Um estudo sobre ferramentas de código aberto para carga de dados de documentos fiscais eletrônicos

Genival José de Moura Neto

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, PB
Junho - 2022

Genival José de Moura Neto

Um estudo sobre ferramentas de código aberto para carga de dados de documentos fiscais eletrônicos

Monografia apresentada ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Raoni Kulesza

João Pessoa, PB
Junho - 2022

Catalogação na publicação
Seção de Catalogação e Classificação

N469e Neto, Genival José de Moura.
Um estudo sobre ferramentas de código aberto para carga de dados de documentos fiscais eletrônicos / Genival José de Moura Neto. - João Pessoa, 2022.
61 f.

Orientação: Raoni Kulesza.
TCC (Graduação) - UFPB/CI.

1. Dados. 2. Big Data. 3. ETL. 4. ELT. 5. Código aberto. 6. Documento fiscal eletrônico. I. Kulesza, Raoni. II. Título.

UFPB/CI CDU 004.6

Elaborado por ROGERIO FERREIRA MARQUES - CRB-15/690

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Ciência da Computação intitulado **Um estudo sobre ferramentas de código aberto para carga de dados de documentos fiscais eletrônicos** de autoria de Genival José de Moura Neto, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Raoni Kulesza
Universidade Federal da Paraíba

Prof. Carlos Eduardo Silveira Dias
Universidade Federal da Paraíba

Prof. Marcelo Iury de Sousa Oliveira
Universidade Federal da Paraíba

João Pessoa, 30 de junho de 2022

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

RESUMO

Este trabalho levantou o estado da técnica na área de extração, carga e transformação de dados e analisou duas alternativas de código-fonte aberto: LoaderNFE e Airbyte. A primeira é composta de várias ferramentas e atualmente realiza o processo de ETL no projeto da Plataforma de Integração de Dados da SEFAZ-PB de modo não automatizado. A segunda propõe uma alternativa baseada em processos ELT e possui uma interface Web para operação e automação de algumas tarefas. Adicionalmente, as duas ferramentas foram avaliadas no ambiente da SEFAZ-PB para extração, transformação e carga de 33 lotes de notas fiscais eletrônicas do consumidor, cada um com 125.000 documentos. O objetivo foi avaliar as principais diferenças de configuração e operação entre as abordagens e realizar um estudo comparativo de desempenho para verificar a viabilidade de utilizar o Airbyte para realização de sincronismo diário entre a base operacional da SEFAZ-PB e a Plataforma de Integração de Dados. Os resultados obtidos indicam que o Airbyte pode ser utilizado em cenários que requerem uma sincronização automática e mais frequente entre fontes e destinos de dados, mesmo sendo mais lento que o LoaderNFE.

Palavras-chave: Big Data, ETL, ELT, código aberto, documento fiscal eletrônico.

ABSTRACT

This work raised the state of the art in the area of data extraction, loading and transformation and analysis of two open-source code alternatives: LoaderNFE and Airbyte. At first, it composts various tools and currently performs the ETL process of the SEFAZ-PB Data Integration Platform project in a non-automated way. Secondly, it proposes an alternative based on ELT processes and has a Web interface for the operation and automation of some tasks. Additionally, the two tools are endorsed in the environment of SEFAZ-PB for extraction, transformation and loading of 33 batches of consumer electronic tax notes, each with 125,000 documents. The objective was to assess the main differences in configuration and operation between the approaches and carry out a comparative study of performance to verify the feasibility of using Airbyte to perform daily synchronization between the operational base of SEFAZ-PB and the Data Integration Platform. The results obtained indicate that Airbyte can be used in scenarios that require an automatic and more frequent synchronization between data sources and destinations, even being slower than LoaderNFE.

Key-words: Big Data, ETL, ELT, open source, electronic tax invoice.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ELT	–	*Extract-Load-Transform*
SQL	–	*Structured Query Language*
dbt	–	*data build tool*
XML	–	*eXtensible Markup Language*
JSON	–	*JavaScript Object Notation*
NFC-e	–	Nota Fiscal de Consumidor Eletrônica
NF-e	–	Nota Fiscal Eletrônica

SUMÁRIO

1. INTRODUÇÃO

A crescente evolução no cenário de *Big Data* vem representando grandes desafios para múltiplos domínios, inclusive para órgãos responsáveis pela gestão pública e fazendária como as Secretarias de Estado da Fazenda (SEFAZ). Por exemplo, na secretaria da Paraíba (SEFAZ-PB) atualmente são recebidas num intervalo de 24 horas mais de 1 milhão de notas fiscais de consumidor eletrônico modelo 65 (NFC-e).

Assim, a capacidade de processar esse grande volume de dados com velocidade e eficiência é essencial para que informações úteis estejam incluídas durante a análise, o que de outra forma poderia levar a decisões de negócio erradas ou, dependendo do cenário, o resultado do processamento pode não ter valor. Por exemplo, as Secretarias Estaduais da Fazenda fazem fiscalizações e possuem um prazo legal para atuar nas empresas. Porém, como o volume de notas fiscais geradas é muito grande e vem crescendo cada vez mais, não adianta as Secretarias fazerem uma análise e perderem o prazo legal que elas poderiam atuar com um determinado contribuinte.

1.1. DEFINIÇÃO DO PROBLEMA

Um dos principais desafios em relação a um processo decisório útil no âmbito de *Big Data* é dispor de dados atualizados [25]. A partir disso, a solução atual utilizada na SEFAZ-PB não é adequada para utilização em cenários que requerem uma sincronização de dados automática e mais frequente. Em primeiro lugar, a solução em questão só permite uma execução manual. Em segundo lugar, ela não tem conexão com a base de origem das notas fiscais, o que dificulta o monitoramento de novos dados. Além disso, a solução em questão lê os dados de um sistema de arquivos e em aspectos como indexação, integridade e segurança uma solução mais adequada deveria interagir com um banco de dados.

1.2. Objetivo geral

Analisar e propor uma solução alternativa à solução atual utilizada na SEFAZ-PB no que diz respeito à extração, transformação e carga de dados de documentos fiscais eletrônicos no contexto do projeto "Plataforma Integrada de Dados" realizada em parceria entre a UFPB (por intermédio dos laboratórios LAVID e ARIA), UEPB (Universidade Estadual da Paraíba) e SEFAZ-PB.

1.2.1. Objetivos específicos

- Levantar o estado da técnica na área de extração, carga e transformação de dados;
- Analisar a solução atual utilizada no projeto "Plataforma de Integração de Dados";
- Documentar os requisitos e arquitetura da solução atual e de uma solução alternativa de código-fonte aberto;
- Realizar testes comparativos de uso das duas soluções num ambiente de homologação da SEFAZ-PB.

1.3. Estrutura da monografia

O restante deste texto está estruturado em mais 5 (cinco) capítulos da seguinte forma:

- O capítulo 2 contém a fundamentação teórica sobre ELT e algumas formas de implantação dessa abordagem, além de apresentar motivos do que vem causando sua adoção cada vez maior em relação às abordagens tradicionais ETL.

- O capítulo 3 descreve a metodologia utilizada para realizar os testes de uso e avaliar o desempenho entre as 2 soluções consideradas.
- O capítulo 4 apresenta a solução atual utilizada na SEFAZ-PB e a solução alternativa de código-fonte aberto construída neste estudo.
- O capítulo 5 apresenta os resultados obtidos nos testes de uso entre as 2 soluções consideradas.
- O capítulo 6 apresenta as considerações finais e possibilidades de trabalhos futuros.

2. CONCEITOS GERAIS E REVISÃO DA LITERATURA

Neste capítulo são descritos os principais conceitos para o entendimento do trabalho, no qual duas abordagens de integração de dados (ETL e ELT) são conceituadas e diferenciadas e como o cenário de *Big Data* vem impactando na escolha entre elas.

2.1. SISTEMAS DE SUPORTE À DECISÃO

Sistemas de Suporte/Apoio à Decisão (SSD/SAD), do inglês *Decision Support Systems* (DSS), são sistemas desenvolvidos para apoiar a solução de um problema gerencial e para aperfeiçoar a tomada de decisão. Esses sistemas utilizam dados disponíveis e permitem ao tomador de decisão realizar inferências sobre os resultados. Surgiram aproximadamente entre as décadas de 60 e 70 resultado de vários fatores, dentre os quais destacam-se a evolução tecnológica ao nível de hardware e de software, uma crescente preocupação com o suporte ao processo de tomada de decisão, um crescente ambiente econômico turbulento e um aumento das pressões competitivas [16].

O grande problema naquele período era a incompatibilidade da arquitetura das bases de dados dos sistemas computacionais para a realização de pesquisas típicas de Sistemas de Apoio à Decisão e, devido à falta de dados históricos, existia muita dificuldade para a criação de relatórios e realizar as análises necessárias ao gerenciamento dos negócios. Somente nos anos 80, quando surgiram os primeiros Sistemas de Gerenciamento de Banco de Dados (SGBD), é que se tornou possível um melhor acesso aos dados disponíveis, à sua formatação e à construção de consultas e relatórios de uma forma mais prática, rápida e barata. Ainda segundo [16], com a chegada ao mercado de novas categorias de sistemas de softwares, como *Data Warehouse* e OLAP, estas atividades se tornaram menos complexas e as consultas e os relatórios passaram a ser confeccionados pelos próprios usuários, sem um conhecimento profundo de tecnologias computacionais.

De modo geral, uma arquitetura DSS apresenta as seguintes características [23 e 24]:

- **Gerenciamento de dados**: incluindo extração de dados, limpeza de dados, integração de dados, bem como armazenamento e manutenção eficiente de grandes quantidades de dados.
- **Análise de dados**: incluindo consultas de informações, geração de relatórios e funções de visualização de dados.
- **Descoberta de Conhecimento**: extração de informações úteis (conhecimento) dos volumes crescentes de dados digitais em bancos de dados.

A camada de gerenciamento de dados é extremamente importante para o sucesso do Sistema de Suporte à Decisão [23 e 24] e é a parte mais demorada em termos de implementação, podendo consumir até 80% do projeto [22].

2.2. DSS E *BIG DATA*

No ano 2000 Peter Lyman e Hal R. Varian publicaram "How Much Information?" que é considerado o primeiro estudo abrangente a quantificar, em termos de armazenamento computacional, a quantidade total de informações novas e originais (sem contar as cópias) criadas no mundo anualmente e armazenadas em quatro meios físicos: papel, filme, óptico (CDs e DVDs) , e magnético. Um estudo semelhante realizado em 2003 pelos mesmos pesquisadores [18] evidenciou que o mundo produziu cerca de 5 exabytes de novas informações em 2002 e que 92% das novas informações foram armazenadas em mídia magnética, principalmente em discos rígidos.

Já em 2011, Martin Hilbert e Priscila Lopez publicaram o artigo "The World's Technological Capacity to Store, Communicate, and Compute Information" [19] no qual estimaram que a capacidade de armazenamento de informações do mundo cresceu a uma taxa anual de aproximadamente 25% entre 1986 e 2007. Além disso, também foi estimado que em 1986, 99,2% de toda a capacidade de armazenamento era analógica, mas em 2007, 94% da capacidade de armazenamento era digital, uma

completa inversão de cenário (em 2002, o armazenamento de informações digitais ultrapassou pela primeira vez o não digital).

O termo *Big Data* surgiu da aceitação de 3 conceitos apresentados por Doug Laney em sua pesquisa de 2001 intitulada "3D Data Management: Controlling Data Volume, Velocity, and Variety" [17]. Informalmente o termo refere-se aos conjuntos de dados, normalmente coletados por empresas e governos, que são tão grandes e complexos que os métodos tradicionais de processamento são inadequados para lidar com os cálculos necessários para entender a vasta quantidade de informações escondidas dentro da sua estrutura. Formalmente, na sua origem, o termo sustentava a existência dos "3Vs": Volume, Velocidade e Variedade. Volume indica o fato da enorme quantidade de dados que precisa ser processada. Velocidade argumenta que a velocidade de processamento também é crucial, pois os dados são gerados e inseridos no armazenamento de dados em alta velocidade. A variedade alega que os dados são provenientes de várias fontes heterogêneas (redes sociais, sensores, dados transacionais, etc.).

A partir de então, com o intuito de superar os novos desafios revelados pelo *Big Data* surgiram alternativas ao *Data Warehouse*, como os *Data Lakes*, que basicamente se diferem de acordo com o volume e o formato dos dados que armazenam para futuras análises [27].

2.3. ETL

O novo cenário conduzido pelo *Big Data* representou um grande desafio para múltiplos domínios, particularmente em termos de Sistema de Suporte à Decisão (DSS). A partir disso, a primeira abordagem utilizada para lidar com a importância de analisar toda essa grande evolução dos dados que se encontravam em diversas bases de dados e em diversos formatos foi chamada de ETL [20, 21 e 22].

ETL é um acrônimo para extrair, transformar e carregar onde os dados são primeiro extraídos da fonte de dados original, depois transformados, incluindo normalização e limpeza, e finalmente carregados no *Data Warehouse*. Geralmente isso

acontece num processo periódico, normalmente executado depois do horário comercial, ou seja, um dia inteiro de dados é processado e carregado fora dos horários de pico [28].

No entanto, com o cenário crescente do *Big Data*, é importante ter dados o mais atualizados quanto possível para análise e decisões de negócio quase em tempo real [25].

2.4. ETL vs ELT

Apesar de existirem diferentes abordagens de implementação de ETL [26], de maneira geral as operações clássicas não suportam a evolução no cenário de *Big Data*, o que vem acarretando tentativas de adaptações [22]. Embora as tecnologias de banco de dados tenham apresentado grandes melhorias de desempenho e escalabilidade na última década, o ETL não conseguiu aprimoramentos consideráveis nesses mesmos aspectos. Como resultado, cada vez mais um gargalo persiste: os dados não conseguem ser facilmente adquiridos para o local de armazenamento (ex.: para o *Data Warehouse*) com um intervalo de atualização necessário, problema conhecido como latência de dados [24 e 28]. Uma das principais desvantagens do ETL é que os dados devem primeiro ser transformados e só então carregados. Isso significa que, dependendo do cenário, as tomadas de decisão podem ser atrasadas devido ao tempo necessário para concluir primeiro todo o processamento ETL, ou mesmo grandes quantidades de dados potencialmente valiosos podem não estar disponíveis durante uma análise, levando a utilização de dados desatualizados que podem acarretar em informações incompletas que comprometem o processo decisório [25].

A partir disso, uma das primeiras abordagens criadas para lidar com tais desafios foi chamada de ELT [25]. Essa abordagem permite primeiro extrair e carregar dados e, em seguida, aplicar transformações sob demanda de acordo com as necessidades do negócio, acelerar o processo de implementação, além de outras vantagens [24, 6 e 29].

Cabe também ressaltar que há uma corrente de pensamento que considera ELT uma transição para EL(T), abordagem essa que desacopla completamente as etapas Extrair-Carregar de qualquer transformação opcional que possa ocorrer, pois o ELT não estaria resolvendo completamente o problema de integração de dados por apresentar seus próprios problemas [11].

2.5. ELT

ELT é um acrônimo para "*Extract, Load, Transform*" e representa um processo de dados usado para replicar dados de uma origem para um destino de acordo com as seguintes etapas (ver Figura 2.1):

1. **Extração**: coleta e extração de dados brutos de uma ou diversas fontes (origem) para posterior integração em um repositório de dados único (destino).
2. **Carregamento**: carregamento dos dados coletados, geralmente num Data Warehouse provisionando-os para uso analítico.
3. **Transformação**: transformação dos dados brutos em dados modelados para, por exemplo, aplicação de BI (do inglês, *Business Intelligence*) [24 e 25].

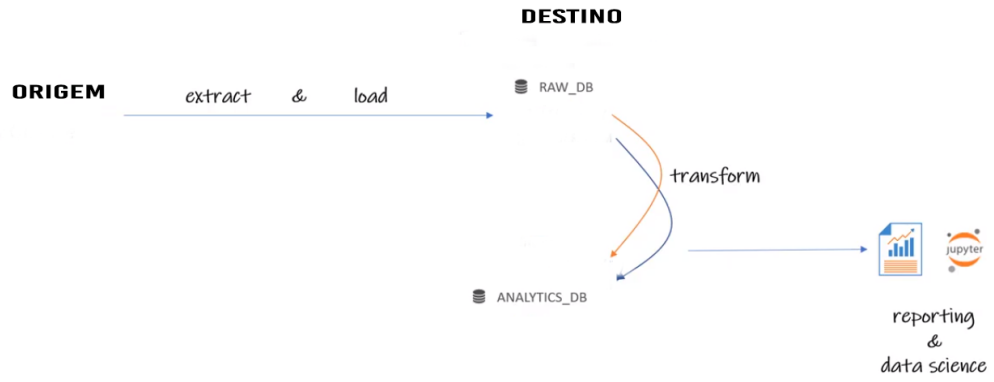

Figura 2.1: Processo ELT

Fonte: Adaptado de [1]

Adicionalmente, como é mostrado na Figura 2.2, a extração pode ocorrer de diferentes fontes de dados, por exemplo, do Google Ads, Salesforce, de uma determinada API ou de qualquer outra origem cujo dados sejam úteis, representem valor para o usuário e que ele tenha acesso.

Existem várias ferramentas especializadas nas etapas de extração e carregamento do processo ELT, como também pode ser visto na Figura 2.2. Algumas são proprietárias e/ou pagas como o Fivetran, outras gratuitas e de código aberto como o Airbyte, porém todas apresentam características em comum no sentido de monitorar mudanças na origem e sincronizar isso no destino, possibilitando ao usuário configurar frequências de atualização baseadas em intervalos de minutos ou horas. Outra característica marcante é que essas ferramentas apresentam algum mecanismo de conexão, do qual são capazes de extrair de diversas fontes de dados comumente utilizadas e carregar em vários dos principais *Data Warehouses*, *Data Lakes* e SGBD. Além disso, muitas dessas ferramentas também permitem a transformação dos dados.

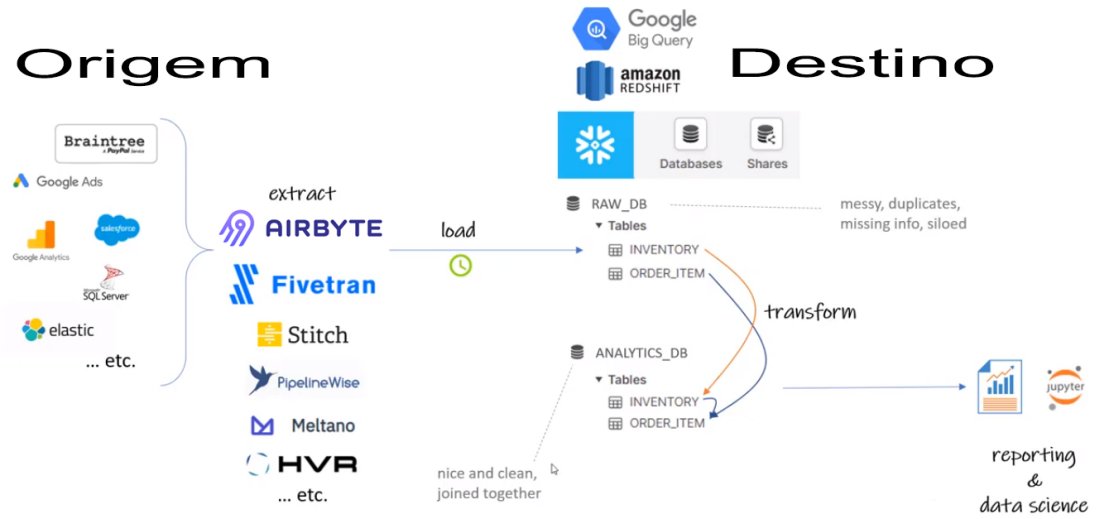

Figura 2.2: Exemplos de fontes de dados e ferramentas comumente utilizadas em pipelines ELT

Fonte: Adaptado de [1]

A Tabela 2.1 apresenta uma breve comparação entre algumas das principais ferramentas utilizadas no processo ELT.

Tabela 2.1: Comparativo entre as principais ferramentas utilizadas no processo ELT.

Fonte: Adaptado de [31]

	Airbyte	**Fivetran**	**Stitch**	**PipelineWise**	**Meltano**
Fundado em	2020	2012	2016	2018	2019
Origens	Aproximadamente 170 apenas no primeiro ano de desenvolvimento. Objetivo de alcançar a marca de 500 até o final de 2022.	Aproximadamente 150.	Aproximadamente 130	Aproximadamente 18	Aproximadamente 140
Destinos	Todos os principais *Data Warehouses*, *Data Lakes* e SGBDs.	Todos os principais *Data Warehouses* e SGBDs.	Todos os principais *Data Warehouses* e SGBDs.	Apenas 5, incluindo Snowflake, Redshift, Postgres, S3 e Bigquery.	Todos os principais *Data Warehouses*, *Data Lakes* e SGBDs.

Integração com outras tecnologias na stack de dados	Kubernetes, dbt e mais, além de contar com a contribuição da comunidade.	Suporte a transformações dbt.	Não	Não	Integrado profundamente com dbt.
Código aberto	Sim (licença MIT e Elastic 2.0 (ELv2))	Não	Sim (licença Apache 2.0)	Sim (licença Apache 2.0)	Sim (licença Apache 2.0)
Customização dos Conectores	Usuários podem editar qualquer um dos pré-existentes para criar o seu próprio.	Limitado aos serviços de nuvem ao qual o Fivetran é integrado.	Usuários podem editar qualquer um dos pré-existentes para criar o seu próprio.	Usuários podem editar qualquer um dos pré-existentes para criar o seu próprio.	Usuários podem editar qualquer um dos pré-existentes para criar o seu próprio.
SDK para facilitar a implementação dos Conectores	Java, Python ou Go	Go	Clojure	Python	Python

2.6. TRANSFORMAÇÕES

As transformações geralmente envolvem a conversão de uma fonte de dados brutos em um formato limpo, validado e pronto para uso. Isso porque, cada vez mais, as organizações precisam ser capazes de extrair análise de seus dados seja para

competir com sucesso no mercado digital, otimizar operações, cortar custos, aumentar a produtividade, etc. [24].

Dentre as estratégias de transformação de dados, existem as seguintes [4 e 5]:

1. **Agregação**, na qual os dados podem ser sumarizados de várias formas. Por exemplo, dados de vendas diárias podem ser agregados para calcular os valores totais mensais e anuais;
2. **Construção de atributos**, na qual novos atributos são adicionados ou criados a partir de atributos existentes;
3. **Discretização**, que envolve a conversão de valores de dados contínuos em conjuntos de intervalos de dados com valores específicos para tornar os dados mais gerenciáveis para análise. Por exemplo, valores brutos de um atributo numérico (por exemplo, idade) podem ser substituídos por rótulos de intervalo (por exemplo, 0–10, 11–20 etc.) ou rótulos conceituais (por exemplo, jovem, adulto, idoso) ; e
4. **Manipulação**, onde os dados são alterados ou modificados para torná-los mais legíveis e organizados. Por exemplo, modificar um valor timestamp para uma data;

Em termos de software, esse processo pode ser feito de uma forma automatizada, levando em consideração algumas das soluções integradas de ferramentas ELT descritas na seção anterior ou através de ferramentas específicas como o Trifacta, gerenciado manualmente/programaticamente através de scripts SQL(do inglês, *Structured Query Language*) ou de uma linguagem de programação para propósitos diversos como Python ou usando uma combinação das duas abordagens, como é o caso da ferramenta EL(T) Airbyte que oferece a possibilidade de integrar transformações baseadas em SQL usando uma ferramenta de transformação especializada conhecida como dbt (do inglês, *data build tool)*.

2.7. Transformações baseadas em SQL

Buscando resolver o problema dos dados em sua forma bruta e conseguir extrair valor deles, seja para identificação de problemas, tomada de decisões ou previsão de tendências, uma das abordagens, como já mencionado, é programaticamente a partir de SQL. Essa abordagem cumpre bem seu papel dando a possibilidade de deixar os dados numa forma pronta para análise, fácil de consultar e entender, agrupados numa fonte única de verdade, tudo isso com um único *script* SQL [1], como mostrado na Figura 2.3.

```sql
create schema if not exists analytics;

drop table if exists analytics.inventory;
create table analytics.inventory as
select
    i.idinventory as id_inventory,
    i.gendate as gen_timestamp,
    i.gendate::date as gen_date,
    i.idisbn as id_isbn,
    i.idsitex as id_warehouse,
    case when i.idsitex >= 2020 then 'new' else 'used' end as book_type
from raw.inventory i;

select gen_date,
    book_type,
    count(*) qty
from analytics.inventory
group by 1, 2
order by 1, 2;

drop table if exists analytics.order_item;
create table analytics.order_item as
select
    oi.idorderitemidentifier as id_order_item,
    oi.idinventory as id_inventory,
    oi.idorder as id_order,
    oi.soldprice as sold_price,
    i.book_type
from raw.order_item oi
inner join analytics.inventory i on i.id_inventory = oi.idinventory;

select book_type,
    avg(sold_price) avg_sold_price,
    count(*) sold_count,
    sum(sold_count) total_revenue
from analytics.order_item
group by book_type;
```

Figura 2.3: Exemplo de script SQL para transformação de dados

Fonte: Adaptado de [1]

Porém, a abordagem anterior tem um sério problema de manutenibilidade à medida em que a equipe de desenvolvimento e o conjunto de tabelas cresce. A partir disso, surgem problemas no gerenciamento da sequência de execução das operações e consultas presentes no *script*, causadas por dependências; no gerenciamento de ambientes, conceito importante na Engenharia de Software para permitir que o desenvolvimento e teste dos códigos não afetem os usuários finais; na identificação de *bugs* e automatização de testes; no gerenciamento de performance; na documentação; etc.

Levando isso em consideração, a ferramenta dbt[1] (*data build tool*) foi criada justamente pensando em resolver esses problemas, permitindo que o processo de transformação de dados seja acompanhado de boas práticas de Engenharia de Software (ver Figura 2.4) como documentação e testes automatizados [3 e 7]. Além disso, o problema com gerenciamento de dependências desaparece, pois a própria ferramenta cria automaticamente um grafo de dependências do tipo DAG (do inglês, *Directed Acyclic Graph*) e utiliza-o para executar as operações na sequência correta e necessária. Tudo isso é possível porque o código dbt é uma combinação de SQL e Jinja[2], um mecanismo de template comumente utilizado no ecossistema Python [2].

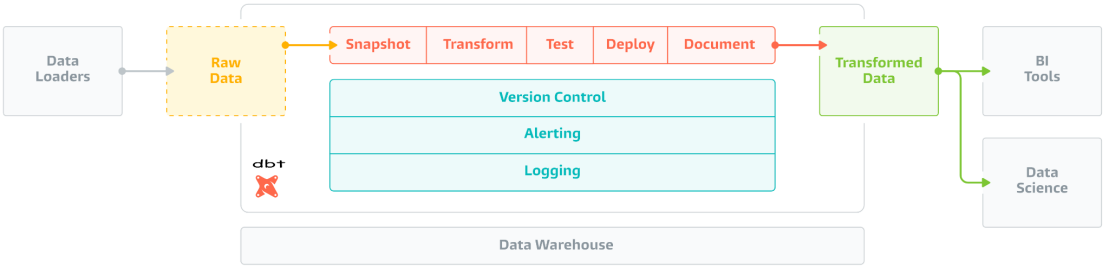

Figura 2.4: Diagrama estrutural da ferramenta dbt

Fonte: Adaptado de [8]

2.8. XML e XPath

XML (do inglês, *eXtensible Markup Language*) é uma linguagem de marcação projetado para armazenar e transportar dados. Ao contrário de outras linguagens de

[1] dbt. Disponível em: <https://www.getdbt.com/>.
[2] Jinja. Disponível em: <https://jinja.palletsprojects.com/>.

marcação, XML não possui um conjunto de *tags* predefinidas, sendo responsabilidade do autor definir tanto as *tags* quanto a estrutura do documento, visando ser autodescritivo - outro aspecto para o qual a linguagem foi projetada [9].

XPath (do inglês, XML *Path Language*) é uma linguagem de consulta para selecionar e navegar em nós baseado numa representação de árvore do documento XML. Para isso, a linguagem utiliza expressões de caminho muito parecidas com as expressões utilizadas no sistema de arquivos de um computador e mais de 200 funções integradas para computar valores (por exemplo, strings, números, filhos do nó atual, etc) do conteúdo de um documento XML [10].

2.9. SPED

Inspirado em modelos de governos eletrônicos de países como Espanha, Chile e México, entre outros, o governo brasileiro criou o Sistema Público de Escrituração Digital (SPED), com isso constituindo uma revolução caracterizada por trazer a contabilidade das empresas para o formato digital; gerando significativa redução nos custos; aumentando a celeridade do processo; auxiliando no combate à sonegação, além de possibilitar o cruzamento de informações entre contribuintes, entre outras vantagens [32].

Existem 3 principais subprojetos que compõem o SPED, sendo estes arquivos eletrônicos que devem ser entregues ao fisco, sendo eles: a Escrituração Contábil Digital (ECD), Escrituração Fiscal Digital (EFD) e a NF-e. Este último dos 3 subprojetos, a NF-e, deu origem a duas outras categorias de documentos fiscais emitidos eletronicamente: Nota Fiscal de Serviço Eletrônica (NFS-e) e Nota Fiscal de Consumidor Eletrônica (NFC-e) [34]. Acontece é que justamente as informações dessas notas ficam armazenadas em XML, seguindo um esquema específico que define o conteúdo do documento eletrônico e a sua organização [33].

3. METODOLOGIA

A partir da pesquisa realizada e levando em consideração a flexibilidade e facilidade em termos de implementação, haja vista uma ampla adoção da comunidade de código aberto, foi escolhido construir a solução alternativa a partir da ferramenta Airbyte, que será descrita em mais detalhes na próxima seção.

Também foi preparado uma infraestrutura para realização dos testes de uso do Airbyte e do LoaderNFE, onde inicialmente foi definida as bases de dados de Origem e Destino que seriam utilizadas neste estudo e os requisitos dos recursos computacionais que seriam utilizados para executar os testes de sincronização dos documentos fiscais tanto com o Airbyte, quanto com a solução atual utilizada na SEFAZ-PB (LoaderNFE).

A base de dados de Origem utilizada foi a base de NFC-e pertencente a Secretaria de Estado da Fazenda da Paraíba - SEFAZ-PB por se tratar de uma base confiável em relação ao documento em questão (NFC-e) e encontrava-se num banco de dados Oracle na versão 12c Enterprise Edition Release 12.1.0.2.0. Já a base de dados de Destino tratou-se de um banco de dados Postgres na versão 13.1. Além disso, os dados utilizados durante a pesquisa encontravam-se anonimizados.

Em relação ao ambiente de execução dos testes, foi utilizada uma máquina virtual proveniente do serviço de nuvem privada da SEFAZ-PB, servindo para hospedar a base de Destino e executar os testes de sincronização com as ferramentas consideradas. A configuração da máquina virtual utilizada para executar as sincronizações, como é mostrado na Figura 3.1, tratou-se: Sistema Operacional Linux CentOS 7, com 64GB de memória e 8 (oito) núcleos de processamento. Ambas as soluções foram configuradas para executar nesse ambiente em suas versões padrão, sem nenhuma configuração adicional. Talvez seja possível otimizar as ferramentas, mas não era o objetivo deste trabalho.

Figura 3.1: Propriedades da Máquina Virtual utilizada para executar as sincronizações

Fonte: Próprio Autor

A partir disso, realizou-se um total de 33 execuções com cada uma das ferramentas consideradas, processando um volume de 125,000 (cento e vinte e cinco mil) NFC-es em cada lote. Tal tamanho de lote foi escolhido, porque as aplicações que utilizam a Plataforma de Integração de Dados da SEFAZ-PB atualmente necessitam de uma sincronização das NFC-es a cada 3 (três) horas, de modo a dividir o volume diário de 1.000.000 (1 milhão) em 8 (oito) partes de 125.000 (cento e vinte e cinco mil).

Uma das métricas de avaliação coletadas foi o tempo de execução da sincronização do lote em relação ao documento fiscal (NFC-e) e os registros extraídos desse documento XML que serão carregados nas tabelas da base de destino. Nesse contexto, um registro é considerado como a combinação de um conjunto de dados em relação ao documento fiscal em si (chamado de fatonfe) e um conjunto de dados em relação aos itens vendidos presentes neste mesmo documento (chamado de fatoitemnfe), já que se trata de operações mercantis [33, 34]. Essa métrica foi escolhida para identificar se as soluções consideradas conseguiriam processar o volume diário de NFC-e exposto anteriormente dividido em intervalos menores e mais frequentes.

Além disso, outra métrica coletada foi em relação a taxa de erros de formatação identificados nos documentos NFC-e presentes no lote. Essa métrica foi escolhida por ser um aspecto relevante para tomada de decisão por gestores da SEFAZ em relação a validação de informações do documento fiscal eletrônico emitido.

Os testes em si foram executados de forma sequencial, em que um lote utilizado para sincronização de dados apresentava documentos NFC-e com valores dos seus atributos identificadores maiores do que os contidos no lote utilizado no teste anterior, formando uma sequência das n próximas notas que deveriam ser processadas, em ordem crescente.

4. DESCRIÇÃO DOS SISTEMAS

Neste capítulo são apresentadas as 2 soluções utilizadas no estudo.

4.1. SISTEMA LOADERNFE

A SEFAZ-PB é um órgão da administração pública responsável por gerenciar e fiscalizar o cumprimento da legislação tributária no contexto da esfera estadual. A UFPB vem desenvolvendo desde 2020 uma Plataforma de Integração de Dados dos sistemas atuais da SEFAZ-PB, incluindo a unificação das bases de dados num *Data Lake*. Dentre as várias aplicações, podemos citar a exploração de grandes volumes de dados de documentos fiscais eletrônicos, como fiscais eletrônicas de serviço (NF-es), notas fiscais eletrônicas do consumidor (NFC-es) e Escriturações Fiscais Digitais (EFD).

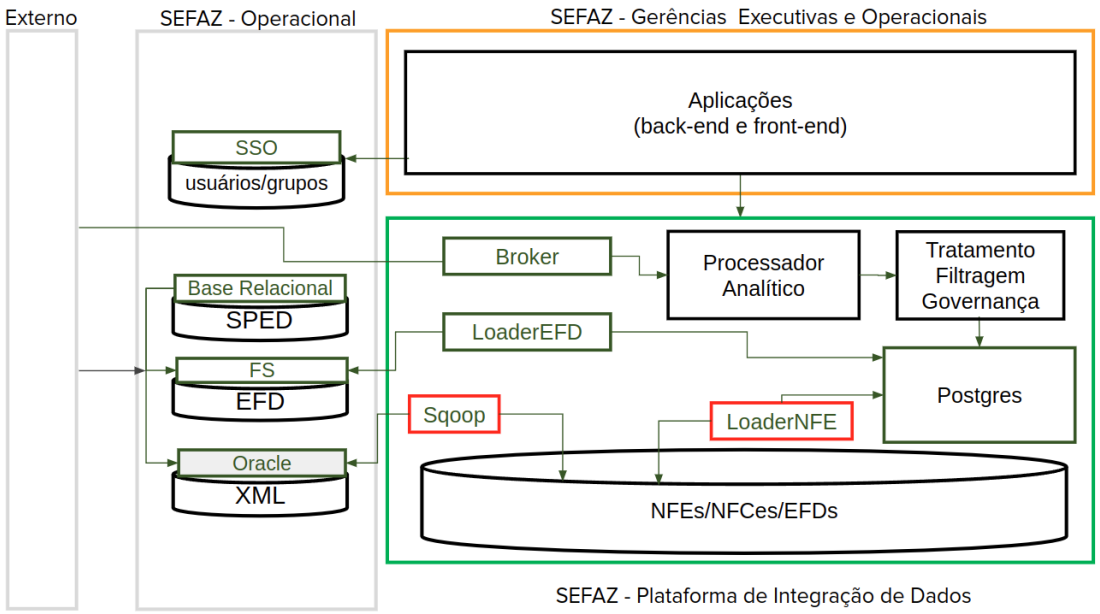

Figura 4.1: Arquitetura lógica da Plataforma de Integração de Dados da SEFAZ-PB

Fonte: Próprio Autor

A arquitetura lógica atual da solução é descrita na Figura 4.1, onde é possível perceber que são recebidos dados de sistemas externos para três bases operacionais da

SEFAZ-PB: i) base relacional operacional da SEFAZ-PB que recebe e armazena todos os documentos fiscais do sistema SPED localizado na SEFAZ do Rio Grande do Sul e é utilizado pelas aplicações atuais da SEFAZ-PB ; ii) armazenamento das EFDs num sistema de arquivo (FS) e; iii) armazenamento dos documentos XML no formato original num SGBD (Oracle). Adicionalmente, também existe na SEFAZ operacional um sistema de autenticação único (do inglês, Single Sign On) que armazena usuários e grupos. Na área verde é possível observar os subsistemas relacionados à Plataforma de Integração de Dados: 1) Sqoop - ferramenta da Apache que realiza a extração de dados da base Oracle e armazena os documentos fiscais que estão no formato XML numa base HFDS (NFEs/NFCes/EFDs); 2) LoaderNFE - ferramenta desenvolvida no projeto para realizar as transformações nos arquivos XML da base HDFS e carga num SGBD Postgres; 3) LoaderEFD - ferramenta desenvolvida no projeto para executar a leitura, transformação e carga de arquivos armazenados no formato TXT de EFDs para o SGBD Postgres; 4) Broker - recebe eventos de outros sistemas externos, por exemplo, eventos de registro de passagem de veículos; 5) Processador Analítico - realiza consultas mais complexas e demoradas; 6) Tratamento, filtragem e governança - permite o acesso a base de dados por meio de uma API Graphql com requisitos de segurança, escalabilidade e desempenho. Já na área laranja é possível observar a camada de aplicações, onde os usuários finais das gerências da SEFAZ-PB têm acesso às funcionalidades por meio front-ends Web que acessam a Plataforma de Integração de Dados por um back-end.

Neste trabalho, o principal objetivo foi entender o LoaderNFE e estudar alternativas. O LoaderNFE é baseado no processo ETL e é responsável por realizar o carregamento de dados relacionados a documentos fiscais como a NF-e e a NFC-e, que se encontram no formato HDFS, para o *Data Lake* SEFAZ-PB. Não obstante, escrito principalmente utilizando a linguagem Go[3], também trata-se de uma ferramenta de código aberto (licença MIT).

Ele pode ser dividido logicamente em 3 (três) módulos:

- **Processamento:** lida com o processamento das notas fiscais XML presentes num arquivo (ou conjunto de arquivos) HDFS; Possui uma dependência

[3] Linguagem Go: Disponível em: <https://go.dev/>.

externa implícita com o Apache Sqoop[4], pois é este quem importa os dados da base de Origem dos documentos XML em questão, um banco de dados relacional Oracle, para o sistema de arquivos distribuídos do Hadoop (HDFS). Além disso, possui dependências com bibliotecas internas padrão da linguagem utilizadas no processamento de arquivos como as bibliotecas: `path, os e regexp`.

- **Carga:** lida com a carga de determinados dados (eventos/acontecimentos) relacionados aos XML processados. Possui dependência com o `pgx`, um *driver* e *toolkit* escrito na linguagem de programação Go para o banco de dados Postgres. Essa dependência é importante não só para se conectar ao *Data Lake* e armazenar os dados relacionados aos documentos fiscais, como também para gerar esses dados, tirando proveito de funções internas do Postgres para processamento de XML que implementam a avaliação de expressões XPath 1.0.
- **Estatísticas:** trata de estatísticas de execução como tempo total de processamento, quantidade de documentos processados, total de documentos com problemas, etc. Possui dependências com a biblioteca interna `time`, para medição do tempo de execução, e a biblioteca externa `uiprogress`, para renderização de barras de progresso em aplicações de console.

Visando uma comparação mais exata, é importante levar em consideração o aspecto da dependência do LoaderNFE com o Apache Sqoop, pois, o tempo total de execução do LoaderNFE será a soma do tempo de importação dos dados da base de Origem para o sistema de arquivos distribuídos do Hadoop (HDFS) com o tempo da abordagem ETL utilizada para processar os arquivos HDFS e Carregar os dados relacionados a NFC-e no *Data Lake* Postgres. A Figura 4.2 mostra um exemplo de importação desse tipo.

[4] Apache Sqoop. Disponível em: <https://sqoop.apache.org/>.

```
sqoop import --connect jdbc:oracle:thin:[redacted] --username  -password  --query "SELECT t.SQNFECONSUMO,
substr(t.DSNFCEXML.getClobVal(), 0) as DSNFCEXML from DF_XML.TBFIS_NFCEXML t where \$CONDITIONS and SQNFECONSUMO > 1991938000 AND
SQNFECONSUMO<= 1992976700" --target-dir /01992976700NFCE --delete-target-dir -m 96 --temporary-rootdir /data/hdfs --split-by SQNFECONSUMO --map-
column-java "DSNFCEXML=String"

2022-06-23 11:15:27,742 INFO mapreduce.ImportJobBase: Transferred 12.4857 GB in 797.6793 seconds (16.0282 MB/sec)
2022-06-23 11:15:27,745 INFO mapreduce.ImportJobBase: Retrieved 1038377 records.
```

Figura 4.2: Exemplo de importação de lote NFC-e da base de Origem para o Sistema de arquivos distribuídos do Hadoop (HDFS)

Fonte: Próprio Autor

4.2. Sistema Airbyte

Airbyte[5] é uma *startup* estadunidense, com sede em São Francisco, que trabalha com integração de diferentes tipos de dados. Com seu produto homônimo de código aberto, a empresa tem como objetivo movimentar e sincronizar dados entre APIs, SGBDs, *Data Warehouses*, *Data Lakes* e outros destinos visando simplicidade e eficiência. Construído com uma arquitetura de microsserviços, é possível configurar e executar pipelines de dados baseados na abordagem ELT.

4.2.1. Diagrama de contexto

Com o Airbyte, que é uma ferramenta ELT voltada para lidar com as etapas de EL (Extrair-Carregar), o usuário consegue se conectar e Extrair dados de várias fontes de dados em sistemas externos, além de carregar e transformar esses dados (com dbt) em vários destinos em outros sistemas externos (ver Figura 4.3).

[5] Airbyte. Disponível em: <https://airbyte.com/>.

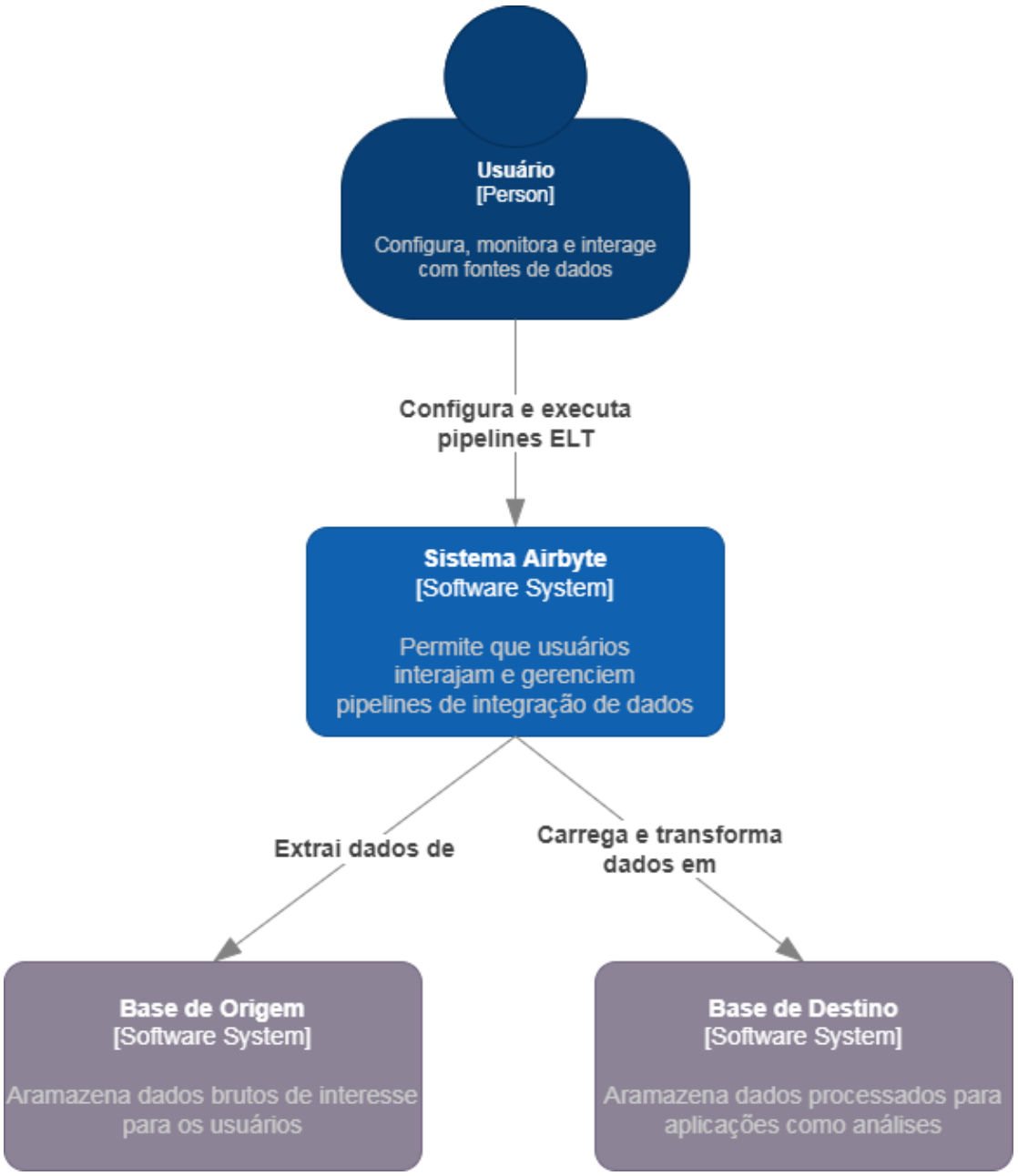

Figura 4.3: Diagrama de contexto do Sistema Airbyte

Fonte: Elaborada pelo autor

4.2.2. Diagrama de container

A Figura 4.4 mostra um diagrama com mais detalhes do sistema, descrevendo os seus containers.

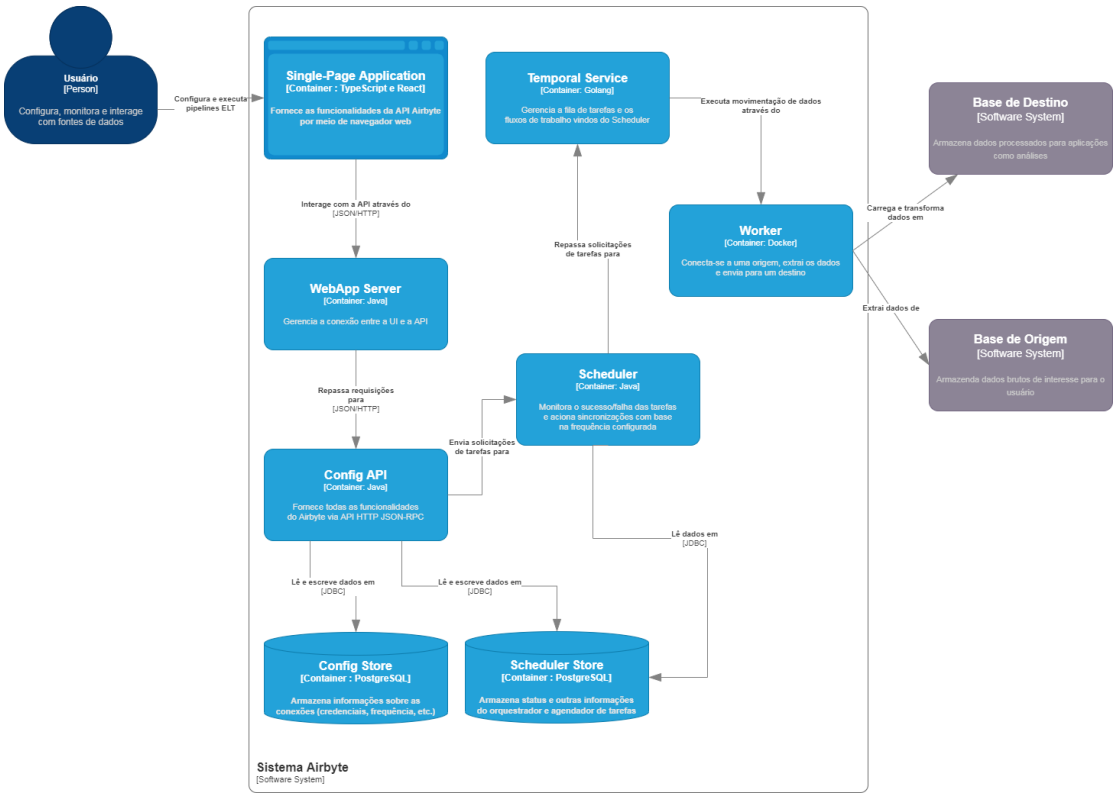

Figura 4.4: Diagrama de container do Sistema Airbyte

Fonte: Elaborada pelo autor

A interface gráfica permite que o usuário interaja com a API Airbyte, porém não diretamente. Por isso, o WebApp Server existe para gerenciar a conexão entre a interface de usuário e a API. O Config API é o controlador geral do sistema e é responsável por configurar e invocar todas as operações fornecidas pelo Airbyte como criação de origens, destinos, conexões, gerenciamento de configurações, etc. O `Scheduler` é quem recebe as solicitações de tarefa da API e é responsável por monitorar o sucesso/falha de sua execução e por acionar a sincronização dos dados de acordo com a frequência configurada. A partir disso, as solicitações são enviadas para o `Temporal Service`, que é quem gerencia a fila de tarefas e os fluxos de trabalho, deliberando sobre questões de paralelização, etc. Já o Worker é quem de fato executa a movimentação de dados, interagindo com conectores de origem e destino, extraindo os dados de uma fonte e gravando numa outra, onde ambas são sistemas externos (ver Figura 4.4).

4.2.3. Diagrama de componente

Como mencionado anteriormente, o módulo Worker é quem de fato executa as tarefas da fila gerenciada pelo `Temporal Service`, o que representa a interação de um Conector. A Figura 4.5 mostra o diagrama de componentes deste módulo.

Conectores são módulos independentes que extraem dados de uma origem e enviam para um destino. Estes são construídos de acordo com uma especificação bem definida que descreve protocolos de comunicação entre os processos e a interface com a qual os dados podem ser movidos entre uma origem e um destino usando Airbyte, tudo baseado em `JSON Schema`[6]. Como a especificação é baseada em Docker[7], isso permite total flexibilidade sobre as tecnologias usadas para implementar novos conectores. É a partir disso que surgem as definições de Conector de Origem e Conector de Destino.

Um Conector de Origem deve implementar a seguinte interface:

1. **spec() -> ConnectorSpecification**

O objetivo desse comando é descrever quais entradas são necessárias para que o Conector de Origem consiga interagir com a fonte de dados subjacente. Por exemplo, para uma fonte Postgres, `ConnectorSpecification` deveria descrever que hostname, port, e password são necessários para que o Conector funcione. A partir disso, a interface de usuário lê o JSON Schema retornado para renderizar os campos de entrada que o usuário deve preencher.

2. **check(Config) -> AirbyteConnectionStatus**

O objetivo desse comando é tentar se conectar à fonte de dados subjacente, verificando se as credenciais fornecidas pelo usuário são utilizáveis. Por exemplo, para uma fonte Postgres, o password informado pode determinar se a tentativa de conexão será bem-sucedida ou não (caso esteja incorreto).

3. **discover(Config) -> AirbyteCatalog**

---

[6] JSON Schema. Disponível em <https://json-schema.org/>.
[7] Docker. Disponível em: <https://www.docker.com/>.

O objetivo deste comando é descrever a estrutura dos dados da fonte de dados subjacente. Por exemplo, no caso de uma fonte de dados que é um banco de dados relacional tradicional, cada mapeamento deveria descrever a(s) tabela(as) e suas respectivas colunas, refletindo os dados que serão considerados durante a etapa de extração. Por outro lado, caso a fonte de dados trata-se de uma API, cada mapeamento deveria descrever os recursos retornados por cada rota (por exemplo api/customers e api/products). O Airbyte denomina essa estrutura de stream.

4. **read(Config, ConfiguredAirbyteCatalog, State) -> Stream<AirbyteMessage>**

O objetivo desse comando é, antes da etapa de extração propriamente dita, ler os dados da fonte de dados subjacente e convertê-los num formato padrão (`AirbyteRecordMessage`) exigido pelo protocolo de comunicação para que seja possível a transferência e eventual replicação dos dados entre os conectores de Origem e Destino. Idealmente, o Conector de Origem apenas extrai os dados que estão representados pelo parâmetro `ConfiguredAirbyteCatalog` que por sua vez é construído a partir de um subconjunto do `AirbyteCatalog` retornado pelo comando anterior (discover) que foi mantido selecionado pelo usuário na interface gráfica a partir de caixas de seleção. Além disso, é possível emitir estados para que, na próxima vez que o Conector de Origem tentar extrair dados, ele disponha de uma forma de verificar o que aconteceu na sincronização de dados anterior. Esse é o conceito que permite sincronizações incrementais apresentado na subseção 4.2.4.

Já um Conector de Destino deve implementar a seguinte interface:

1. **spec() -> ConnectorSpecification**
2. **check(Config) -> AirbyteConnectionStatus**

O objetivo dos comandos spec e check do Conector de Destino são exatamente os mesmos descritos para o Conector de Origem.

3. **write(Config, AirbyteCatalog, Stream<AirbyteMessage>(stdin)) -> void**

O objetivo desse comando é ler o fluxo de mensagens recebido e gravar na fonte de dados subjacente apenas os `AirbyteRecordMessage` que correspondam a estrutura descrita pelo parâmetro `AirbyteCatalog`.

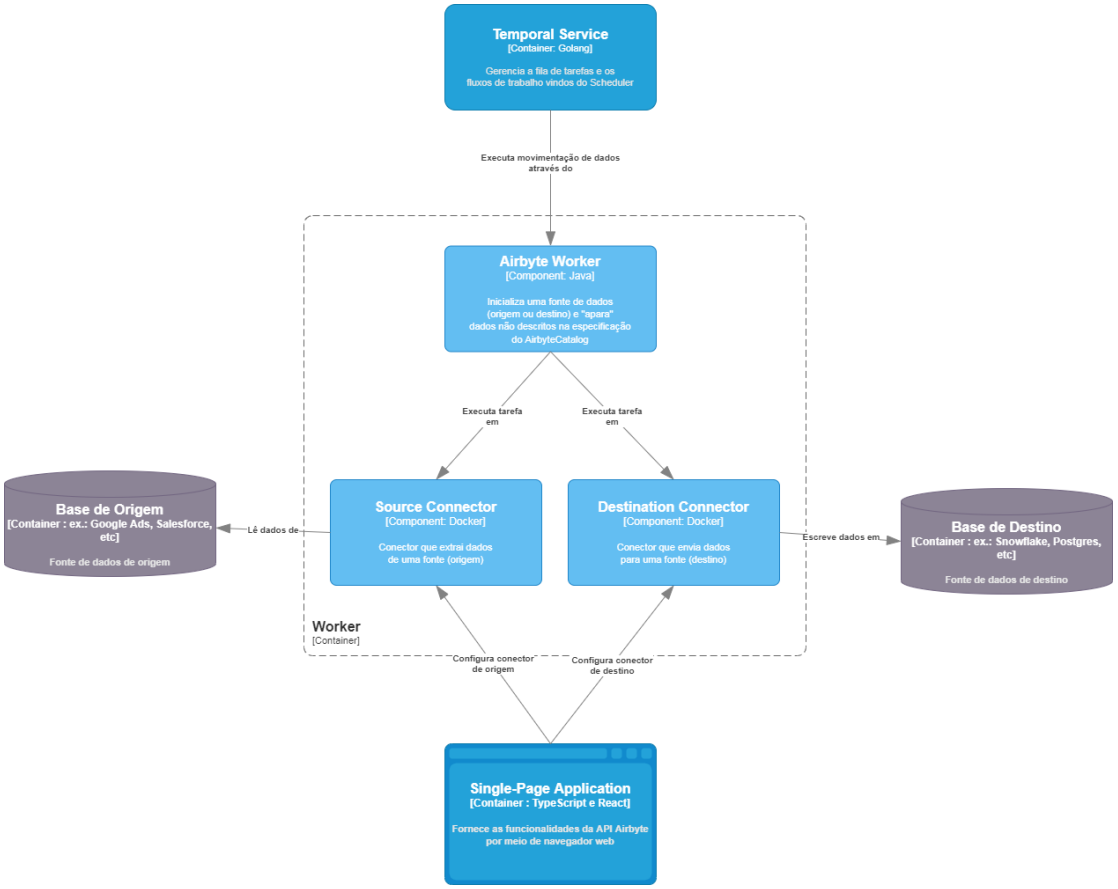

Figura 4.5: Diagrama de componente do Sistema Airbyte

Fonte: Elaborada pelo autor

A partir disso, todas as interações com Conectores que ocorrem no Airbyte são executadas como tarefas por um Worker. A seguir são apresentados exemplos de Workers:

- *Spec worker*: recupera a especificação de um Conector (as entradas necessárias para executar este Conector)
- *Check* connection worker: verifica se as entradas para um Conector são válidas e podem ser usadas para executar a sincronização
- *Discovery worker*: recupera o esquema da fonte de origem subjacente ao Conector
- *Sync worker*: usado para sincronizar dados entre uma origem e um destino

Existem Workers que interagem com um único Conector (ex.: spec, check, discover) e Workers que interagem com 2 Conectores (ex.: sync, reset), porém ambos tem 4 responsabilidades principais em seu ciclo de vida:

1. Iniciar todos os containers Docker dos Conectores necessários para a tarefa.
2. Facilitar a passagem de mensagens de ou para o container de um Conector.
3. Retornar a saída da tarefa.
4. Encerrar todos os containers que ele iniciou.

A Figura 4.6 mostra o ciclo de vida de um Worker que interage com 2 Conectores.

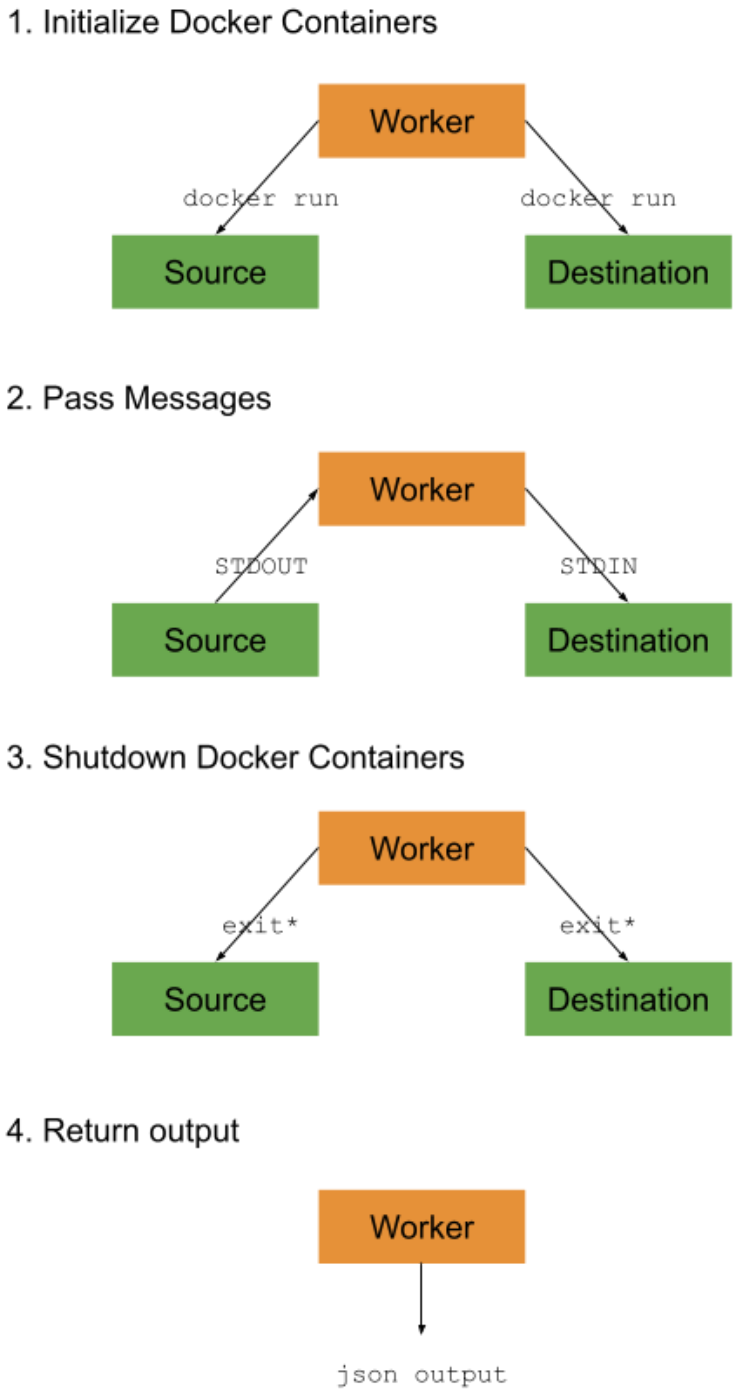

Figura 4.6: Ciclo de vida de um Worker que interage com 2 Conectores

Fonte: Documentação do Airbyte [30]

Já as tarefas executadas pelos Workers podem ser representadas pela máquina de estado da Figura 4.7.

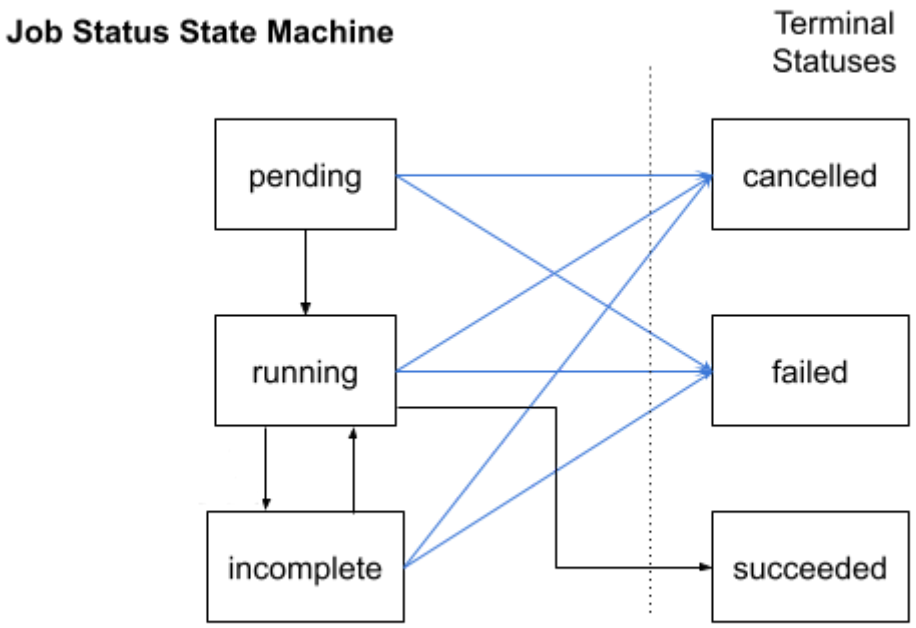

Figura 4.7: Máquina de estado das tarefas executadas pelos Workers

Fonte: Adaptado de [30]

Quando uma tentativa de sincronização falha, o status da tarefa passa para o estado incompleto. A partir disso, é realizada uma nova tentativa de sincronização, voltando para o estado de execução. No total, há 3 tentativas no caso de falhas sucessivas sendo que, depois da última tentativa a tarefa acaba no estado final de falha.

4.2.4. MODOS DE SINCRONIZAÇÃO

No Airbyte, um modo de sincronização controla como os dados devem ser replicados (leitura e gravação). Uma sincronização será acionada numa das circunstâncias a seguir: a partir de uma solicitação manual clicando no botão "Sync Now" na interface do usuário; a partir do agendador, em que uma sincronização é executada sempre que o tempo decorrido desde a última sincronização (seja ela acionada manualmente ou devido a um agendamento) tenha excedido o intervalo configurado pelo usuário (ex.: a cada 2 horas).

O comportamento de um modo é refletido em seu nome, que é composto por duas partes:

1. A primeira parte do nome denota como o Conector de Origem lê os dados da fonte de origem, que pode ser uma das formas a seguir:
    - *Incremental*: lê apenas os dados novos ou modificados adicionados à origem desde a última tarefa de sincronização.
    - *Full Refresh*: lê tudo da fonte de origem.
2. A segunda parte do nome denota como o Conector de Destino grava os dados, que pode ser uma das formas a seguir:
    - *Overwrite*: Sobrescreve os dados existentes no destino, primeiro excluindo os que já existem e depois gravando os dados recebidos.
    - *Append*: Grava apenas anexando os dados recebidos às tabelas existentes no destino, o que pode acarretar duplicação.
    - *Deduped History*: Utiliza uma tabela intermediária adicional para manter um histórico de alterações e a partir disso a tabela final é produzida desduplicando os intermediários com o auxílio de uma chave primária.

Sendo assim, um modo de sincronização é uma combinação de um modo de origem e destino juntos. A partir disso, a interface do usuário pode exibir as seguintes opções, sempre que os conectores de origem e destino oferecerem suporte:

- *Full Refresh Overwrite*: Sincroniza tudo e substitue os dados na fonte de destino sobrescrevendo-os.
- *Full Refresh Append*: Sincroniza tudo e anexa dados no destino.
- *Incremental Append*: Sincroniza novos registros e anexa dados no destino.
- *Incremental Deduped History*: Sincroniza novos registros e anexa dados no destino, também fornecendo uma exibição desduplicada que espelha o estado dos dados na origem.

4.2.5. Transformações no Airbyte

Como explicado anteriormente, o Airbyte é uma ferramenta ELT voltada para lidar com as etapas de EL (Extrair-Carregar). A partir disso, os dados são gravados na fonte de destino numa tabela bruta que segue a nomenclatura `_airbyte_raw_<stream name>` na qual os dados propriamente ditos são armazenados numa coluna de formato JSON. Essa tabela também contém algumas outras colunas, caracterizadas como colunas de metadados porque são adicionadas para acompanhar informações importantes sobre cada registro como hora em que o registro foi emitido e gravado pelo Conector de Destino e, em casos específicos (*Deduped History*), também é adicionado colunas com metadados que servirão para questões de histórico de modificações e desduplicação da tabela final. De acordo com o Airbyte, essa tabela é criada porque um princípio central da filosofia ELT é que os dados devem permanecer intocados à medida que passam pelos estágios E e L, para que os dados brutos estejam sempre acessíveis. Sendo assim, se existir uma versão não modificada dos dados no destino, ela poderá ser transformada novamente sem a necessidade de sincronizar os dados novamente [24].

Porém, para casos típicos de análise, provavelmente será necessário que essa coluna JSON seja normalizada [12, 13 e 14]. O Airbyte configura por padrão a aplicação desse processo na etapa T (Transformação), que é chamada de Normalização Básica. Das implementações para a etapa T elencadas na seção 2.2, atualmente, essa opção de normalização é implementada usando o que é chamado

pelo Airbyte de *dbt Transformer*, como apresentado na Figura 4.8, que apresenta exemplos de implementação utilizadas pelo Airbyte em cada uma das etapas do processo ELT.

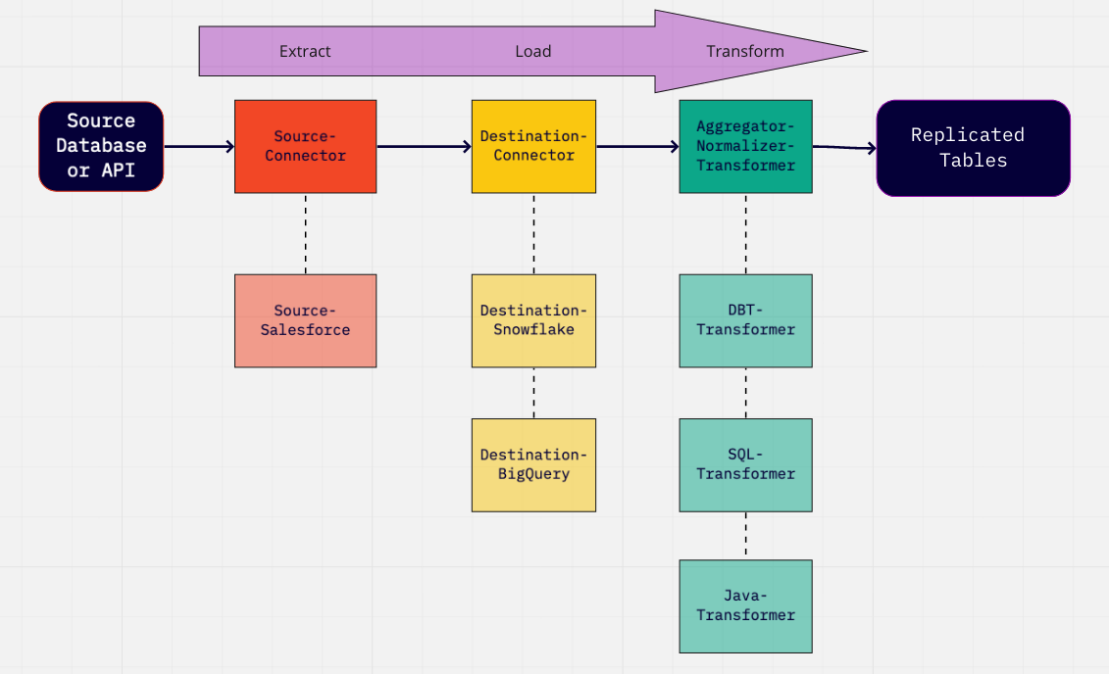

Figura 4.8: Exemplos de implementação utilizadas pelo Airbyte em cada uma das etapas do processo ELT

Fonte: Documentação do Airbyte [15]

O dbt Transformer em questão é empacotado numa imagem Docker da seguinte forma:

- Pacote Python contendo normalização baseada num conjunto fixo de regras consideradas pelo Airbyte para gerar modelos SQL no dbt.
- dbt utilizado para compilar e executar os modelos sobre os dados nas fontes de destino que o suportam.

Dessa forma, o objeto JSON é mapeado para os tipos e formatos nativos da fonte de destino, baseado nos tipos descritos no AirbyteCatalog.

Além disso, a própria equipe da Airbyte reconhece que a normalização é um problema orientado a detalhes e que, com um conjunto fixo de regras, não é possível

normalizar os dados de forma a cobrir todos os casos de uso necessários [15]. Por isso, é possível desabilitar a opção de Normalização Básica e configurar a etapa de Transformação de maneira customizada podendo implementar um *Transformer* em diferentes linguagens de programação, criá-los em um mecanismo de análise como o Apache Spark[8] ou utilizar uma ferramenta de transformação como o dbt [12, 15].

4.3. Solução Alternativa utilizando o Airbyte

A solução construída é um Conector de Origem customizado para o Airbyte utilizando um SDK (do inglês, *Software Development Kit*) específico para a linguagem de programação Go[9]. O SDK em questão foi desenvolvido pela Bitstrapped [10], uma empresa canadense focada em infraestrutura para dados em nuvem com GCP (do inglês, *Google Cloud Platform*). Visando agilidade durante o desenvolvimento, o SDK foi construído de modo que os desenvolvedores possam se concentrar na lógica de negócio e não ter que se preocupar excessivamente com o protocolo Airbyte. Além disso, para realizar o processamento das notas fiscais XML foi utilizado um pacote `XPath` para a linguagem Go chamado `xmlquery`[11]. Já para a etapa de Transformação foi construído um dbt Transformer.

Cabe também ressaltar as versões utilizadas:

- Airbyte versão v0.35.32-alpha
- Imagem Docker Go (golang) versão 1.17-buster
- SDK versão v0.0.6
- xmlquery versão v1.3.10
- Imagem Docker dbt (fishtownanalytics) versão 0.19.1

Sendo assim, a solução é responsável por realizar o carregamento de dados relacionados aos documentos fiscais NFC-e, que se encontram diretamente em sua base de origem, um banco de dados relacional Oracle, para um *Data Lake* Postgres.

---

[8] Apache Spark. Disponível em: <https://spark.apache.org/>.
[9] Airbyte Golang SDK. Disponível em: <https://github.com/bitstrapped/airbyte>.
[10] Bitstrapped. Disponível em: <https://www.bitstrapped.com/>.
[11] xmlquery. Disponível em: <https://github.com/antchfx/xmlquery>.

Na implementação da função spec, é retornado um JSON Schema (ver parte esquerda da Figura 4.9) que representa as entradas necessárias para se conectar ao SGBD Oracle, que são os campos password, username e `connect_string`. Além disso, 2 campos adicionais são exigidos para declarar como a leitura dos dados será executada inicialmente, que são os campos `sqn_initial_value`, que representa o valor do atributo identificador da primeira nota fiscal que será buscada, e first_rows, que representa um lote das n notas fiscais que também serão trazidas da base de dados, sendo que os valores do atributo em questão são do tipo inteiro e formam uma sequência crescente.

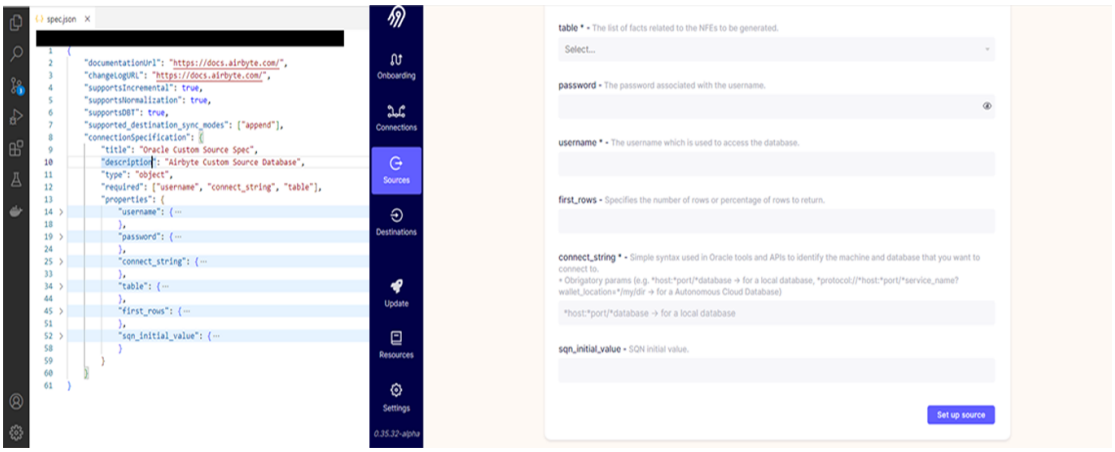

Figura 4.9: Exemplo de resultado da implementação da função spec do protocolo Airbyte

Na implementação da função check é recuperado as entradas preenchidas pelo usuário no formulário, necessárias para se conectar ao SGBD Oracle (ver Figura 4.10) e realizado uma tentativa de conexão ao banco. Caso a tentativa seja bem-sucedida, será apresentado um símbolo de *check* com uma coloração verde (ver parte inferior da Figura 4.10).

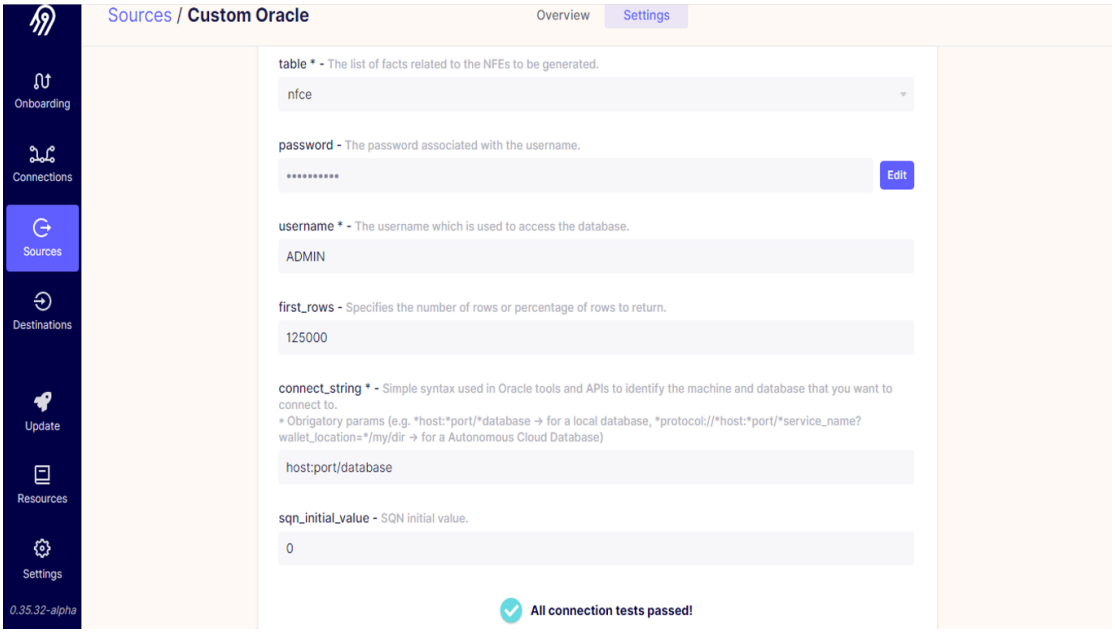

Figura 4.10: Exemplo de resultado da implementação da função check do protocolo Airbyte

Na implementação da função discover, é retornado um `JSON Schema` (ver parte esquerda da Figura 4.11) que representa as estruturas dos streams que poderão ser lidos (`AirbyteCatalog` apresentado na subseção 4.2.3). Além disso, é possível deixar selecionadas (ver parte inferior direita da Figura 4.11) apenas as caixas de seleção referentes aos streams desejados pelo usuário que de fato serão processados durante a etapa de Extração (`ConfiguredAirbyteCatalog` apresentado na subseção 4.2.3).

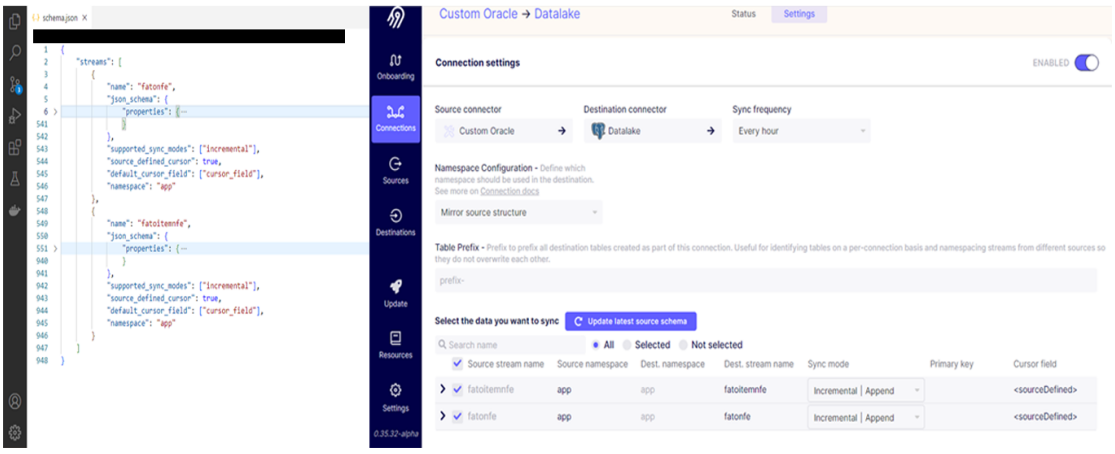

Figura 4.11: Exemplo de resultado da implementação da função discover do protocolo Airbyte

Na implementação da função read, suas primeiras linhas representam a recuperação do estado (State apresentado na subseção 4.2.3) da última sincronização de dados realizada. As linhas seguintes contém a conexão com o banco de dados, a partir das entradas fornecidas pelo usuário (ver exemplo na Figura 4.10). Depois é realizado uma consulta a base de dados para buscar as NFC-es que serão processadas, sendo que na primeira execução é utilizado o próprio valor de entrada fornecido pelo usuário no campo `sqn_initial_value` para identificar qual o valor referente ao atributo identificador da primeira nota que será buscada e a sequência das n próximas notas que também serão retornadas, em ordem crescente. Da segunda sincronização em diante, é utilizado o valor emitido no estado referente ao atributo identificador da última NFC-e processada na sincronização anterior, possibilitando o conceito de sincronizações incrementais apresentado na subseção 4.2.4.

As próximas linhas da função read (ver parte esquerda da Figura 4.12) apresentam um laço de repetição para iterar sobre os streams selecionados (`ConfiguredAirbyteCatalog` apresentado na subseção 4.2.3) e processar devidamente, com o `xmlquery`, os conteúdos XML distintos de cada um deles (ver parte direita da Figura 4.12) e gerar os registros apropriados. Por fim, as últimas linhas da função read indicam a criação de um arquivo contendo informações sobre eventuais NFC-e com erros de formatação durante a sincronização em questão, logo em seguida é realizado um log com o tempo de processamento referente a etapa de Extração dessa mesma sincronização e, por último, é emitido um novo estado que representa o início do lote de NFC-es que será extraído na próxima sincronização.

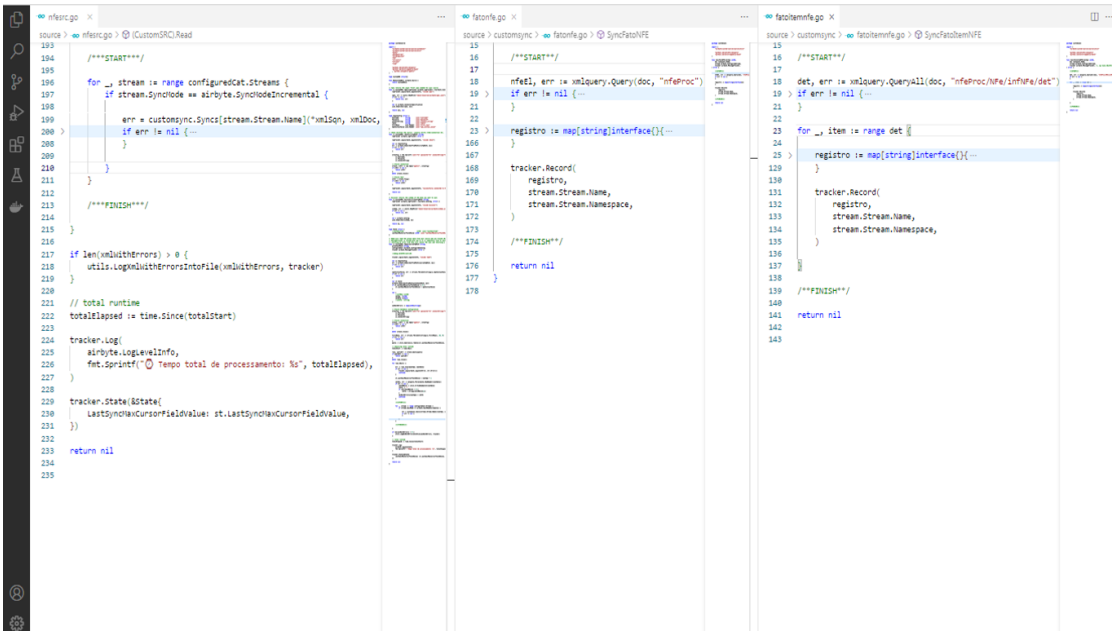

Figura 4.12: Exemplo de resultado da implementação da função read do protocolo Airbyte

A implementação da etapa de Transformação foi realizada a partir de um dbt Transformer (apresentado na subseção 4.2.5) customizado que mapeou o objeto JSON para os tipos apropriados a partir do operador CAST do Postgres (ver exemplo na Figura 4.13) e inseriu nas tabelas de destino corretas.

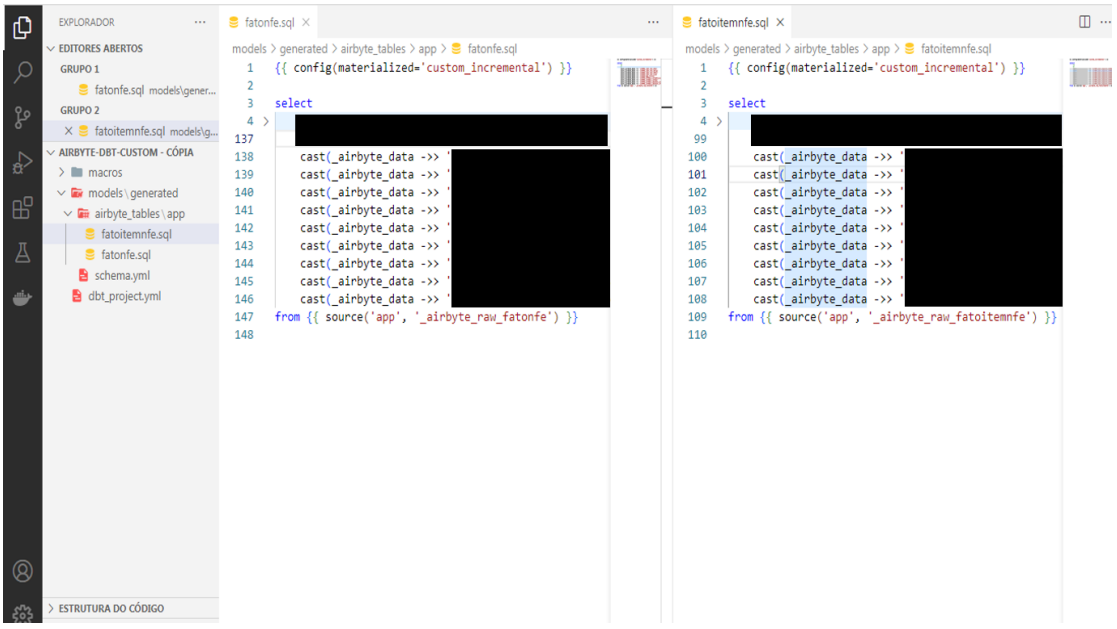

Figura 4.13: Exemplo do projeto dbt customizado utilizado no Airbyte

Por fim, para utilizar esse dbt Transformer customizado é necessário que o projeto dbt esteja num repositório git[12] [12], que no caso deste trabalho foi o Github[13], e configurar uma nova etapa de Transformação no Airbyte (ver exemplo na Figura 4.14).

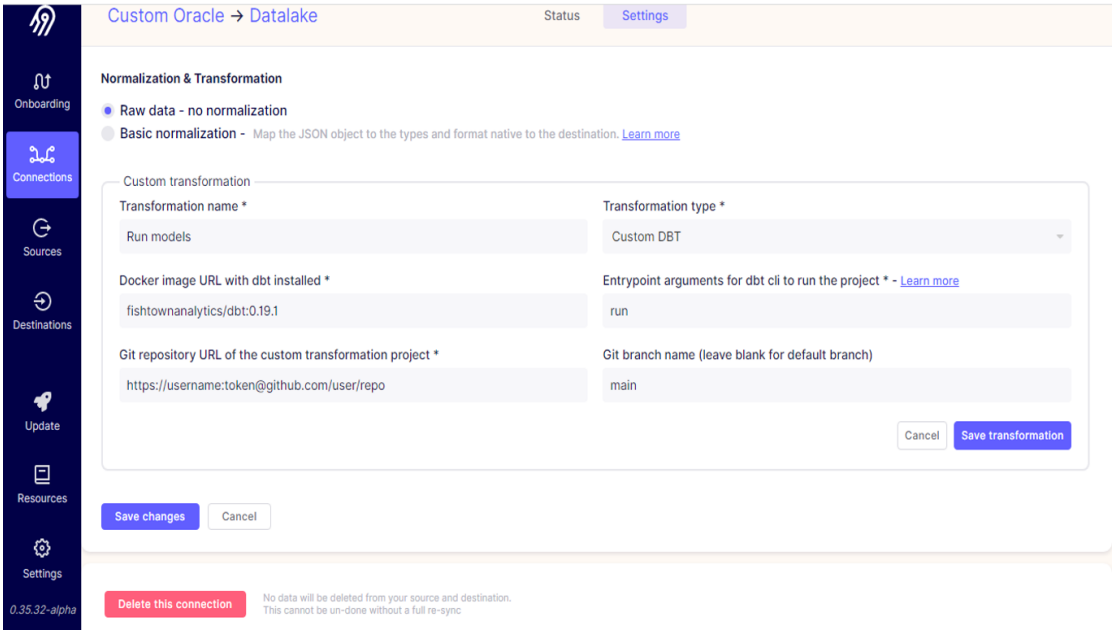

Figura 4.14: Exemplo de configuração do projeto dbt customizado na etapa de Transformação do Airbyte

[12] Git. Disponível em: <https://git-scm.com/>.
[13] GitHub. Disponível em: <https://github.com/>.

5. APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Ao término da execução dos testes, os dados foram coletados e as amostras resultantes foram utilizadas para a geração dos gráficos e tabelas. Nas subseções a seguir serão exibidos esses resultados para cada uma das soluções consideradas.

A Figura 5.1 mostra um gráfico com os resultados obtidos em relação ao tempo de processamento de cada um dos lotes na solução atual da SEFAZ-PB. Os picos no tempo de processamento, mais evidentes entre os lotes 10 e 16, indicam a quantidade maior de registros que precisaram ser processados por lote, como mostrado no gráfico da Figura 5.2. Isso significa que, no total, existiam mais itens por NFC-e naqueles lotes, o que refletiu no tempo maior necessário para processar o lote inteiro.

Já as Figuras 5.3 e 5.4 mostram os gráficos com os resultados obtidos, sob as mesmas perspectivas, para a solução com o Airbyte. Assim como no LoaderNFE, os picos no tempo de processamento também aconteceram nos mesmos lotes que apresentavam uma quantidade maior de registros.

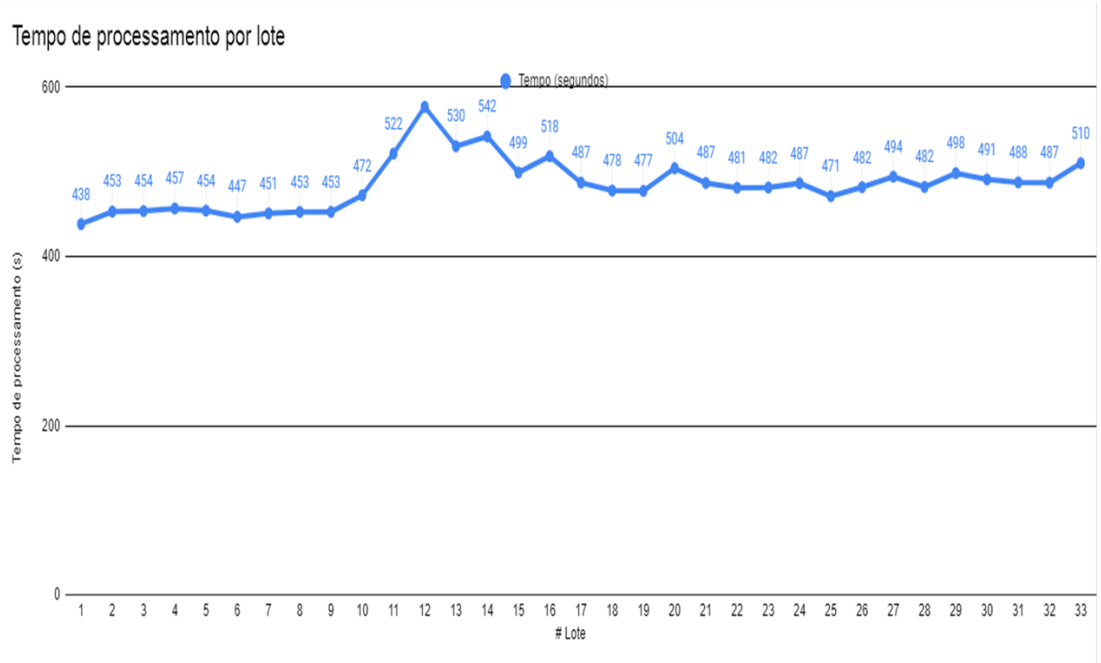

Figura 5.1: Gráfico com tempo de processamento por lote para o LoaderNFE

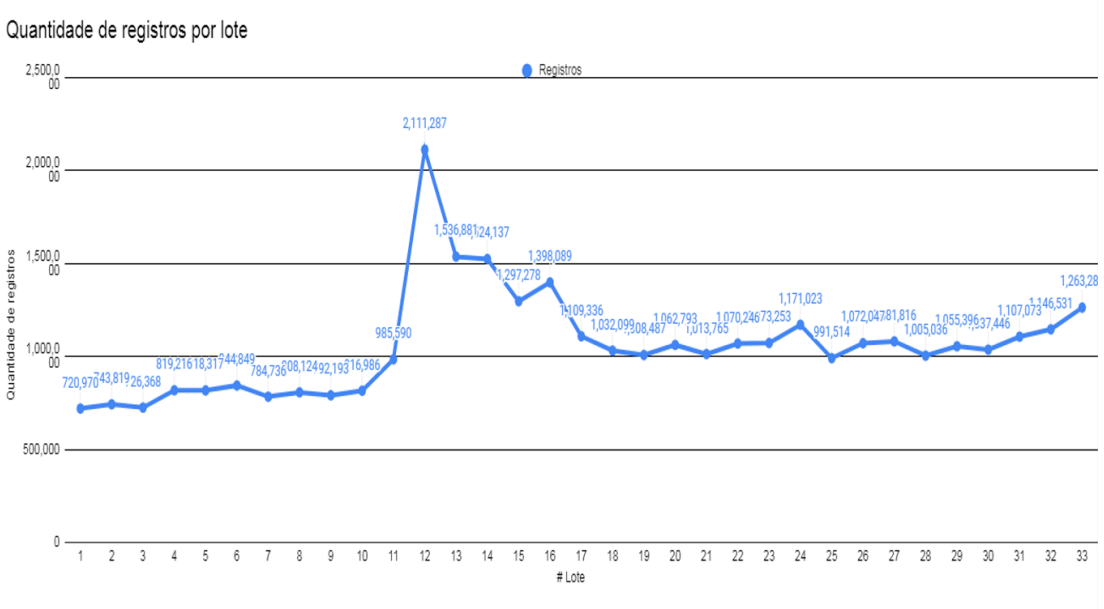

Figura 5.2: Gráfico com quantidade de registros processados por lote para o LoaderNFE

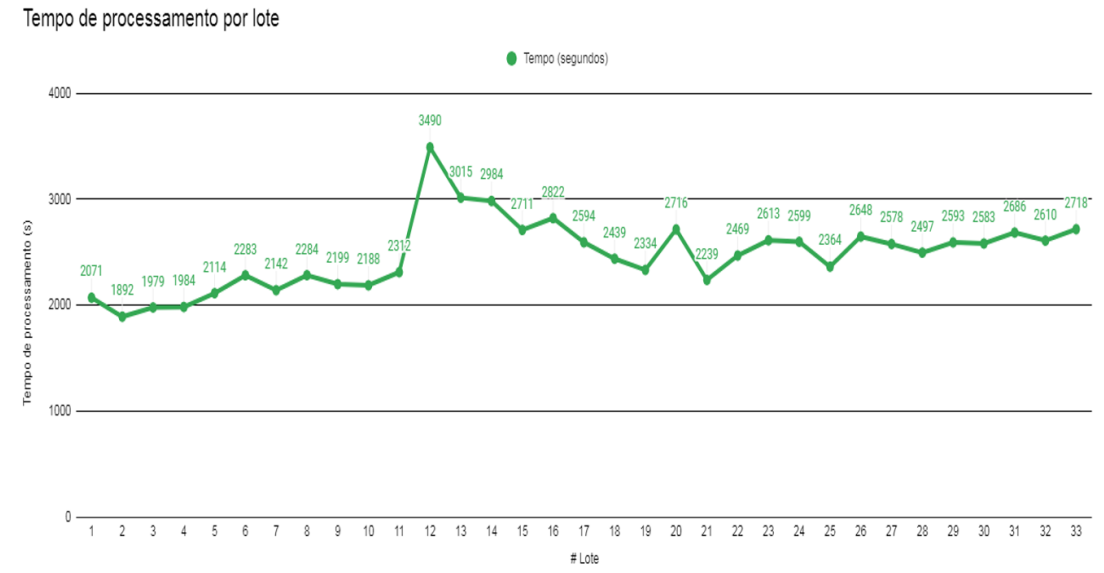

Figura 5.3: Gráfico com tempo de processamento por lote para o Airbyte

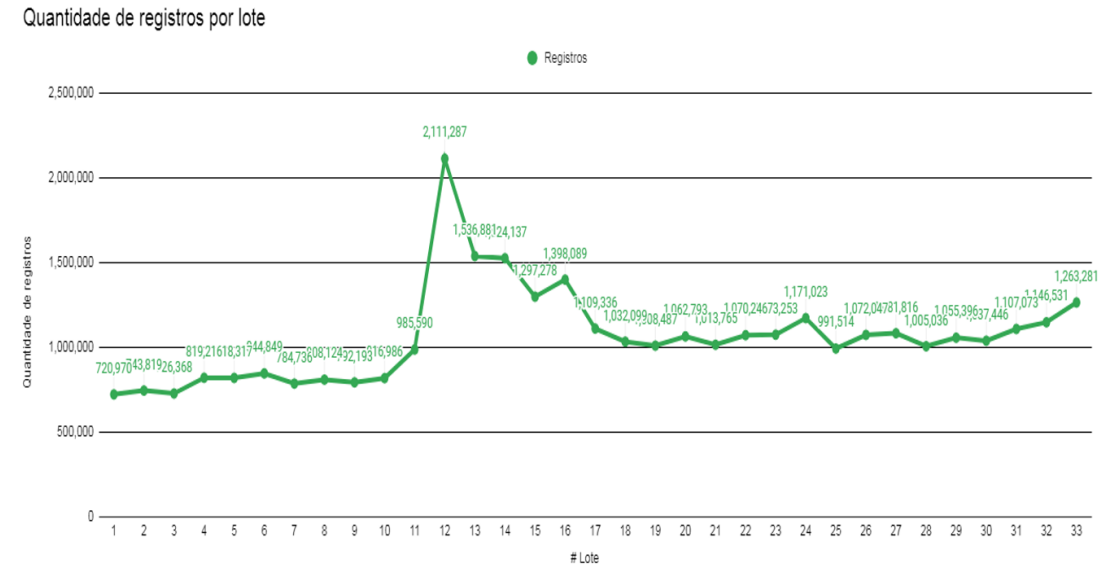

Figura 5.4: Gráfico com quantidade de registros processados por lote para o Airbyte

A Figura 5.5 exibe um gráfico com uma comparação dos resultados obtidos em relação aos tempos de processamento entre a solução atual da SEFAZ-PB e a solução utilizando o Airbyte. É possível observar que além de ser mais rápido, o LoaderNFE também apresenta um valor médio mais estável de tempo por lote, comparado com o Airbyte.

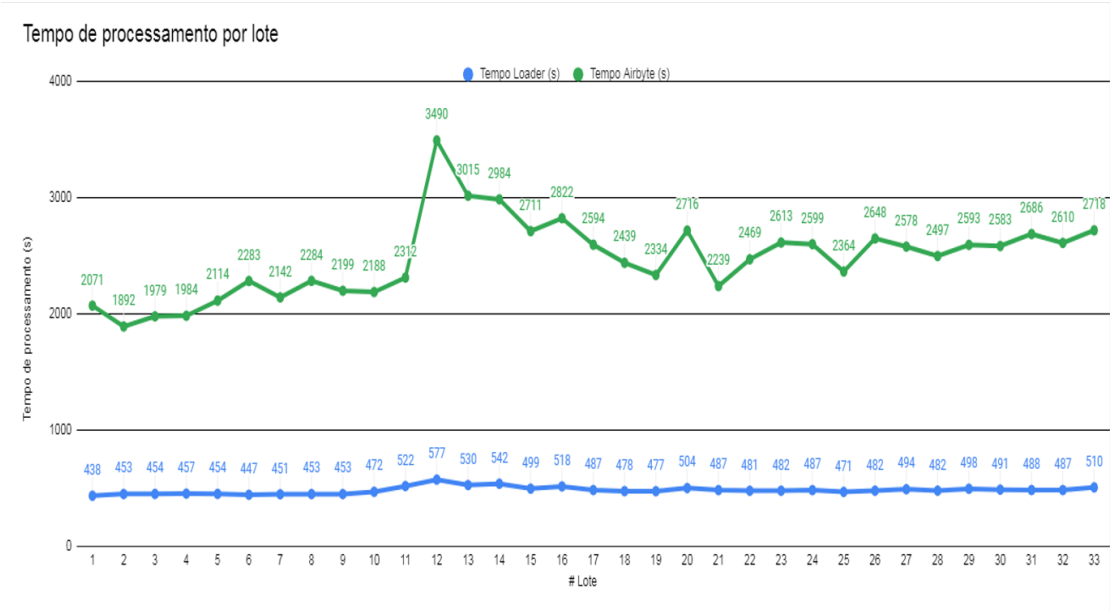

Figura 5.5: Gráfico com comparação dos tempos de processamento por lote entre o Airbyte (verde) e o LoaderNFE (azul)

Já as Figuras 5.6 e 5.7 mostram os gráficos com os resultados obtidos em relação aos erros identificados nas notas durante a etapa de Extração, respectivamente, na solução atual da SEFAZ-PB e na solução utilizando o Airbyte. O LoaderNFE identificou mais erros nos lotes finais dos testes e não identificou nenhum erro entre os lotes 10 e 16, que já foram chamados atenção anteriormente por apresentarem picos no tempo de processamento em ambas as soluções e também apresentavam um tamanho total em GB maior se comparado ao restante dos outros lotes.

De outra forma, a solução utilizando o Airbyte identificou uma quantidade maior de erros justamente entre os lotes 10 e 16. Porém, por questão de restrições de acesso à base de Origem das NFC-es não foi possível verificar qual solução apresentou o comportamento correto nesse aspecto, ou seja, qual solução identificou

corretamente a quantidade de erros em notas que existiam em cada lote utilizado nos testes.

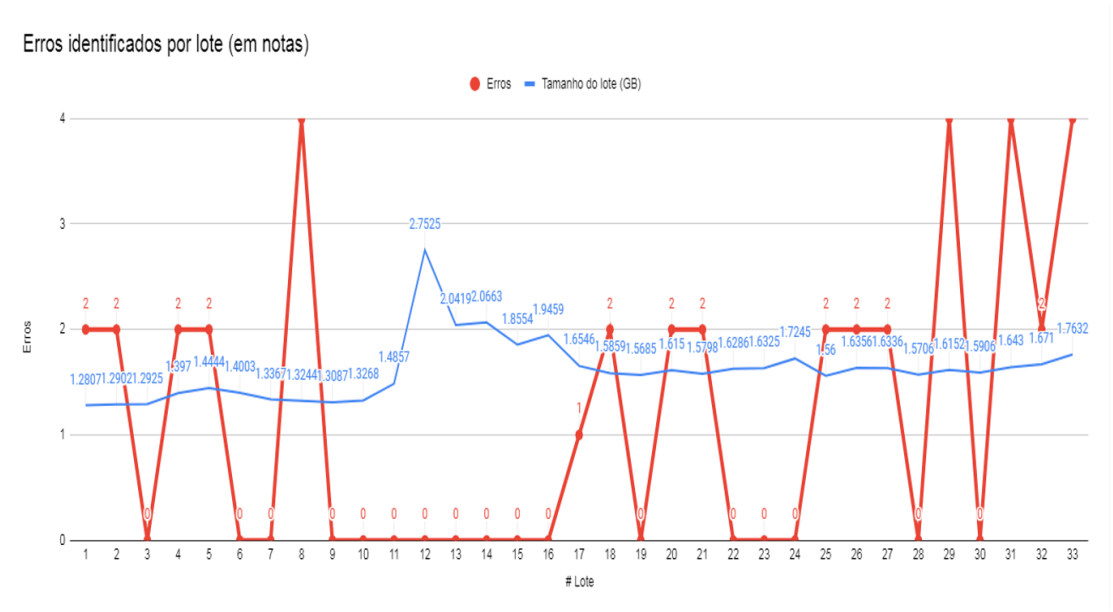

Figura 5.6: Gráfico com erros identificados por lote para o LoaderNFE

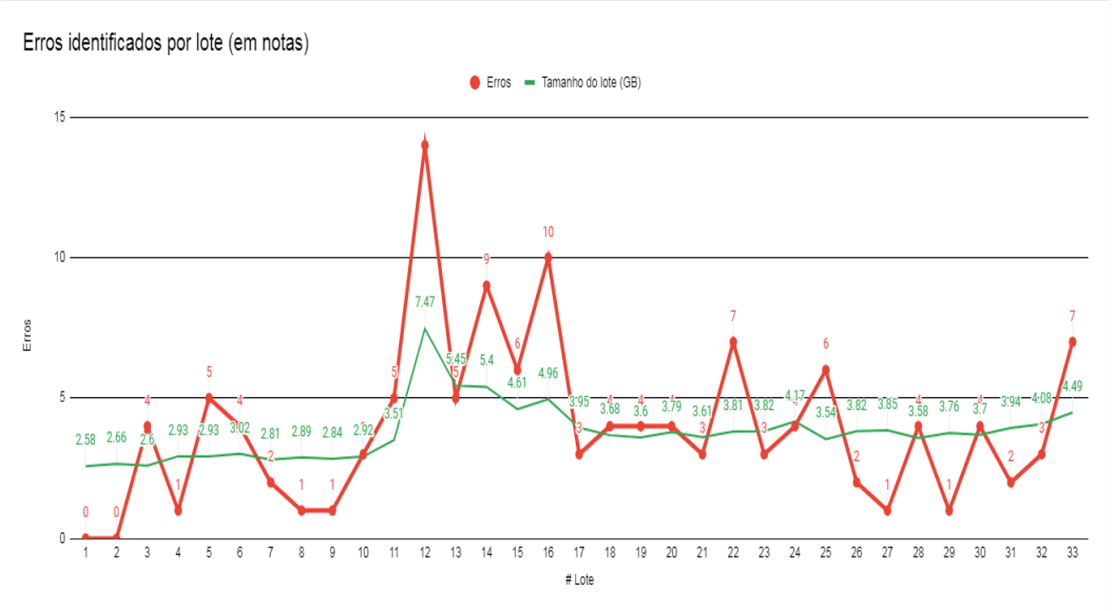

Figura 5.7: Gráfico com erros identificados por lote para o Airbyte

A Tabela 5.1 apresenta uma comparação entre as quantidades médias de notas processadas nos intervalos apresentados para o LoaderNFE e para o Airbyte. As médias em questão foram calculadas considerando-se a soma do total de notas processadas nais quais não foram identificados erros em cada um dos lotes dividido pela soma dos tempos de processamento dos lotes (em segundos).

Tabela 5.1: Comparação das quantidades médias de notas processadas nos intervalos apresentados entre o LoaderNFE e o Airbyte

	LoaderNFE	Airbyte
	Resultado	Resultado
Tempo de carga por Milhão (em minutos)	64,67	330,31
Notas por Segundo	257,70	50,46
Notas por Minuto	15.462,09	3.027,43
Notas por Hora	927.725,48	181.645,56
Notas por Dia	22.265.411,63	4.359.493,52
Notas por Semana	155.857.881,38	30.516.454,63
Notas por Mês	667.962.348,78	130.784.805,58

Já a Tabela 5.2 apresenta uma comparação entre as quantidades médias de registros processados nos intervalos apresentados para o LoaderNFE e para o Airbyte. As médias em questão foram calculadas considerando-se a soma do total de registros processados das notas nais quais não foram identificados erros em cada um dos lotes dividido pela soma dos tempos de processamento dos lotes (em segundos).

Tabela 5.2: Comparação das quantidades médias de registros processados nos intervalos apresentados entre o LoaderNFE e o Airbyte

	LoaderNFE	Airbyte
	Resultado	Resultado
Registros por Segundo	2.270,56	444,51
Registros por Minuto	136.233,59	26.670,37
Registros por Hora	8.174,015,53	1.600.222,33
Registros por Dia	196.176.372,79	38.405.335,84
Registros por Semana	1.373.234.609,53	268.837.350,87
Registros por Mês	5.885.291.183,70	1.152.160.075,16

É possível observar que, diariamente, o LoaderNFE consegue processar cerca de 5 vezes mais notas e registros do que o Airbyte. Apesar disso, diante do atual cenário na secretaria da Paraíba (SEFAZ-PB) onde são recebidas num intervalo de 24 horas pouco mais de 1 milhão de notas fiscais de consumidor eletrônica (NFC-e), ambas as soluções seriam capazes de processar com folga o volume em questão.

6. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho levantou o estado da técnica na área de extração, carga e transformação (ELT) de dados e apresentou uma alternativa de código-fonte aberto que pode ser utilizada para integração de dados de documentos fiscais eletrônicos que segue essa abordagem utilizando a ferramenta Airbyte. Também foi analisada e documentada a solução LoaderNFE, que é utilizada na SEFAZ-PB no que diz respeito à extração, transformação e carga (ETL) de dados de documentos fiscais eletrônicos no contexto do projeto "Plataforma Integrada de Dados". Além disso, foram desenvolvidos módulos no Airbyte para permitir a realização de testes comparativos de desempenho com o LoaderNFE no cenário de processamento de NFC-es num ambiente de homologação da SEFAZ-PB.

Os resultados coletados indicaram que a solução atual utilizada na SEFAZ-PB apresenta um desempenho em termos de tempo de processamento cerca de 5 vezes superior à solução proposta utilizando o Airbyte. Apesar disso, diante do atual cenário na SEFAZ-PB, onde são recebidas num intervalo de 24 horas pouco mais de 1 milhão de notas fiscais de consumidor eletrônica (NFC-e), a solução proposta mostra-se como uma boa alternativa na busca de automatizar um processo de integração de dados para o documento fiscal em questão disponibilizando dados atualizados com uma frequência adequada.

Como trabalhos futuros, é sugerido atualizar a solução proposta para utilizar a versão mais recente do Airbyte e com isso evitar problemas desconhecidos com possíveis *bugs* que já tenham sido solucionados. Também é preciso analisar o porquê a solução proposta identificou mais erros nas notas do que o LoaderNFE e corrigir esse comportamento caso ele esteja incorreto. Além disso, é interessante aumentar o volume de dados e realizar novos testes para verificar cenários de possíveis melhorias de implementação nas etapas de Extração e/ou Transformação.

Também seria interessante realizar uma análise comparativa entre soluções ELT de código aberto visando descrever vantagens e desvantagens em termos de parâmetros como implementação, manutenção, escalabilidade etc. Com isso, seria

possível disponibilizar uma base de conhecimento maior para ajudar na escolha de qual ferramenta de integração de dados seria mais interessante para uma determinada necessidade ou cenário.

REFERÊNCIAS

[1] Higbee, Trevor. “dbt and the Modern Data Warehouse”. YouTube, 2020. Disponível em:<https://youtube.com/playlist?list=PLkz_ZOuHaaRkDcANO2G57B82uritwZJEV>. Acesso em: 1 mar. 2022.

[2] Handy, Tristan. “What, exactly, is dbt?”. dbt blog, 2017. Disponível em:<https://blog.getdbt.com/what-exactly-is-dbt/>. Acesso em: 17 mar. 2022.

[3] Sommerville, Ian. Software Engineering, 10th edition. London: Pearson Education, 2016.

[4] Pratt, Mary K.; Bernstein, Corinne. “data transformation”.TechTarget, 2022. Disponível em:<https://www.techtarget.com/searchdatamanagement/definition/data-transformation>. Acesso em: 15 mar. 2022.

[5] Han, Jiawei; Kamber, Micheline; Pei, Jian. Data mining : concepts and techniques, 3rd edition. Waltham: Elsevier, 2012.

[6] Blasi, Isabel. “ETL X ELT : qual a diferença?”. Indicium blog, 2020. Disponível em:<https://blog.indicium.tech/etl-vs-elt-diferencas/>. Acesso em: 17 mar. 2022.

[7] “Data build tool”. Wikipedia. Disponível em:<https://en.wikipedia.org/wiki/Data_build_tool>. Acesso em: 17 mar. 2022.

[8] Sulonen, Mikko. “Positioning data build tool, dbt, in the data tooling landscape”. Solita blog, 2021. Disponível em:<https://data.solita.fi/positioning-data-build-tool-dbt-in-the-data-tooling-landscape/>. Acesso em: 17 mar. 2022.

[9] “Introduction to XML”. W3Schools. Disponível em:<https://www.w3schools.com/xml/xml_whatis.asp>. Acesso em: 15 mai. 2022.

[10] “XPath Tutorial”. W3Schools. Disponível em:<https://www.w3schools.com/xml/xpath_intro.asp>. Acesso em: 15 mai. 2022.

[11] Lafleur, John. “Why the Future of ETL Is Not ELT, But EL(T)”. Airbyte blog, 2020. Disponível em:<https://airbyte.com/blog/why-the-future-of-etl-is-not-elt-but-el>. Acesso em: 7 jun. 2022.

[12] “Transformations and Normalization”. Airbyte. Disponível em:<https://docs.airbyte.com/operator-guides/transformation-and-normalization/transformations-with-sql/>. Acesso em: 3 mai. 2022.

[13] “Description of the database normalization basics”. Microsoft. Disponível em:<https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description>. Acesso em: 7 mai. 2022.

[14] “Database normalization”. Wikipedia. Disponível em:<https://en.wikipedia.org/wiki/Database_normalization>. Acesso em: 7 mai. 2022.

[15] “Understand Airbyte - Basic Normalization”. Airbyte. Disponível em:<https://docs.airbyte.com/understanding-airbyte/basic-normalization/>. Acesso em: 28 abr. 2022.

[16] Martins, Anderson; Ribeiro, Anailson; Lima, Diogo; Dias, Éder; Marcos, Tharcisio. “Sistemas de Apoio à Decisão - SAD (Sistema de Suporte à Decisão-SSD)”. Disponível em:<https://docplayer.com.br/5761213-Sistemas-de-apoio-a-decisao-sad-sistema-de-suporte-a-decisao-ssd.html>. Acesso em: 9 jun. 2022.

[17] Laney, Douglas. “3D Data Management: Controlling Data Volume, Velocity and Variety”. META Group Research Note, 2001.

[18] Lyman, Peter; Hal R. Varian. “How Much Information”, 2003. Disponível em:<http://www.sims.berkeley.edu/how-much-info-2003>. Acesso em: 10 jun. 2022.

[19] Hilbert, Martin; López, Priscila. “The World’s Technological Capacity to Store, Communicate, and Compute Information”. Science, volume 332, 2011.

[20] Power, Daniel. “What is ETL software and how is it related to DSS?”. Disponível em:<http://dssresources.com/faq/index.php?action=artikel&id=50>. Acesso em: 10 jun. 2022.

[21] Mallek, Hana; Ghozzi, Faiza; Gargouri, Faiez. “Towards Extract-Transform-Load Operations in a Big Data context”. International Journal of Sociotechnology and Knowledge Development, volume 12, 2020.

[22] Moly, Marzana Ifat; Roy, Ovijit; Hossain, Md. Alomgir. “An Advanced ETL Technique for Error Free Data in Data Warehousing Environment". International Journal of Scientific Research & Engineering Trends, volume 5, 2019.

[23] Bâra, Adela; Lungu Ion. “Improving Decision Support Systems with Data Mining Techniques”. Advances in Data Mining Knowledge Discovery and Applications, edited by Adem Karahoca, IntechOpen, 2012.

[24] Ortega, Pablo Michel Marín; Dmitriyev, Viktor; Abilov, Marat; Gómez, Jorge Marx. “ELTA: New Approach in Designing Business Intelligence Solutions in Era of Big Data”. Procedia Technology, volume 16, 2014.

[25] Waas, Florian; Wrembel, Robert; Freudenreich, Tobias; Thiele, Maik; Koncilia, Christian; Furtado, Pedro. “On-Demand ELT Architecture for Right-Time BI: Extending the Vision”. International Journal of Data Warehousing and Mining, volume 9, 2013.

[26] Nwokeji, Joshua Chibuike; Matovu, Richard. “A Systematic Literature Review on Big Data Extraction, Transformation and Loading (ETL)”. Intelligent Computing, Proceedings of the 2021 Computing Conference, volume 2, 2021.

[27] Fang, Huang. “Managing Data Lakes in Big Data Era: What’s a data lake and why has it became popular in data management ecosystem”. 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015.

[28] Wibowo, Ardianto. “Problems and available solutions on the stage of Extract, Transform, and Loading in near real-time data warehousing (a literature study)”. 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA), 2015.

[29] Tayade, Dipti M.. “Comparative Study of ETL and E-LT in Data Warehousing”. International Research Journal of Engineering and Technology (IRJET), volume 6, 2019.

[30] “Workers & Jobs”. Airbyte. Disponível em:<https://docs.airbyte.com/understanding-airbyte/jobs/>. Acesso em: 19 jun. 2022.

[31] “Comparison of the top ETL / ELT tools”. Airbyte. Disponível em:<https://airbyte.com/etl-tools-comparison>. Acesso em: 23 jun. 2022.

[32] Geron, Cecília; Finatelli, João; Faria, Ana; Romeiro, Maria. “SPED – Sistema Público de Escrituração Digital: Percepção dos Contribuintes em relação os impactos de sua adoção". Revista de Educação e Pesquisa em Contabilidade, volume 5, 2011.

[33] Santos, Daniel Fagner Pedroza. “Um estudo comparativo sobre uso de modelos de dados para notas fiscais eletrônicas". Dissertação de Mestrado, UFPB, Centro de Informática, Programa de Pós-Graduação em Informática, 2021.

[34] “Entenda as diferenças entre NFe, NFSe e NFCe”. ConexãoNF-e blog, 2020. Disponível em:<https://blog.conexaonfe.com.br/entenda-a-diferenca-entre-nfe-nfse-e-nfce/>. Acesso em: 24 jun. 2022.