 – Interface do BigQuery.



Fonte: Acervo do autor.

Nessa interface, como já fora mostrado previamente neste mesmo capítulo um usuário pode navegar livremente pelas tabelas externas criadas pelo autor e realizar consultas em SQL a qualquer momento. Um exemplo de consulta genérica que pode ser feita com a implementação do novo DW está apresentada na Figura 34:

Figura 34 – Consulta genérica na base de ordens de venda com informações de valores e produtos.



Fonte: Acervo do autor.

Por fim, o usuário é capaz de realizar consultas o quão complexas desejar na tes-

tagem de hipóteses e obtenção de informações críticas de faturamento. Na Figura 35, a consulta apresentada retorna o faturamento diário obtido para vendas realizadas no dia 13 de Novembro de 2023 apenas na categoria de pijamas de cor azul médio. Podem ser feitas diversas consultas adicionais, e até mesmo acompanhar a evolução histórica diária de peças específicas e associar seu desempenho com a época do ano, coleções, períodos de promoção, festas natalinas ou quaisquer outras hipóteses que precisem ser validadas. A remoção completa do atrito entre a disponibilidade dos dados e o desejo de consulta permite liberdade total dos analistas na tomada de decisão,.

Figura 35 – Consulta de validação do desempenho diário de um produto específico.

Fonte: Acervo do autor.

5.7 DEMAIS REQUISITOS ATINGIDOS

Os demais requisitos funcionais e não funcionais não mencionados, nomeadamente F1, F2, F3, F7, F8, F9 (Tabela 5) e NF2, NF3 e NF4 (Tabela 6) são consequência das escolhas de ferramentas e arquitetura implementada. Sendo assim, o requisito F1 fora atingido por outra equipe de consultoria responsável pela ingestão, enquanto os requisitos F2, F3 e F7 são derivados das ferramentas Spark, GCS e Delta Lake utilizadas, e portanto são atingidos no sucesso de sua implantação. O próprio Delta Lake e GCS também dão suporte ao versionamento de tabelas, permitindo que sejam criadas novas versões sem que se afete a manutenção do serviço para o usuário final, cumprindo o requisito F8. Para finalizar os requisitos funcionais, F9 é cumprido ao realizar a entrega final de todas as tabelas *gold* do escopo inicial.

Quanto aos requisitos não funcionais, NF2 é garantido ao utilizar o sistema

de auto-escalabilidade do GKE, NF3 é providenciado pela própria plataforma com gerenciamento de acesso da GCP, o IAM, e o último requisito a ser tratado, NF4, é atingido com as rotinas de *logs* e coleta de métricas implementadas, que serão discutidas no Capítulo 6, mas que também é atingido.

Com isso, todos os requisitos especificados no projeto foram efetivamente cumpridos com a entrega do mesmo.

6 ANÁLISE DE RESULTADOS

Este capítulo é dedicado à análise de resultados obtidos com a implementação do projeto. Serão abordadas duas frentes distintas: uma voltada à avaliação técnica, envolvendo tempos de execução, dificuldade técnica e métricas, e uma segunda envolvendo o alcance do projeto dentro do cliente, impactos financeiros e ganhos de eficiência de negócio.

6.1 AVALIAÇÃO TÉCNICA

Nesta seção serão avaliados e discutidos alguns pontos e métricas relevantes na avaliação do desempenho dos *pipelines* do DW.

6.1.1 Volumes e tempos de processamento

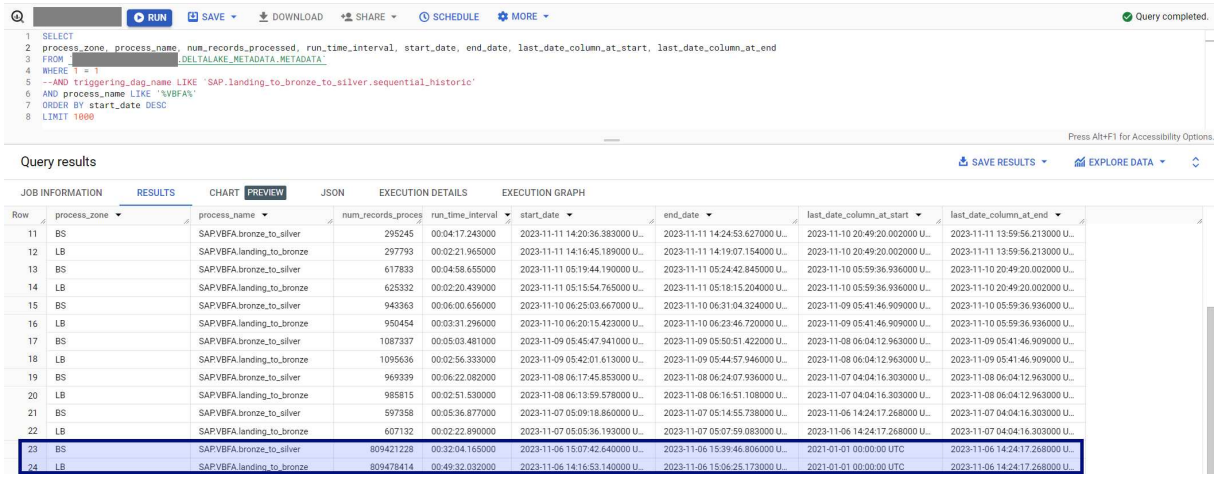
Como requisito não funcional (NF4) da Tabela 6, os sistemas de processamento devem registrar *logs* de execução contendo tempos e quantidade de registros processada. Essa coleta de métricas é importante para avaliar o desempenho das *applications* e acompanhar a evolução das bases de dados de forma a otimizar o custo total do DW, visto que, se subutilizado, o *cluster* pode escalar para baixo desligando e deixando de faturar nós do GKE. Assim, foi implementada uma rotina que ao longo da execução de cada etapa de processamento de cada tabela, são cadastrados em uma tabela externa adicional (em um GCS *bucket* de metadados a ser lido como tabela externa).

Dentre os dados guardados para cada execução, tem-se: a qual tipo de processamento a execução se refere (*landing* para *bronze*, *bronze* para *silver*, etc.), qual DAG disparou a execução, as colunas de última data usadas filtragem para a carga incremental (tanto a maior data de captura no início do processo e a final, que se tornará parâmetro de filtro para a próxima execução), data de início e fim do processamento no horário de Brasília (assim permitindo calcular a diferença, totalizando no tempo total gasto), *tags* de filtragem e o volume total de registros tratados por execução.

A Figura 36 retrata uma consulta realizada no próprio BigQuery buscando informações na tabela de métricas sobre execuções de processamento sobre uma tabela VBFA do SAP. Destacado em azul estão execuções da *landing* para *bronze* e também *bronze* para *silver* que foram cargas completas da tabela, feitas no início de Novembro de 2023 por motivos de reingestão completa da tabela por uma dessincronização ocorrida no Debezium, responsabilidade de uma outra equipe de consultoria. É possível verificar que foram processados cerca de 800 milhões de registros em 49 e 32 minutos para *LB* e *BS*, respectivamente. Além disso, todas as outras execuções mostradas retratam volumes incrementais diários que variam bastante (muito por conta de finais de semana e feriados), mas usualmente na faixa de 300 a 900 mil registros diários em

5 minutos ou menos.

Figura 36 – Interface do BigQuery com consulta à métricas de execução da tabela VBFA.

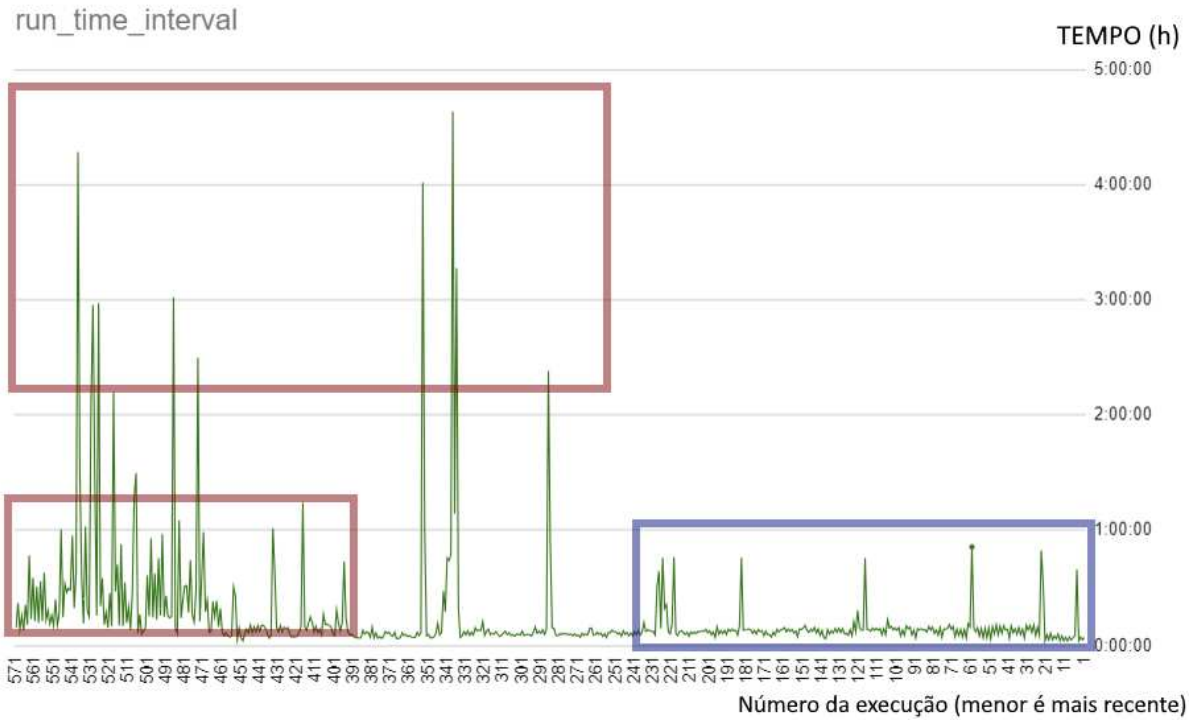


Fonte: Acervo do autor.

Esses valores se mostram muito próximos dos praticados no mercado. De acordo com a empresa *Databricks*, fornecedora de serviços em nuvem para computação, utilizando as configurações padrão do *Databricks Runtime* o esperado para uma operação distribuída como o *merge* (operação aplicada nos *pipelines* deste projeto) devem estar em torno de 6 a 9 minutos para 27 mil registros de atualização sobre um conjunto total de 1.5 bilhões de registros (DATABRICKS, 2022). A escalabilidade de tempo com relação aos volumes envolvidos não é linear e é função tanto do volume incremental, quanto do volume total, mas considerando que estão sendo tratados de 300 a 900 mil registros (muito mais se comparados aos 27 mil do artigo) sobre 800 milhões (pouco mais da metade do artigo referenciado), então os resultados estão satisfatórios para o cliente, visto que entrega as bases atualizadas em tempo hábil e com custos aceitáveis.

Outra informação interessante que pode-se observar é a evolução do desenvolvimento do projeto. Exportando essa tabela da Figura 36 e desenhando um gráfico ordenado temporalmente (Figura 37) (no caso, o eixo horizontal está quantificado em “quantidades de execuções atrás”, então um valor de 50 significa 50 execuções atrás). Sendo assim é possível perceber (com destaque em vermelho na Figura 37) que bem no começo do desenvolvimento do projeto os tempos de execução para essa tabela VBFA eram inconsistentes e existiam picos muito grandes de até 5 horas de duração. Ficando claro (em azul) que conforme o projeto ganhou maturidade e as otimizações nos *pipelines* foram sendo feitas, os tempos estabilizaram na casa de 5 minutos para incremental, com picos de até quase 1 hora para cargas completas.

Figura 37 – Evolução temporal do tempo de execução dos pipelines para tabela VBFA.

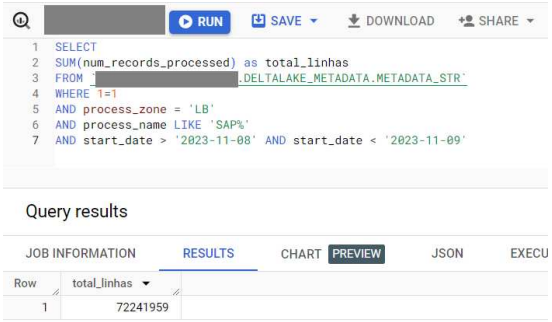


Fonte: Acervo do autor.

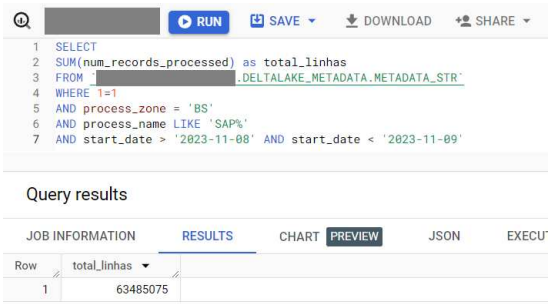
Em valores absolutos, somando-se o volume de processamento de todas as tabelas em um período de um dia inteiro, pode-se obter o volume incremental diário processado pela arquitetura inteira. Utilizando como base outra consulta no BigQuery com filtro para uma data específica, tem-se como resultado os apresentados na Figura 38, com os quais podemos afirmar que no processamento incremental, isto é, *landing* até *silver*, diariamente foram tratados um total de 72 milhões de registros na etapa *LB*, e 63 milhões de registros na etapa *BS*, totalizando mais de 130 milhões de registros processadas em um único dia.

A etapa *SG* (*silver* para *gold*) não é feita de forma incremental - assunto tratado na Seção 5.3 - e por isso o volume aqui é bem alto. Toda a camada *gold* é recriada diariamente com um volume total tratado de quase 1.2 bilhões de registros.

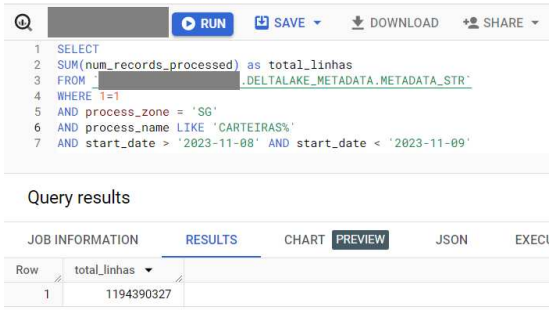
Figura 38 – Volumes tratados entre dias 8 e 9 de Novembro de 2023.



(a) Volume diário processado na etapa *LB*.



(b) Volume diário processado na etapa *BS*.



(c) Volume diário processado na etapa *SG*.

Fonte: Acervo do autor.

6.1.2 Custo em regime permanente

Considerando o pior dos casos, em que todos os vinte nós estejam sendo faturados vinte e quatro horas por dia ao longo de um mês, o custo esperado para essa arquitetura é de US$ 12.200 mensais, considerando todas as ferramentas hospedadas no GKE. Porém, como previamente mencionado, em regime permanente não estão sendo faturados todos os vinte nós, mantendo-se uma média ao longo do dia de oito a dezesseis nós, o que corta o custo mensal (assumindo uma média de doze nós operacionais a todo momento) para US$ 7.320 mensais.

Ainda existe algum custo para o armazenamento do serviço GCS, no qual o armazenamento de 1TB de arquivos custa - no momento da escrita deste relatório - US$ 20 dólares mensais, e atualmente o armazenamento é de cerca de 53TB entre todas as tabelas de todas as camadas da arquitetura medalhão implementada, somando ainda algumas tabelas de outros projetos. A estimativa isolada para apenas as tabelas referentes ao projeto em questão é de cerca de 35TB, o que gera um adicional de US$ 700 de custo.

Existe, também, custos associados ao processamento feito no BigQuery, que, embora leve (pois não faz todo o trabalho pesado de ETL, visto que isso é feito pelo Spark no GKE), ainda é usado pelos analistas e executivos na descoberta de dados, avaliações, exportações de dados, geração de relatórios automatizados, consumo de

outra aplicações via API e qualquer outro tipo de consulta. Uma média de uso dos últimos três meses de projeto coloca o BigQuery com um custo mensal de cerca de US$ 600.

Na totalidade, esses três grandes serviços que compõem toda as camadas do DW custa em torno de US$8.620 mensais, o que em uma conversão no momento de escrita resulta em cerca de R$42.000 mensais. Esse valor parece bastante alto, mas considerando o tamanho da empresa cliente, isso representa uma parcela muito pequena do faturamento total anual, e possui potencial de ganho muito maior que seu custo, considerando que boas decisões de negócio agora podem ser tomadas com base em fatos reais coletados e agregados no DW.

6.2 IMPACTO NO CLIENTE

Nesta seção serão apresentados e/ou parafraseados alguns resultados que o cliente compartilhou com a BIX Tech e o autor.

6.2.1 Agilização de consultas

Uma das maiores dores no quesito consultas e análises *ad hoc* ou outras avaliações a serem feitas sob demanda era o grande intervalo de tempo que o requisitante precisava esperar para sequer conseguir os dados necessários para a consulta. O time interno de analistas agora pode visitar a interface *front-end web* do BigQuery e realizar toda e qualquer consulta necessária, seja nas tabelas réplica da *silver* para buscar algum registro específico, seja na *bronze* para verificar o histórico de um pedido, ou seja na *gold* buscando informações resumidas e pré-processadas.

Não se pode dizer que tempo de execução para esse tipo de análise caiu de até duas semanas para alguns segundos pois a análise em si é um processo que não é automatizável e requer que o analista busque e descubra *insights* a partir dos dados, porém o atrito entre times e o intervalo entre um analista pensar em fazer uma análise e poder de fato atuar e trabalhar em cima de uma hipótese caiu, de fato, de duas semanas para alguns segundos.

Cumulativamente, dentre toda uma equipe de analistas de faturamento do setor B2B, composto por um coordenador e duas analistas são economizadas semanalmente cerca de 20 horas de trabalho antes manual (envolvendo abertura de chamado, requisição de um excerto de dados, intervalo até o recebimento, limpeza e caracterização dos dados) apenas neste núcleo. Cada hora economizada de um time hierarquicamente alto em uma empresa são potenciais milhões de reais em faturamento que podem ser acumulados com uma decisão rápida e correta.

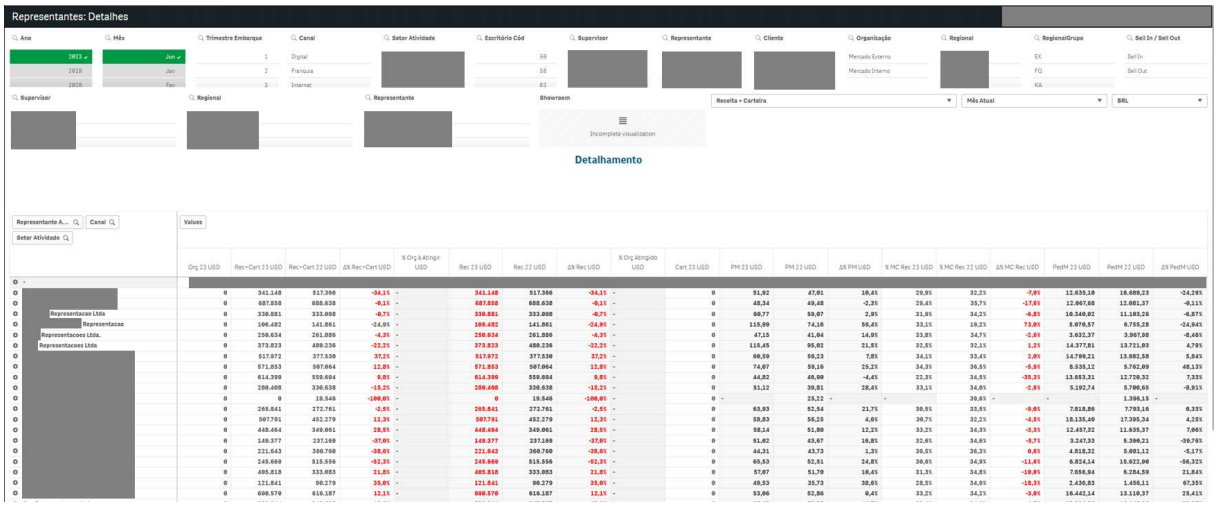
6.2.2 Populando *dashboards* com dados

Além da agilização das consultas, alguns *dashboards* internos puderam ser montados e populados. Dois times distintos utilizavam duas ferramentas de montagem de painéis de BI (QlikSense e PowerBI) que eram populados às vezes semanalmente, às vezes mensalmente de forma manual com excertos de dados exportados por outros times e entregues aos analistas. Um dos analistas relatou que por vezes esses painéis sequer eram consultados e o conhecimento potencialmente contido neles era desprezado simplesmente por falta de mão-de-obra para fazer a extração, limpeza e população do *dashboard* com dados.

Algumas visualizações foram compartilhadas enquanto ainda estavam em uma etapa de desenvolvimento e estão representadas e descritas a seguir. Alguns trechos foram completamente censurados por questões de anonimidade.

Na Figura 39, um painel de inteligência de negócios abrindo alguns detalhamentos por representante de vendas, contendo informações de orçamento para cada representante e valores que metrificam seu desempenho de carteira e outros indicadores envolvendo a variação de pedidos e efetivações ao longo de anos.

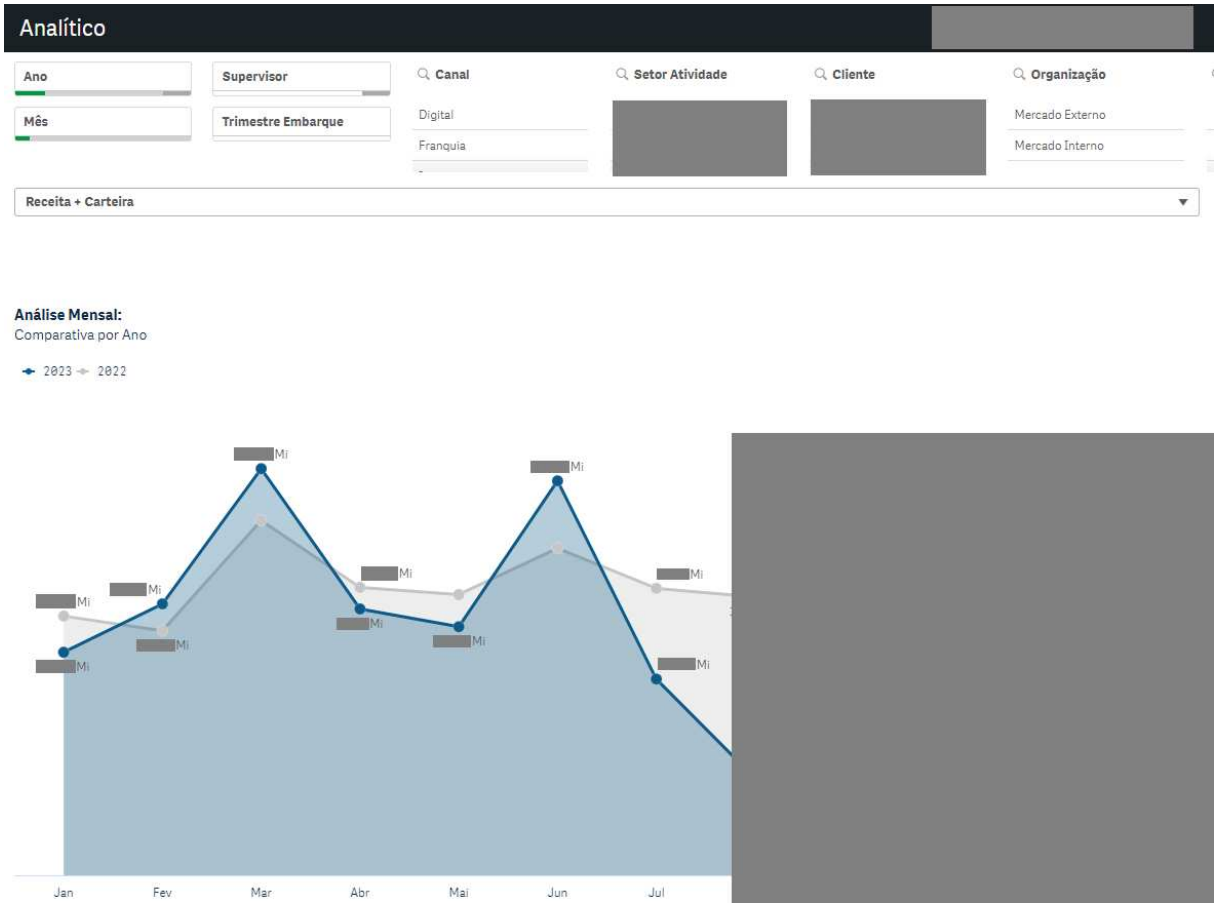
Figura 39 – Painel de representantes.



Fonte: Acervo do autor.

Na Figura 40, um painel com detalhamentos de faturamento com comparativo de crescimento anual:

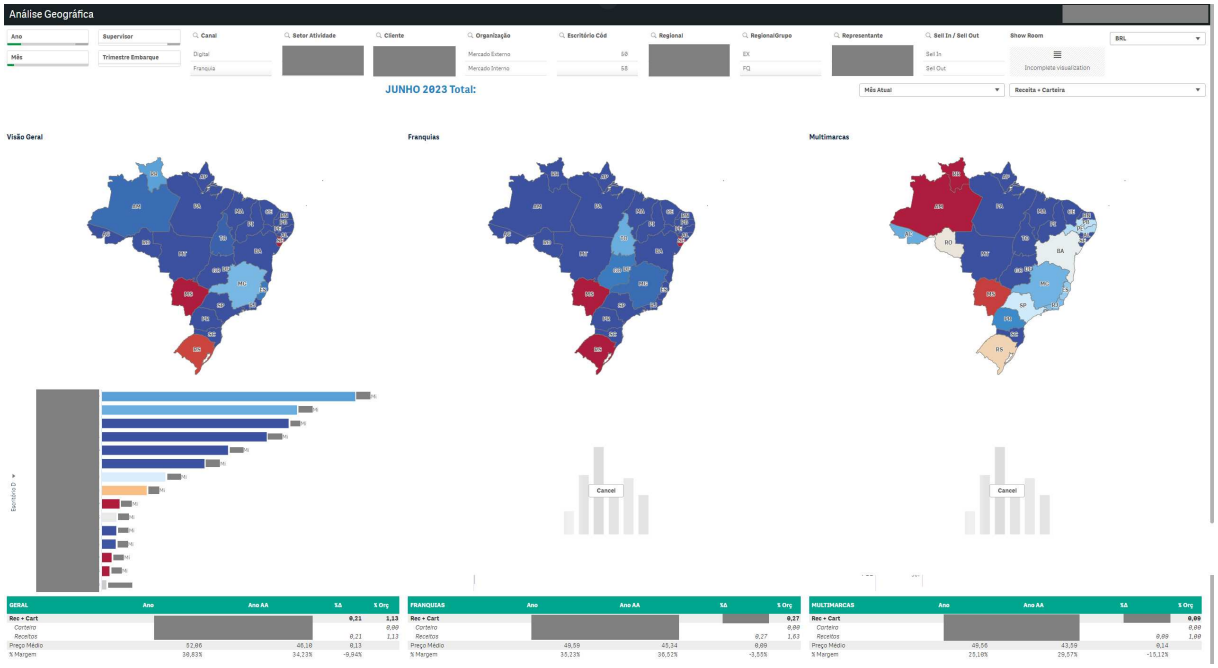
Figura 40 – Painel de análise de faturamento anual.



Fonte: Acervo do autor.

Já na Figura 41, uma abertura de faturamento quebrado em franquias ou multi-marcas regionalmente, avaliando o desempenho de marcas e lojas para cada Estado brasileiro.

Figura 41 – Painel de análise geográfica.



Fonte: Acervo do autor.

6.2.3 Áreas impactadas

A implementação do DW impacta diretamente em múltiplas equipes, pois, embora tenha sido originalmente planejado para o uso em um grande painel chamado BI Carteiras, o escopo do projeto é tão grande, abrangendo todo o sistema transacional e ERP que operacionaliza a empresa, que múltiplos times puderam se beneficiar das bases extraídas e processadas.

Primeiro de tudo, como esses dados alimentam a verdadeira “cabine de comando” de todo o setor de varejo da empresa cliente, é de se esperar um grande retorno financeiro, mesmo descontando todo o custo mensal requerido para manutenção e sustentação da arquitetura. É bastante difícil mensurar o impacto de um projeto grande como o apresentado pois, como supracitado, atinge o time de executivos do setor de varejo inteiro, incluindo: times focados no desempenho e fechamento de novos negócios e parceiros de venda, equipes focadas no comércio direto B2C, equipes buscando expansão no setor B2B, equipes de desenho e moda com interesse no desempenho de venda dos produtos projetados, executivos buscando relatórios financeiros detalhados e muitos outros. Todas essas equipes já possuíam algum tipo de inteligência de negócio e apoio de dados, mas sempre de forma esporádica e sob demanda, e agora gozam da possibilidade de interagir com dados relevantes à suas respectivas áreas em poucos cliques.

O gerenciamento de acessos à camada de consulta BigQuery é feito pela equipe

interna de TI, e conta atualmente com 47 usuários cadastrados com acesso direto ao BigQuery, isto é, pelo menos 47 pessoas do alto escalão da empresa foram diretamente afetados pela implementação do DW e podem realizar consultas às bases de dados limpas e validadas na busca por inteligência e conhecimento sobre o próprio negócio a partir dos dados disponibilizados.

7 CONCLUSÃO

Esse projeto contemplou grande parte da implementação da arquitetura de dados de uma grande empresa do ramo do varejo. Dentre os maiores desafios encontrados pelo autor no decorrer dos mais de 10 meses de projeto estão, em primeiro lugar, o alto volume de dados tratados, com algumas tabelas passando de bilhões de registros, e em alguns casos múltiplas tabelas com bilhões de registros precisavam ser unidas ou mapeadas entre si, o que pode ocasionar em um produto cartesiano com quintilhões ($1.000.000.000^2$) de combinações possíveis se nenhuma otimização for realizada ou pré-programada. Em segundo lugar, grandes desafios se deram no âmbito técnico-pessoal, com requisitos sendo mudados, prioridades trocadas, decisões tomadas tardiamente e mudanças de equipe ao longo dos meses de projeto, e em terceiro lugar, os desafios gerenciais, como consultor, o autor esteve pressionado à gerenciar expectativas, estipular e cumprir prazos e equilibrar trabalho, faculdade, família, relacionamento, amigos, saúde física e mental e lazer.

Em conclusão, o projeto implementado na empresa de varejo deu-se por completo, entregando valor ao cliente, confirmado por relatos e experiências com o DW construído, e permitindo uma análise quantitativa e qualitativa sobre operações de venda tanto diretamente para o consumidor final (B2C) quanto para outras empresas revendedoras ou parceiras (B2B). Foram entregues, em um primeiro momento, tabelas-réplica das presentes no SGBD transacional do sistema ERP, permitindo que analistas façam consultas *ad-hoc* livremente, sem dependência de processos burocráticos para consulta direto no ERP, sem restrições de horário e com desempenho (rapidez de resultado) muito superior do que sem o DW implantado, em casos reduzindo o atrito de até duas semanas para quase instantâneo. Também foram entregues as tabelas fato e dimensão para o modelo estrela desenhado, disponibilizando para consulta informações com enfoque na entidade de negócio de ordens de venda e múltiplas dimensões de apoio para bem caracterizar as particularidades de cada ordem: clientes associados, remessas feitas, produtos envolvidos, fretes e outros aspectos relevantes.

Isso permite que o cliente agora possa realizar quaisquer operações em cima das tabelas entregues de forma livre e quase instantânea. Consultas ao DW agora tomam segundos de processamento em cima dos dados já preparados, em vez de tomar semanas entre extração, preparação e tomada de decisão efetiva como acontecia previamente. Isso, claro, sem falar nas vantagens culturais que serão incrementalmente nutridas pelos colaboradores: cada vez mais o cliente deve utilizar-se de informação e conhecimento na tomada de decisão, abolindo o “achismo”, e também as vantagens tecnológicas que agora são possíveis, com grandes fluxos de dados potencialmente alimentando modelos preditivos e inferências estatísticas sobre o desempenho da máquina de vendas do setor do varejo, frente essa que está se iniciando com alguns times

internos.

Conforme retratados no Capítulo 6, os resultados impactaram na capacidade dos analistas da empresa cliente em realizar análises exploratórias no desempenho do setor de vendas do varejo de artigos de moda, habilitou o desenvolvimento de painéis de inteligência de negócio para resumir as informações coletadas e proporcionar o cálculo de Key Performance Indicators (KPIs) para metrificar e comparar esse desempenho ao longo do tempo. Como legado, o projeto também deixa aberta a possibilidade de novos projetos no âmbito de análise preditiva e estatística sobre vendas e faturamento utilizando-se de ferramentas e conceitos de ciência de dados na criação de modelos com AI e ML.

No geral, a formação em Engenharia de Controle e Automação se deu crucial em muitas etapas do projeto. As habilidades de solução de problemas, pensamento crítico, gestão de recursos e trabalho em equipe estão muitos presentes no mundo corporativo, e são especialmente úteis quando se tenta unir o conhecimento técnico com o conhecimento de negócios. Conhecimentos técnicos de redes, arquitetura de software, teoria de conjuntos, avaliação de desempenho, integração de sistemas e muitos outros estiveram presentes na graduação e também se mostraram válidos no mercado de trabalho, sejam como atores principais ou secundários no sucesso do projeto. Outro ponto de relevância é a capacidade analítica praticada no curso, exercitando a visão sistemática de processos com entradas pré-determinadas e as saídas esperadas nos processos.

7.1 TRABALHOS FUTUROS

Ainda existem grandes otimizações a serem feitas no projeto no âmbito técnico. Um dos maiores empecilhos para melhoria geral no desempenho dos *pipelines* de processamento está na versão defasada da ferramenta Apache Spark, que no momento da escrita se encontra estável na versão 3.5.0, e está implantado na versão 3.1.1 na infraestrutura do cliente. Essa defasagem é relativamente complexa de ser resolvida pois uma atualização na motorização do Spark requer uma atualização na versão da camada semântica do DW, o Delta Lake. Uma atualização de diversas ferramentas ao mesmo tempo é extremamente significativa e pode fazer com que *pipelines* críticos falhem, ou, no mínimo, deixem de ser executados enquanto a migração ocorre. Um grande limitante para isso é que ainda não há um ambiente pré-definido para desenvolvimento devido ao alto custo de implementação, e portanto, não existe ambiente para testar essa migração atualmente sem que haja ruptura em toda a infraestrutura atual. Essa migração de versões destravaria muitas funcionalidades e otimizações internas da motorização do Spark e na interpretação do Delta Lake, trazendo grandes benefícios de tempo e, principalmente, custo.

Outro ponto importante de ressaltar como melhoria futura é a revisão das rotinas

de extração e replicação das tabelas dos bancos de dados transacionais, visto que esta etapa não foi desenvolvida pelo autor, nem pela BIX Tech, e hoje conta com alguns problemas de perda de conexão e dessincronização, requerendo, ocasionalmente, a reingestão completa de algumas tabelas, e isso implica que os repositórios para as tabelas afetadas na *landing, bronze, silver* e eventualmente *gold* precisem ser completamente deletados, as tabelas reingeridas e reprocessadas, causando um atrito muito grande, atrasos e custos adicionais em uma eventual necessidade de escalar horizontalmente o *cluster*.

REFERÊNCIAS

ABADI, Daniel J.; MADDEN, Samuel R.; HACHEM, Nabil. Column-Stores vs. Row-Stores: How Different Are They Really? *In*: PROCEEDINGS of the 2008 ACM SIGMOD International Conference on Management of Data. New York, NY, USA: Association for Computing Machinery, 2008. P. 967–980. DOI: 10.1145/1376616.1376712. Disponível em: https://www.cs.umd.edu/~abadi/papers/abadi-sigmod08.pdf. Acesso em: 15 nov. 2023.

ACKOFF, Russell. From data to wisdom. **Journal of Applied Systems Analysis**, p. 3–4, 1989. Disponível em: http://www-public.imtbs-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/Ackoff89.pdf. Acesso em: 28 set. 2023.

ASTRONOMER. **Understanding the Airflow UI**. 2023. Disponível em: https://github.com/astronomer/airflow-guides/blob/main/guides/airflow-ui.md. Acesso em: 6 nov. 2023.

BEESETTY, Y.; PRAMOD, B.; VINEET, K. **Cloud Data Warehouse Market by Application Organization Size - Global Forecast to 2026**. [*S.l.*], 2021. Disponível em: https://www.alliedmarketresearch.com/data-warehousing-market. Acesso em: 19 out. 2023.

CAI, Wenju *et al.* Climate impacts of the El Niño–Southern Oscillation on South America. **Nature Reviews Earth Environment**, v. 1, p. 215–231, abr. 2020. DOI: 10.1038/s43017-020-0040-3. Disponível em: https://www.researchgate.net/publication/340635829_Climate_impacts_of_the_El_Nino-Southern_Oscillation_on_South_America. Acesso em: 8 out. 2023.

CHAMBERS, Bill; ZAHARIA, Matei. **Spark: The Definitive Guide Big Data Processing Made Simple**. 1st. [*S.l.*]: O'Reilly Media, Inc., 2018. ISBN 1491912219. Disponível em: https://analyticsdata24.files.wordpress.com/2020/02/spark-the-definitive-guide40www.bigdatabugs.com_.pdf. Acesso em: 7 nov. 2023.

CHANDY, K. Mani. **Event Driven Architecture**. Edição: LING LIU e M. TAMER ÖZSU. Boston, MA: Springer US, 2009. P. 1040–1044. ISBN 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_570. Disponível em: https://doi.org/10.1007/978-0-387-39940-9_570.

CODD, E. F. A Relational Model of Data for Large Shared Data Banks. **Commun. ACM**, v. 13, p. 377–387, 1970. Disponível em: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf. Acesso em: 8 out. 2023.

CONN, Samuel. OLTP and OLAP data integration: A review of feasible implementation methods and architectures for real time data analysis, p. 515–520, mai. 2005. DOI: 10.1109/SECON.2005.1423297. Disponível em: https://www.researchgate.net/publication/4140602_OLTP_and_OLAP_data_integration_A_review_of_feasible_implementation_methods_and_architectures_for_real_time_data_analysis. Acesso em: 17 nov. 2023.

DATABRICKS. **Faster Merge Performance with Low Shuffle Merge and Photon**. Databricks. Out. 2022. Disponível em: https://www.databricks.com/blog/2022/10/17/faster-merge-performance-low-shuffle-merge-and-photon.html. Acesso em: 20 nov. 2023.

DELTA LAKE FOUNDATION. **Delta Lake Time Travel**. 2023. Disponível em: https://delta.io/blog/2023-02-01-delta-lake-time-travel/. Acesso em: 20 nov. 2023.

ECONOMIST, The. **Global shipping costs are returning to pre-pandemic level**. 2023. Disponível em: https://www.economist.com/graphic-detail/2023/01/09/global-shipping-costs-are-returning-to-pre-pandemic-levels. Acesso em: 20 set. 2023.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Fundamentals of Database Systems**. 7th. [*S.l.*]: Pearson, 2015. ISBN 0133970779. Disponível em: https://www.auhd.edu.ye/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf. Acesso em: 6 out. 2023.

GCP. **Cloud Storage Pricing**. 2023. Disponível em: https://cloud.google.com/storage/pricing#south-america. Acesso em: 27 out. 2023.

GOOGLE CLOUD PLATFORM. **BigQuery API Documentation**. [*S.l.*: *s.n.*], 2023a. Disponível em: https://cloud.google.com/bigquery/docs/reference/rest. Acesso em: 16 nov. 2023.

GOOGLE CLOUD PLATFORM. **Google Cloud Platform**. [*S.l.*: *s.n.*], 2023b. Disponível em: https://console.cloud.google.com/. Acesso em: 17 nov. 2023.

GOOGLE CLOUD PLATFORM. **Google Cloud Products**. 2023c. Disponível em: https://cloud.google.com/products#featured-products/. Acesso em: 21 nov. 2023.

HASHEM, Ibrahim Abaker Targio; YAQOOB, Ibrar; ANUAR, Nor Badrul; MOKHTAR, Salimah; GANI, Abdullah; ULLAH KHAN, Samee. The rise of "big data" on cloud computing: Review and open research issues. **Information Systems**, v. 47, p. 98–115, 2015. ISSN 0306-4379. DOI: https://doi.org/10.1016/j.is.2014.07.006. Disponível em: https://www.sciencedirect.com/science/article/pii/S0306437914001288. Acesso em: 16 nov. 2023.

INMET. **Banco de Dados Meteorológicos do Instituto Nacional de Meteorologia**. 2022. Disponível em: https://bdmep.inmet.gov.br/. Acesso em: 29 set. 2023.

INMON, William H. **Building the Data Warehouse**. 4. ed. [*S.l.*]: Wiley Publishing Inc., 2005. Disponível em: https://ia800202.us.archive.org/9/items/2005BuildingTheDataWarehouse4thEditionWilliamH.Inmon/2005%20-%20Building%20The%20Data%20Warehouse%20%284th%20Edition%29%20%28William%20H.%20Inmon%29.pdf. Acesso em: 26 out. 2023.

KOLONKO, Kamil. **Performance comparison of the most popular relational and non-relational database management systems**. 2018. Master of Science in Software Engineering – Blekinge Institute of Technology Faculty of Computing. Disponível em: https://www.diva-portal.org/smash/get/diva2:1199667/FULLTEXT02.pdf. Acesso em: 16 out. 2023.

KSHEMKALYANI, Ajay D.; SINGHAL, Mukesh. **Distributed Computing: Principles, Algorithms, and Systems**. 1. ed. USA: Cambridge University Press, 2008. ISBN 0521876346. Disponível em: https://books.google.com.br/books?hl=en&lr=&id=G7SZ32dPuLgC&oi=fnd&pg=PR7&dq=transitioning+to+distributed+computing&ots=fO3rs0NC8g&sig=842_lJCRLCbFjt-Sbj-qsG9rnmA#v=onepage&q&f=true. Acesso em: 21 nov. 2023.

MOFADDEL, Mahmoud; TAVANGARIAN, Djamshid. A Distributed System with a Centralized Organization. *In*: 1. Workshop Cluster-Computing. [*S.l.*: *s.n.*], 1997. Disponível em: https://www.researchgate.net/publication/2241128_A_Distributed_System_with_a_Centralized_Organization. Acesso em: 21 nov. 2023.

MUNHOZ DE MEDEIROS, Mauricius; HOPPEN, Norberto; MAÇADA, Antonio Carlos. Data science for business: benefits, challenges and opportunities. **The Bottom Line**, ahead-of-print, mar. 2020. DOI: 10.1108/BL-12-2019-0132. Disponível em: https://www.researchgate.net/publication/340234958_Data_science_for_business_benefits_challenges_and_opportunities. Acesso em: 17 nov. 2023.

PETROV, Alex. **Database Internals**. 1. ed. United States of America: O'Reilly Media, Inc., 2019. Disponível em: https://dokumen.pub/database-internals-a-deep-dive-into-how-distributed-data-systems-work-1492040347-9781492040347-i-3646633.html. Acesso em: 17 out. 2023.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. **Database Management Systems**. 3. ed. United States of America: McGraw-Hill, Inc., 2002. ISBN 0072465638. Disponível em: https://xuanhien.files.wordpress.com/2011/04/database-management-systems-raghu-ramakrishnan.pdf. Acesso em: 17 out. 2023.

SUBRAHMANYAM, Voore; KUMAR, Sarvesh; SRIVASTAVA, Satyajee; BIST, Ankur Singh; SAH, Basant; PANI, Niroj Kumar; BHAMBU, Pawan. Optimizing horizontal scalability in cloud computing using simulated annealing for Internet of Things. **Measurement: Sensors**, v. 28, p. 100829, 2023. ISSN 2665-9174. DOI: https://doi.org/10.1016/j.measen.2023.100829. Disponível em: https://www.sciencedirect.com/science/article/pii/S2665917423001654. Acesso em: 6 nov. 2023.

TRADE, United Nations Conference on; DEVELOPMENT. **UNCTAD Handbook of Statistics 2022 - Maritime transport**. 2022. Disponível em: https://unctad.org/system/files/official-document/tdstat47_FS14_en.pdf. Acesso em: 20 set. 2023.

ZAHARIA, Matei A.; GHODSI, Ali; XIN, Reynold; ARMBRUST, Michael. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. *In*: CONFERENCE on Innovative Data Systems Research. [*S.l.*: *s.n.*], 2021.

Disponível em: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf. Acesso em: 17 nov. 2023.