UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Victor Rodrigues Gouvêa

Estudo comparativo de ferramentas de orquestração: Dagster e Airflow

Florianópolis
2025

Victor Rodrigues Gouvêa

Estudo comparativo de ferramentas de orquestração: Dagster e Airflow

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Renato Fileto, Dr.

Florianópolis
2025

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

Gouvêa, Victor Rodrigues
Estudo comparativo de ferramentas de orquestração: Dagster e Airflow / Victor Rodrigues Gouvêa ; orientador, Renato Fileto, 2025.
101 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2025.

Inclui referências.

1. Ciências da Computação. 2. Orquestração de dados. 3. ETL. 4. Apache Airflow. 5. Dagster. I. Fileto, Renato. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Victor Rodrigues Gouvêa

Estudo comparativo de ferramentas de orquestração: Dagster e Airflow

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Graduação em Ciência da Computação.

Florianópolis, 18 de Novembro de 2025.

Prof. Lúcia Helena Martins Pacheco, Dra.
Coordenadora do Curso

Banca Examinadora:

Prof. Renato Fileto, Dr.
Orientador
Universidade Federal de Santa Catarina

Prof. Carina Friedrich Dorneles, Dra.
Avaliadora
Universidade Federal de Santa Catarina

Prof. Ronaldo dos Santos Mello, Dr.
Avaliador
Universidade Federal de Santa Catarina

Florianópolis, 2025.

RESUMO

A crescente complexidade dos ecossistemas de dados torna as ferramentas de orquestração essenciais para a automação e governança de fluxos de trabalho. No entanto, a escolha entre soluções modernas, como Apache Airflow e Dagster, representa um desafio para profissionais da área, que precisam ponderar entre maturidade, desempenho e paradigmas de desenvolvimento. Este trabalho apresenta um estudo comparativo empírico entre Airflow e Dagster, com o objetivo de avaliar suas características, vantagens e limitações em um cenário de aplicação real. Para isso, foram desenvolvidos e executados pipelines de Extração, Transformação e Carga (ETL) em ambas as ferramentas. O estudo de caso consistiu na coleta, processamento e integração de dados públicos sobre os candidatos das eleições de 2022 (a partir de arquivos CSV) e 2024 (via web scraping de dados JSON) em Santa Catarina, que foram consolidados em um banco de dados relacional unificado. A análise comparativa abrangeu três eixos principais: experiência de desenvolvimento, interface e monitoramento, e desempenho. O Dagster destacou-se pela superior experiência de desenvolvimento, promovendo maior reutilização de código através de seu paradigma orientado a ativos e partições, e oferecendo observabilidade de dados mais rica e uma interface mais intuitiva. Em contrapartida, o Airflow foi mais eficiente, com tempos de execução significativamente menores em cargas de dados massivas, sendo mais de duas vezes mais rápido no cenário mais intensivo. Conclui-se que a escolha entre as ferramentas envolve um trade-off fundamental: o Airflow se sobressai em cenários que demandam máximo desempenho e escalabilidade, enquanto o Dagster é a escolha ideal para projetos onde a produtividade do desenvolvedor, a governança e a linhagem dos dados são prioritárias. Este estudo fornece, portanto, subsídios práticos e quantitativos para auxiliar na tomada de decisão técnica entre essas duas importantes plataformas.

Palavras-chave: ETL. Workflows. Orquestração de dados. Apache Airflow. Dagster.

ABSTRACT

The growing complexity of data ecosystems makes orchestration tools essential for the automation and governance of workflows. However, choosing between modern solutions such as Apache Airflow and Dagster poses a challenge for professionals in the field, who must weigh maturity, performance, and development paradigms. This work presents an empirical comparative study between Airflow and Dagster, aiming to evaluate their characteristics, advantages, and limitations in a real-world application scenario. To this end, Extract, Transform, and Load (ETL) processes were developed and executed using both tools. The case study involved the collection, processing, and integration of public data of candidates from the 2022 elections (from CSV files) and 2024 elections (via web scraping of JSON data) in Santa Catarina, consolidated into a unified relational database. The comparative analysis covered three main axes: development experience, interface and monitoring, and performance. Dagster stood out for its superior development experience, promoting greater code reuse through its asset- and partition-oriented paradigm, while offering richer data observability and a more intuitive interface. In contrast, Airflow was more efficient, with significantly shorter execution times for large-scale data loads—over twice as fast in the most intensive scenario. It is concluded that the choice between the two tools involves a fundamental trade-off: Airflow excels in scenarios that demand maximum performance and scalability, while Dagster is the ideal choice for projects where developer productivity, data governance, and lineage are priorities. This study therefore provides practical and quantitative insights to support technical decision-making between these two important platforms.

Keywords: ETL. Workflows. Data Orchestration. Airflow. Dagster,

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CDC	Captura de Dados Alterados
CLI	*Command-Line Interface*
CNPJ	Cadastro Nacional da Pessoa Jurídica
CPF	Cadastro de Pessoas Físicas
CSV	*Comma-Separated Values*
DAG	*Directed Acyclic Graph* (Grafo Acíclico Direcionado)
DSA	*Data Staging Area* (Área de Staging de Dados)
ETL	Extração, Transformação e Carregamento
IBGE	Instituto Brasileiro de Geografia e Estatística
JSON	*JavaScript Object Notation*
OLTP	*Online Transaction Processing*
PDF	*Portable Document Format*
SC	Santa Catarina
SCOAP	Sponsoring Consortium for Open Access Publishing in Particle Physics
TSE	Tribunal Superior Eleitoral

SUMÁRIO

1 INTRODUÇÃO

O crescente volume de dados e a necessidade de transformá-los em informações úteis tornam os processos de extração, transformação e carga (ETL, do inglês Extraction, Transformation and Load) essenciais, em arquiteturas como as de data warehouses, data lakes e mediadores. Com a intensificação das demandas por atualizações em tempo real e operações de integração de dados de diferentes fontes, o papel das ferramentas de orquestração se torna fundamental para garantir que esses processos sejam realizados de forma eficiente, padronizada e resiliente. Historicamente, os processos de ETL existiram de forma implícita, como tarefas rotineiras de programação, até que, no início do século XXI, começaram a ganhar importância como componentes críticos para o sucesso de projetos de data warehousing (VASSILIADIS; SIMITSIS, 2009).

Nesse contexto, ferramentas modernas de orquestração de dados trouxeram grandes benefícios para a organização de processos complexos, aumentando a eficiência, confiabilidade e padronização, o que resulta em uma maior qualidade e consistência dos dados coletados (PETRAITYTĖ, 2024). Além disso, essas ferramentas trazem uma melhor visibilidade do workflow através de interfaces gráficas próprias, permitindo que desenvolvedores e gerentes monitorem com mais eficiência os fluxos de dados.

1.1 MOTIVAÇÃO

Dentro dessa categoria de ferramentas, podemos destacar Airflow e Dagster, que são duas ferramentas muito populares no contexto da orquestração de dados. Ambas têm muitas semelhanças entre si, como a viabilização de fluxos de dados especificados por código, conexões embutidas com as tecnologias mais recentes voltadas para dados, uma boa experiência de desenvolvimento local, código aberto e agendamento de tarefas (PETRAITYTĖ, 2024). Porém, cada uma tem suas funcionalidades e características únicas, as quais fazem com que se destaquem de formas diferentes.

Mesmo tendo conhecimento de certas vantagens e desvantagens de cada tecnologia, os profissionais de área de dados e desenvolvimento ainda têm grandes dificuldades em escolher com confiança qual ferramenta de orquestração utilizar quando surgem demandas que implicam na implementação de pipelines de dados. Essa lacuna na literatura prática motiva a necessidade de uma análise comparativa direta entre as duas tecnologias.

1.2 PROBLEMA DE PESQUISA

Diante da dificuldade de escolha enfrentada por profissionais, este trabalho busca resolver o seguinte problema: a ausência de uma comparação empírica e direcionada que detalhe, em um cenário prático, as vantagens e desvantagens do Airflow e do Dagster para a implementação de pipelines de dados. Portanto, a pergunta central que este estudo busca responder é: Quais são as principais diferenças em termos de usabilidade, desempenho e funcionalidades entre Airflow e Dagster quando aplicados a um caso de uso real de orquestração de fluxos de ETL?

1.3 OBJETIVOS

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é comparar de forma prática e empírica as ferramentas de orquestração de dados Airflow e Dagster, com a finalidade de detalhar as características gerais de ambas as ferramentas e especificidades de cada uma. Para essa finalidade, pretende-se implementar fluxos de dados com ambas as ferramentas, utilizando casos de demandas reais que requerem o uso de ferramentas de orquestração de dados.

1.3.2 Objetivos Específicos

1. Efetuar um estudo comparativo teórico e pratico das ferramentas de orquestração de processos Airflow e Dagster;
2. Definir e implementar fluxos de ETL em ambas as ferramentas, os quais servirão de base para a comparação entre elas.
3. Conduzir uma análise comparativa qualitativa e quantitativa, avaliando as ferramentas com base na experiência de desenvolvimento, modularidade, usabilidade da interface, capacidades de monitoramento e nos resultados de desempenho obtidos.
4. Realizar uma análise exploratória dos dados consolidados, validando a eficácia dos pipelines desenvolvidos e apresentando insights quantitativos sobre o cenário eleitoral;

1.4 JUSTIFICATIVA E RELEVÂNCIA

Este trabalho é importante pois oferece um guia prático para profissionais da indústria de dados, auxiliando na tomada de decisão sobre qual tecnologia de orquestração adotar. Para a academia, a relevância está na produção de uma análise empírica que contribui para a literatura sobre engenharia de dados, comparando duas

ferramentas modernas e populares em um cenário controlado. O impacto potencial do estudo é a redução do tempo e do risco associados à escolha e implementação de pipelines de dados, promovendo um avanço tecnológico ao facilitar a adoção de boas práticas de orquestração.

1.5 METODOLOGIA

Inicialmente, serão explorados os conceitos fundamentais relacionados a ETL, fluxos de trabalho (workflows) e ferramentas de orquestração de processos de ETL de dados. Esta etapa envolverá uma pesquisa abrangente para compreender os fundamentos teóricos e práticos que embasam o uso dessas tecnologias. Em seguida, o foco será direcionado ao estudo das ferramentas Dagster e Airflow, analisando suas arquiteturas, documentação e boas práticas de uso, com o objetivo de consolidar o conhecimento técnico sobre essas soluções.

Na sequência, serão desenvolvidos fluxos de dados utilizando as tecnologias mencionadas. Como exemplo prático, serão implementados dois pipelines de dados que coletarão informações sobre os candidatos do estado de Santa Catarina nas eleições federais de 2022 e municipais de 2024. Essa aplicação prática permitirá validar os conceitos aprendidos e testar as funcionalidades das ferramentas em cenários reais.

Após a implementação dos processos de ETL de dados, será realizada uma análise detalhada dos resultados obtidos com cada tecnologia. Essa análise comparará aspectos como usabilidade, desempenho e funcionalidades exclusivas de cada ferramenta. Adicionalmente, será apresentada uma análise exploratória dos dados processados pelos pipelines desenvolvidos, visando caracterizar o conjunto de dados resultante.

Ao final deste trabalho de conclusão de curso, espera-se apresentar uma comparação empírica detalhada e fundamentada entre as ferramentas Airflow e Dagster, além de demonstrar fluxos de dados implementados que comprovem as conclusões alcançadas sobre as características e aplicações de cada tecnologia.

1.6 ESTRUTURA DO TRABALHO

O restante deste trabalho está organizado da seguinte maneira. O Capítulo 2 apresenta os fundamentos e tecnologias utilizados no desenvolvimento do trabalho e necessários para o seu entendimento. O Capítulo 3 discute o scraping dos dados de uma das fontes utilizadas no trabalho e o esquema do banco de dados modelado para as aplicações desenvolvidas. O Capítulo 4 descreve o processo de desenvolvimento dos fluxos de dados propostos. O Capítulo 5 relata a comparação prática das ferramentas. O Capítulo 6 apresenta uma análise exploratória dos dados coletados pelos fluxos

desenvolvidos. E, finalmente, o Capítulo 7 finaliza o trabalho através da apresentação de conclusões e temas para trabalhos futuros.

2 FUNDAMENTOS

Este capítulo define os principais conceitos usados no trabalho e necessários ao seu entendimento, assim como descreve as técnicas e ferramentas usadas na análise proposta. A Seção 2.1 esclarece os principais conceitos à workflows e orquestração de dados. A Seção 2.2 aborda as ferramentas de orquestração de dados a serem comparadas no trabalho proposto. A Seção 2.3 revisa comparações prévias feitas entre as ferramentas Apache Airflow e Dagster na literatura, destacando as vantagens e limitações teóricas identificadas em estudos anteriores.

2.1 WORKFLOWS E PROCESSOS DE ETL

De maneira geral, workflows são automações de procedimentos nos quais documentos, informações ou tarefas transitam entre diferentes participantes, sejam eles humanos ou sistemas, seguindo um conjunto definido de regras (HOLLINGSWORTH, 1995). O objetivo final é alcançar ou contribuir para um resultado de negócio específico. Essa automação pode envolver tanto atividades manuais organizadas de forma sistemática quanto operações totalmente informatizadas, sendo esta última o foco das ferramentas modernas de orquestração de dados.

No contexto de sistemas computacionais, um workflow pode ser definido como a facilitação ou automação informatizada de um processo de negócio. Essa automação permite que processos complexos sejam modelados, monitorados e executados de forma estruturada, com flexibilidade para futuras alterações. Para padronizar a compreensão desses sistemas, a *Workflow Management Coalition (WfMC)* propôs o *Workflow Reference Model (WfRM)*, uma arquitetura conceitual que continua sendo a base para as ferramentas modernas.

Estrutura de um Sistema de Workflow segundo o WfRM

O WfRM divide um sistema de workflow em duas fases principais: a de definição (*Build-time*) e a de execução (*Run-time*), conforme ilustrado na Figura 1. A partir deste modelo, podemos extrair os componentes e conceitos fundamentais que definem a arquitetura de qualquer orquestrador de dados.

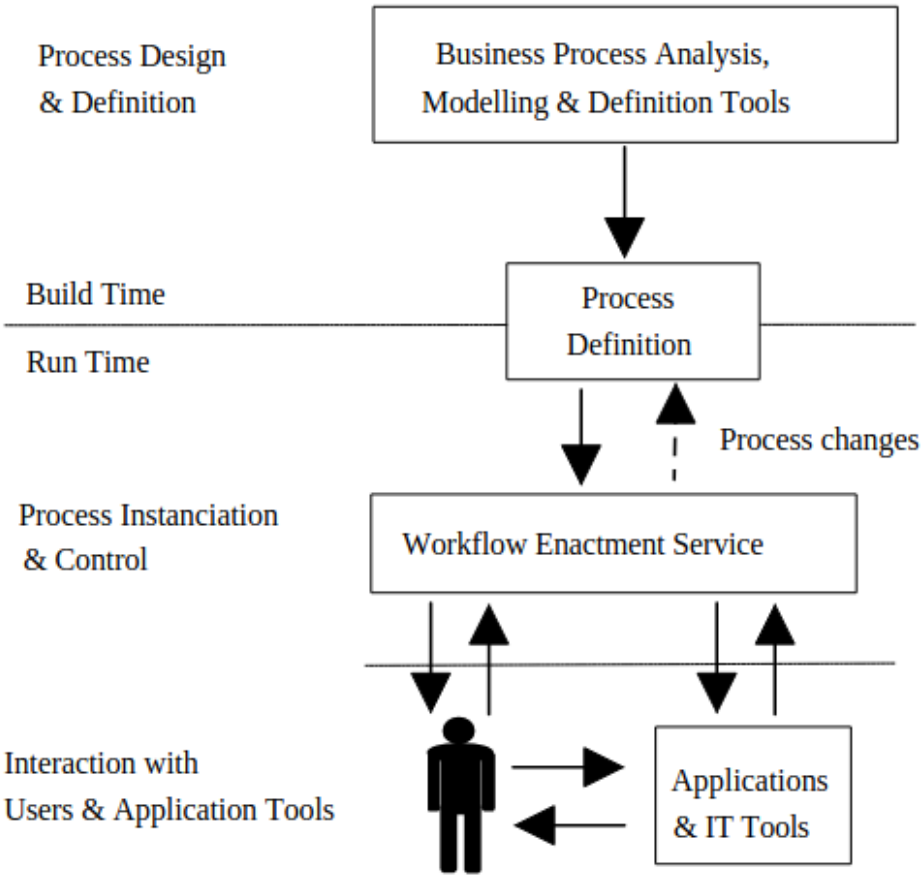


Figura 1 – Características de um Sistema de Workflow, destacando a separação entre Build-time e Run-time

Fonte: Retirado de (HOLLINGSWORTH, 1995).

Build-time e a *Process Definition*: A fase de *Build-time* é o momento em que o fluxo de atividades é desenhado e modelado. O resultado desta fase é um artefato crucial conhecido como *Process Definition*: a representação computacional e formal de um processo. Esta definição contém a lógica do workflow, incluindo suas etapas individuais, que o WfRM denomina *Process Activities*, e as regras de transição entre elas. Esta é a etapa onde o desenvolvedor escreve o código que define o pipeline.

Run-time e o *Workflow Enactment Service*: A fase de *Run-time* é onde a *Process Definition* é interpretada e executada. O componente central desta fase é o *Workflow Enactment Service*, o motor do sistema, que é responsável por criar, gerenciar e executar as instâncias do workflow. Suas funções podem ser divididas em duas áreas principais:

- **Run-time Process Control**: Refere-se ao controle interno do processo. É a função que instancia e monitora as execuções, agenda a sequência de *Process Activities* e gerencia seus estados (e.g., pendente, em

execução, falhou). É o que os orquestradores modernos chamam de *scheduler* e *executor*.

- **Run-time Activity Interactions**: Refere-se à interação do motor com o mundo externo. Quando uma *Process Activity* precisa ser executada, o motor utiliza uma *Invoked Application Interface* para acionar um executor e gerenciar a troca de informações e dados com ele.

A esses componentes clássicos, as implementações modernas somam a capacidade de distribuição (espalhar a execução por múltiplos nós) e uma ênfase crescente em observabilidade, que abrange métricas, logs, alertas e rastreabilidade de execuções, que são requisitos críticos para pipelines em ambientes de produção.

Processos de Extração, Transformação e Carga (ETL)

Complementarmente ao conceito de workflows, os processos de Extração, Transformação e Carga (ETL) representam uma categoria específica e fundamental de workflows, especialmente no contexto de data warehousing e integração de dados. Eles são responsáveis por todas as operações que ocorrem nos bastidores de uma arquitetura de data warehouse, desde a coleta de dados em sistemas de origem até sua disponibilização em um formato adequado para análise (VASSILIADIS; SIMITSIS, 2009).

Um processo ETL, em sua descrição de alto nível, pode ser decomposto nas seguintes fases principais:

1. **Extração (Extraction):** Nesta fase inicial, os dados são extraídos de diversas fontes de dados. Essas fontes podem ser extremamente heterogêneas, incluindo bancos de dados relacionais de sistemas transacionais (OLTP), sistemas legados, arquivos em variados formatos (como planilhas e documentos de texto), páginas web, e até mesmo fluxos de dados contínuos (streaming). Idealmente, para otimizar o processo, apenas os dados que foram alterados desde a última execução do ETL (novas inserções, atualizações e exclusões) devem ser extraídos, uma técnica conhecida como Captura de Dados Alterados (CDC). A extração deve ser realizada com o mínimo de sobrecarga e interferência nos sistemas de origem, que frequentemente possuem suas próprias atividades administrativas concorrentes.
2. **Transformação (Transformation):** Após a extração, os dados são geralmente movidos para uma área intermediária especializada, conhecida como Área de Staging de Dados (DSA). É nesta área que ocorrem as principais operações de transformação, homogeneização, limpeza dos dados, integração, etc. A modelagem conceitual dessas transformações é crucial, especialmente nos estágios iniciais de um projeto de data warehouse, para rastrear

as inter-relações entre atributos e definir as atividades de ETL necessárias (VASSILIADIS; SIMITSIS; SKIADOPOULOS, 2002). Essa modelagem pode envolver a definição de entidades, atributos e restrições de ETL.

3. **Carga (Loading):** Finalmente, os dados transformados e limpos são carregados no data warehouse central e em seus componentes associados (VASSILIADIS; SIMITSIS, 2009). A carga deve ser eficiente, especialmente considerando os grandes volumes de dados. Técnicas de carga em lote são preferíveis a inserções linha a linha. Considerações de desempenho durante a carga incluem a manutenção de índices e visões materializadas.

Tradicionalmente, os processos ETL são executados periodicamente para atualizar o data warehouse durante períodos de baixa carga do sistema, dentro de uma janela de tempo específica. No entanto, as necessidades de negócios modernas impulsionam a demanda por atualização de data warehouses em tempo quase real, levando a avanços tecnológicos nessa área.

Em suma, a compreensão dos workflows como mecanismos de automação e orquestração de processos de negócio é fundamental, e sua aplicação se torna particularmente tangível e crítica nos processos de Extração, Transformação e Carga (ETL). Enquanto os workflows fornecem a estrutura e o controle para a execução de tarefas sequenciais ou paralelas, os processos ETL representam uma implementação especializada desses fluxos, focada na coleta, tratamento e disponibilização de dados para fins analíticos. A sinergia entre a robustez da modelagem de workflows e a especificidade das operações de ETL é, portanto, essencial para garantir a qualidade, consistência e confiabilidade dos dados que alimentam novos projetos e resultam em tomadas de decisão nas organizações.

2.2 FRAMEWORKS PARA ORQUESTRAÇÃO DE DADOS

Frameworks de orquestração de dados desempenham um papel essencial na engenharia de dados, ao fornecerem estruturas organizadas para coordenar, automatizar e monitorar processos complexos de manipulação de dados (PETRAITYTĖ, 2024). Esses frameworks permitem a execução e acompanhamento de pipelines de forma padronizada, eficiente e transparente, reduzindo intervenções manuais e minimizando erros. Além disso, promovem grande flexibilidade, facilitando a adaptação a novos requisitos de negócio e tecnologias emergentes. Assim, tornam-se ferramentas indispensáveis para garantir que os fluxos de dados estejam alinhados às boas práticas e aos objetivos organizacionais.

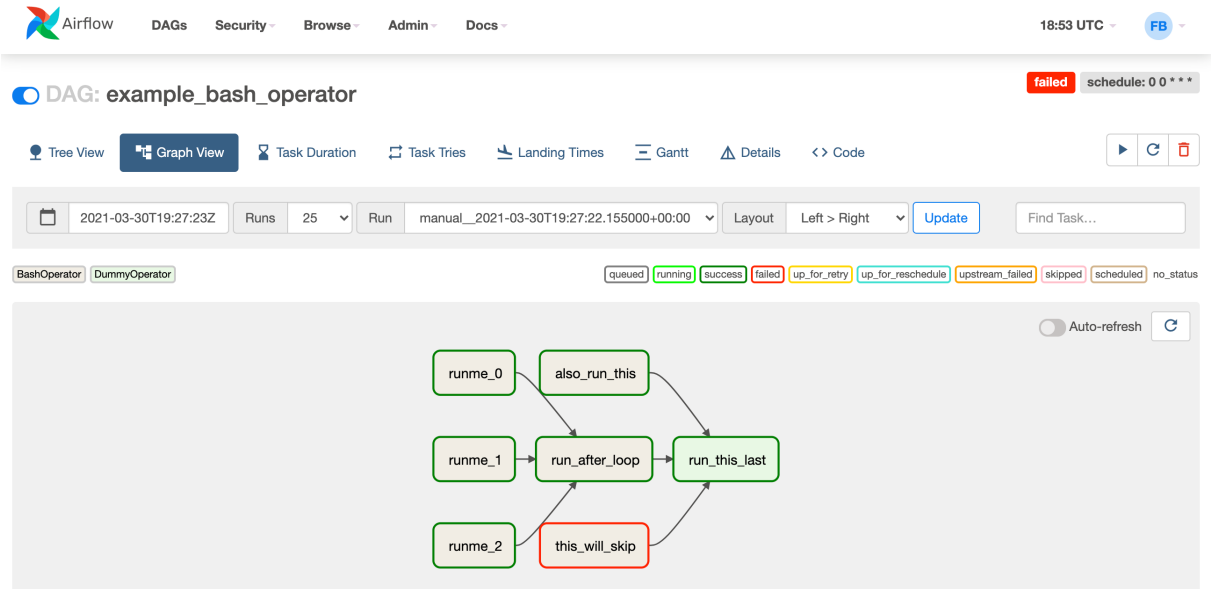
Tanto Apache Airflow quanto Dagster são ferramentas desenvolvidas para a criação de workflows e orquestração de dados, e são muito relevantes no contexto atual desse tipo de tecnologia.

2.2.1 Apache Airflow

O Apache Airflow é uma das ferramentas mais consolidadas e amplamente utilizadas no gerenciamento de fluxos de dados. Como framework de código aberto, desenvolvido em Python, permite que profissionais de dados definam seus pipelines como código, promovendo maior controle, reprodutibilidade e versionamento. Sua robustez e flexibilidade fazem do Airflow uma referência clássica quando se trata de orquestração de dados e automação de workflows. De acordo com a pesquisa realizada pela própria comunidade do Airflow em 2022 (AIRFLOW, 2022), a ferramenta é baixada milhões de vezes por mês e é utilizada por milhares de empresas de diferentes portes em seus processos de dados. O projeto teve início na Airbnb em 2014, criado por Maxime Beauchemin, e ingressou no programa de incubação da Apache Software Foundation em março de 2016, sendo promovido a projeto de nível superior em 2019 (ROACH, 2022).

Quando pensamos nas principais funcionalidades da ferramenta, devemos descrever sua unidade mais simples: as tarefas (tasks). Elas podem ser entendidas como operações dentro de um pipeline de dados. Quando essas tarefas são organizadas com base em suas dependências, são formadas as estruturas conhecidas como DAG's. Essas estruturas explicitam a ordem a qual as tarefas devem ser executadas, permitindo uma visão clara do fluxo de trabalho e garantindo consistência na execução do fluxo. A Figura 2 mostra um exemplo de visualização de um grafo referente a um fluxo de trabalho no Airflow, onde é possível entender as dependências das DAG's e os seus status referentes a uma execução específica.

Figura 2 – Exemplo de grafo de visualização no Airflow



Fonte: Retirado de (AIRFLOW, 2025).

Para dar suporte à execução dessas tarefas, o Airflow é composto por quatro

componentes principais:

- Scheduler: responsável por verificar periodicamente quais DAG's devem ser executados com base em uma data de início e um intervalo de agendamento definidos no código.
- Executor: gerencia a execução das tarefas, alocando os recursos necessários para que sejam executadas, seja localmente, em múltiplos processos ou até mesmo em ambientes distribuídos.
- Metadata Database: banco de dados que armazena todas as informações sobre a execução dos DAG's e tarefas, além de variáveis, conexões e outras configurações definidas pelo usuário.
- Webserver (Interface Web): fornece uma interface gráfica intuitiva, onde os usuários podem visualizar o estado dos DAG's, monitorar execuções, reexecutar tarefas manualmente, configurar variáveis e acompanhar o desempenho geral dos pipelines.

A construção dos fluxos de trabalho pode ser feita de diferentes maneiras, sendo a mais recente a *TaskFlow API*, uma abstração que permite definir tarefas como funções Python decoradas com `@task`. Essa abordagem simplifica o desenvolvimento, tornando o código mais legível e intuitivo ao tratar as tarefas como funções e os dados passados entre elas como seus resultados. A definição do fluxo de trabalho se assemelha a um script Python padrão, como no exemplo conceitual abaixo (Listing 2.2.1), que resume um fluxo comum de ETL:

```python
import pandas as pd
from typing import Dict, List
from airflow.decorators import dag, task

dag(dag_id='etl_candidatos_conceitual', ...)
def etl_pipeline_conceitual():
    """
    DAG conceitual que demonstra o fluxo de ETL com a TaskFlow API.
    """

    @task
    def extract_csv_data() -> Dict[str, pd.DataFrame]:
        """
        Função de extração resumida para exemplo
        """

        return {...}

    @task
    def transform_data(raw_data: Dict[str, pd.DataFrame]) -> Dict[str, List[Dict]]:
        """
        Função de transformação resumida para exemplo
        """

        return {...}
    ...

    # --- Definição do fluxo de trabalho da DAG ---

    # 1. Extrai os dados
    extracted_data = extract_csv_data()

    # 2. Transforma os dados (dependência em relação a extração)
    transformed_data = transform_data(extracted_data)
    ...
# Instancia a DAG
etl_pipeline_conceitual()
```

Fonte: O autor.

Listing 2.2.1 – Exemplo de função em Python para gerar o resumo das despesas de um candidato

Como o exemplo demonstra, a dependência entre as tarefas `extract_csv_data` e `transform_data` é criada implicitamente, simplesmente passando o resultado da primeira como argumento para a segunda. Essa sintaxe torna o fluxo de dados explícito e o código da DAG significativamente mais limpo e fácil de manter.

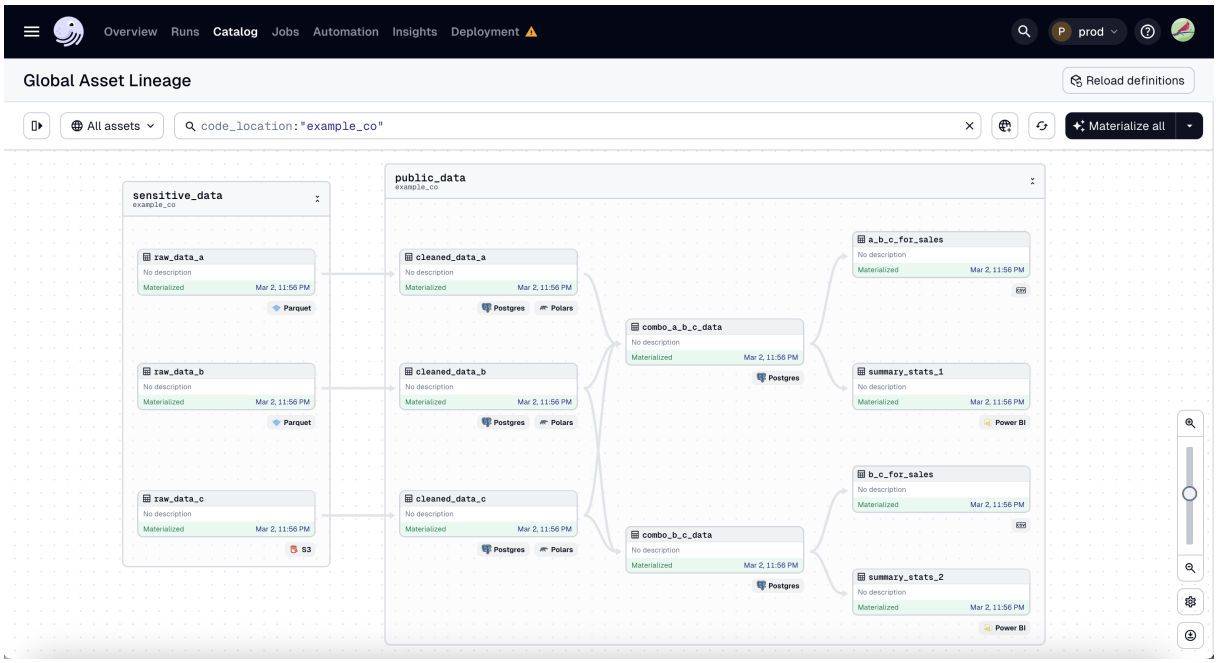
Em resumo, o Apache Airflow oferece um conjunto completo de funcionalidades para orquestrar e monitorar workflows complexos de maneira eficiente, confiável e flexível, sendo uma das ferramentas mais populares quando se trata de orquestração de dados.

2.2.2 Dagster

O Dagster adota uma abordagem diferenciada e centrada em assets (asset-based) para a construção de pipelines de dados. No contexto da ferramenta, qualquer objeto de dados armazenado de forma persistente, como um arquivo ou uma tabela,

é considerado um asset. Esses ativos são definidos diretamente no código, por meio de funções escritas em Python, utilizando decorators que permitem definir diversas características dos assets, como declarar explicitamente seus insumos, regras de materialização e comportamentos desejados. Ao serem executadas, essas funções fazem com que o Dagster crie automaticamente as dependências entre etapas e realiza a materialização dos ativos, ou seja, sua geração ou atualização no armazenamento. Essa abordagem orientada por assets facilita o rastreamento de como os dados são produzidos, transformados e consumidos dentro de um pipeline, proporcionando maior clareza e controle sobre o ciclo de vida dos dados (BENDRIKOV, 2024). A Figura 3 ilustra como a ferramenta representa visualmente essa linhagem de dados.

Figura 3 – Exemplo do grafo de assets na interface do Dagster



Fonte: Retirado de (DAGSTER. . . , 2025).

Além disso, o Dagster oferece um framework coeso para definir, agendar e monitorar fluxos de trabalho, enfrentando de forma unificada os desafios que surgem em abordagens tradicionais, como orquestração manual ou scripts isolados. Sua arquitetura modular favorece a reutilização de componentes, testabilidade e observabilidade das execuções. Os principais conceitos da ferramenta incluem (DAGSTER. . . , 2025):

- Assets: artefatos de dados gerados durante a execução, fundamentais para rastreamento de linhagem e controle de versões.
- Jobs: agrupamentos de ops com suas dependências, que representam um fluxo de trabalho completo.
- Schedules: agendamentos recorrentes que permitem a execução automática de workflows em intervalos definidos.

- Sensors: gatilhos reativos que executam workflows com base em mudanças externas, como a chegada de um novo arquivo ou a conclusão de um processo.

Em suma, o Dagster se destaca no ecossistema de orquestração de dados principalmente por sua abordagem inovadora centrada e baseada em assets, e seu forte suporte à modularidade e sua ênfase em boas práticas de engenharia, como tipagem, testabilidade e rastreamento de linhagem dos dados também se mostram como características muito relevantes na recente ascensão da ferramenta.

2.3 COMPARAÇÕES ENTRE AIRFLOW E DAGSTER NA LITERATURA

Outros trabalhos já abordam a utilização de frameworks de orquestração, destacando Apache Airflow e Dagster devido às suas capacidades de lidar com pipelines de dados complexos. O estudo (PETRAITYTĖ, 2024) explora a aplicação de Airflow para melhorar a gestão de grandes volumes de dados no projeto SCOAP, um repositório de acesso aberto para publicações em física de partículas. Esse trabalho realiza uma comparação direta entre Airflow e Dagster, destacando que, embora o Airflow ofereça robustez, escalabilidade e uma grande variedade de plugins, ele apresenta desafios na sua complexidade de configuração e manutenção. Em contrapartida, o estudo identifica o Dagster como mais intuitivo, especialmente em relação à curva de aprendizado e ao desenvolvimento de pipelines baseados em assets. Ainda assim, (PETRAITYTĖ, 2024) menciona que o Dagster pode não oferecer a mesma capacidade de escala e a ampla base de suporte comunitário do Airflow.

No estudo publicado pela (DATACAMP, 2024), um comparativo direto entre Airflow e Dagster é apresentado, evidenciando características técnicas e práticas das duas ferramentas. O Airflow é destacado pela sua comunidade ativa e pela sua grande base de plugins e integrações, tornando-o uma ferramenta ideal para workflows complexos e altamente customizados. Por outro lado, o Dagster é elogiado por sua abordagem intuitiva baseada em ativos, o que simplifica significativamente o desenvolvimento e teste de pipelines, proporcionando verificações de tipos embutidas que garantem maior consistência e integridade dos dados processados. No entanto, o estudo menciona como limitação do Dagster a sua menor maturidade em relação ao Airflow, resultando em menos suporte da comunidade para problemas específicos e soluções mais avançadas.

Como observado nos estudos apresentados, Apache Airflow e Dagster possuem diversas similaridades fundamentais na maneira como são utilizados e no papel central que desempenham em ecossistemas modernos de dados. Ambos permitem a criação de pipelines definidos por código, oferecem integrações nativas com tecnologias de ponta e facilitam o desenvolvimento local e a execução de testes. Entretanto,

cada ferramenta apresenta particularidades que as tornam adequadas para diferentes contextos e preferências de desenvolvimento.

A Tabela 1, adaptada de (DATACAMP, 2024), resume de forma objetiva os principais critérios comparativos entre Airflow e Dagster. Na tabela são abordados aspectos como conceito de pipeline, representação de tarefas, flexibilidade no agendamento, experiência de desenvolvimento local, robustez do sistema de tipagem, extensibilidade, tamanho e maturidade da comunidade, curva de aprendizado, facilidade para testes e maturidade na implantação em produção. Esses critérios fornecem uma visão clara e concisa das principais diferenças e vantagens teóricas de cada ferramenta, destacadas também pelos estudos citados anteriormente.

Tabela 1 – Comparação entre Airflow e Dagster

Critério	**Airflow**	**Dagster**
Conceito de pipeline	Usa *Directed Acyclic Graphs* (DAG's) para representar *pipelines*, com *tasks* como unidades básicas.	Utiliza uma abordagem baseada em *assets*, em que cada objeto de dados é chamado de *asset*, com operações chamadas de *ops*.
Representação de tarefas	As *tasks* são definidas usando *operators* ou a *TaskFlow API* para definir *tasks* baseadas em funções.	*Ops* são usadas para representar etapas no *pipeline*, com entradas e saídas tipadas para maior clareza.
Scheduling	Altamente flexível, com CRON, *timetables* e *data-aware triggers*.	Menos ênfase em agendamento, porém atrelado a *assets* e suas dependências.
Experiência de desenvolvimento local	Suporta desenvolvimento local via *Airflow CLI* para iteração rápida e testes.	Também suporta desenvolvimento e testes locais, mas com menos recursos disponíveis na comunidade.
Sistema de tipagem	Suporta *Python-type hints*, mas não é uma funcionalidade principal.	Sistema de *typing* robusto para validar entradas e saídas em cada etapa, sendo parte central do design do *pipeline*.
Extensibilidade	Altamente extensível, com mais de 1600 integrações, *custom operators*, *sensors*, etc.	Extensibilidade mais limitada, porém intuitiva para usuários de Python.
Ecossistema e Comunidade	Comunidade grande e madura, com milhares de contribuidores e ampla variedade de recursos (blogs, tutoriais, etc.).	Comunidade menor, mas em rápido crescimento, oferecendo oportunidades de influência e contribuição.
Curva de aprendizado	Curva de aprendizado mais íngreme, devido à extensa gama de recursos e configurações personalizadas.	Curva de aprendizado menor, com criação de *pipelines* mais intuitiva e abordagem baseada em *assets*.
Testabilidade	Testes são possíveis, mas não configurados como foco central, podendo exigir configurações adicionais.	Forte ênfase em testes, com uma abordagem baseada em *assets* que facilita o teste de todo o *pipeline*.
Implantação em produção	Pode ser executado em vários ambientes, desde Kubernetes on-premise até serviços totalmente gerenciados na nuvem.	Menos maduro em termos de implantação em produção, mas ainda flexível para casos de uso corporativos, possuindo um serviço próprio para deploy

Fonte: Adaptado de (DATACAMP, 2024)

Logo, a tabela comparativa reflete de forma consistente as percepções teóricas abordadas na literatura revisada, permitindo uma visualização clara das vantagens relativas de cada ferramenta em contextos distintos. Tal alinhamento entre literatura e aspectos técnicos enfatiza que a escolha entre Airflow e Dagster deve levar em consideração requisitos específicos do projeto, como complexidade desejada, curva de aprendizado da equipe e maturidade das funcionalidades necessárias. Apesar dos estudos citados já pontuarem claramente vantagens e desvantagens das ferramentas, outros aspectos relevantes podem surgir ao longo do desenvolvimento prático, permitindo conclusões adicionais e mais assertivas sobre a utilização dessas ferramentas em cenários reais.

Nesse cenário, o estudo comparativo empírico desenvolvido neste trabalho busca complementar essas iniciativas predominantemente conceituais ao aplicar ambos os orquestradores em um caso de uso completo de ETL com dados eleitorais reais, medindo sistematicamente tempo de execução, consumo de recursos e impacto das decisões de modelagem dos pipelines. Ao articular os resultados práticos com os critérios teóricos já discutidos na literatura, este trabalho oferece evidências quantitativas e qualitativas que ajudam a reduzir a lacuna entre recomendações em alto nível e a realidade de projetos de engenharia de dados, permitindo avaliar com maior precisão as implicações concretas da escolha entre Airflow e Dagster em ambientes de produção.

3 DESCRIÇÃO, COLETA E ORGANIZAÇÃO DOS DADOS

Este capítulo detalha o processo de coleta e organização dos dados utilizados neste estudo comparativo. O texto se inicia com a apresentação do estudo de caso, detalhando as fontes de dados de 2022 e 2024 (Seção 3.1). Em seguida, a Seção 3.2 descreve como foi desenvolvido o scraper para a extração dos dados de 2024, ressaltando a estruturação dos dados em formato JSON. Por fim, a Seção 3.3 apresenta o esquema relacional projetado para unificar os dados de ambas as fontes, descrevendo a estrutura que garante uma base consistente e eficiente para as análises subsequentes.

3.1 ESTUDO DE CASO: DADOS PÚBLICOS SOBRE POLÍTICOS

O workflow desenvolvido neste trabalho tem como objetivo a coleta automatizada de dados referentes aos candidatos do estado de Santa Catarina nas eleições de 2022 e 2024. As informações obtidas sobre os candidatos incluem dados cadastrais, prestação de contas eleitorais, gastos de campanha, doadores e demais registros financeiros disponibilizados pelos órgãos oficiais.

A integração e análise dessas informações são fundamentais para possibilitar comparações entre diferentes pleitos e identificar potenciais vínculos entre políticos eleitos e estruturas empresariais, como sociedades econômicas. Com isso, busca-se fornecer subsídios para a detecção de indícios de corrupção, práticas de favorecimento ou desvios de verbas públicas, contribuindo para uma maior transparência no cenário político.

As próximas subseções apresentam iniciativas existentes relacionadas à coleta e padronização de dados eleitorais no Brasil, além de detalhar as fontes específicas utilizadas para as eleições de 2022 e 2024, com destaque para seus principais atributos e o contexto no qual serão empregados neste trabalho.

3.1.1 Iniciativas de Coleta de dados sobre políticos

Diversos trabalhos têm se dedicado a facilitar o uso e a análise de dados eleitorais brasileiros, reconhecendo a importância das informações disponibilizadas pelo Tribunal Superior Eleitoral (TSE), apesar das dificuldades práticas decorrentes da falta de padronização e inconsistências nos dados originais.

O trabalho (MEIRELES; SILVA; COSTA, 2021) propôs o pacote *electionsBR*, desenvolvido em R, que oferece funções específicas para automatizar a coleta, limpeza e organização de dados eleitorais diretamente do TSE. Esse pacote resolve problemas comuns, como a falta de padronização nos formatos dos arquivos, inconsistências em codificações e documentação insuficiente, permitindo uma análise mais rápida e

transparente dos dados eleitorais. As funções do *electionsBR* abrangem candidaturas, apurações eleitorais, dados sobre legendas e partidos, resultados eleitorais e perfis dos eleitores, o que facilita o acesso dos pesquisadores às informações necessárias para estudos na área de ciência política.

Por sua vez, (VASCONCELOS *et al.*, 2021) apresenta o dataset *CandiDATA*, que cobre um período ainda mais amplo, de 1945 a 2020. Este dataset busca solucionar problemas semelhantes aos enfrentados pelo *electionsBR*, porém com uma abordagem diferente, incluindo padronização de formatos, codificação consistente (UTF-8), integração com códigos do IBGE e inferência automática de informações faltantes, como gênero e ocupação dos candidatos. O resultado é um dataset consolidado e padronizado em formato aberto (CSV), facilitando significativamente análises históricas e permitindo integrações mais fáceis com outras bases de dados sociais e econômicas.

3.1.2 Dados dos Candidatos das Eleições 2022

A primeira fonte de dados utilizada neste trabalho é o conjunto disponibilizado pelo Tribunal Superior Eleitoral, referente às eleições de 2022. Trata-se de uma base pública e oficial, que reúne informações detalhadas sobre todos os candidatos registrados no pleito.

Essas informações estão disponíveis atualmente na página "Candidatos 2022", no site oficial do Governo Federal[1]. Nesta página, especificamente na aba "Recursos", encontram-se diversas seções contendo arquivos relacionados a diferentes temas. Cada seção disponibiliza arquivos no formato CSV, acompanhados também por um documento PDF detalhando a estrutura e descrevendo as colunas desses arquivos CSV. Como o escopo deste trabalho concentra-se exclusivamente nos candidatos do estado de Santa Catarina, serão considerados somente os arquivos CSV correspondentes a esse estado. A seguir, são descritos cada um dos arquivos CSV utilizados, focando nos principais atributos considerando o contexto deste trabalho, sem repetir descrições de atributos presentes em mais de um arquivo CSV. O Anexo A transcreve as descrições completas dos arquivos CSV utilizados tal como providas na fonte.

Candidatos

Este arquivo reúne informações detalhadas sobre os candidatos registrados. Os atributos mais relevantes para o contexto deste trabalho são:

- **NM_CANDIDATO**: Nome completo do candidato.
- **NR_CANDIDATO**: Número do candidato exibido na urna eletrônica.
- **DT_NASCIMENTO**: Data de nascimento (DD/MM/YYYY).
- **DS_GENERO**: Sexo do candidato.

[1] https://dados.gov.br/dados/conjuntos-dados/candidatos-2022

- **NR_CPF_CANDIDATO**: Número do CPF do candidato.
- **NR_TITULO_ELEITORAL_CANDIDATO**: Número do título eleitoral do candidato.
- **SQ_CANDIDATO**: Número sequencial gerado internamente pelos sistemas eleitorais, para identificar cada candidato entre os arquivos CSV.
- **DS_CARGO**: Cargo disputado (deputado, governador, senador, etc).
- **NR_PARTIDO**: Número do partido político do candidato.
- **SG_PARTIDO**: Sigla do partido político do candidato.
- **NM_PARTIDO**: Nome completo do partido político do candidato.

Este arquivo é importante para identificar os dados individuais de cada candidato e a qual partido pertence, para depois cruzar com as outras bases dessa mesma fonte.

Bens de candidatos

Contém informações detalhadas dos bens patrimoniais declarados pelos candidatos, essenciais para o estudo do patrimônio acumulado e para detectar possíveis discrepâncias patrimoniais. Os principais atributos são:

- **NR_ORDEM_BEM_CANDIDATO**: Número de ordenação do bem patrimonial material do candidato.
- **DS_TIPO_BEM_CANDIDATO**: Tipo de bem patrimonial declarado (imóveis, veículos, investimentos, etc.).
- **DS_BEM_CANDIDATO**: Descrição detalhada do bem.
- **VR_BEM_CANDIDATO**: Valor do bem declarado, em reais.
- **DT_ULT_ATUAL_BEM_CANDIDATO**: Data da última atualização do bem no sistema do governo.

Notas Fiscais

Essencial para verificar despesas de campanha e fornecedores, esta base detalha:

- **NM_UNIDADE_ARRECADADORA**: Nome da unidade arrecadadora.
- **DT_EMISSAO**: Data da emissão da nota fiscal.
- **NR_NOTA_FISCAL**: Número da nota fiscal.
- **VR_NOTA_FISCAL**: Valor total da nota fiscal emitida.
- **NR_CHAVE_ACESSO**: Número da chave de acesso da nota fiscal.
- **NM_URL_ACESSO**: URL de acesso da nota fiscal.

- **NR_CPF_CNPJ_EMITENTE**: CPF ou CNPJ do emissor da nota fiscal.

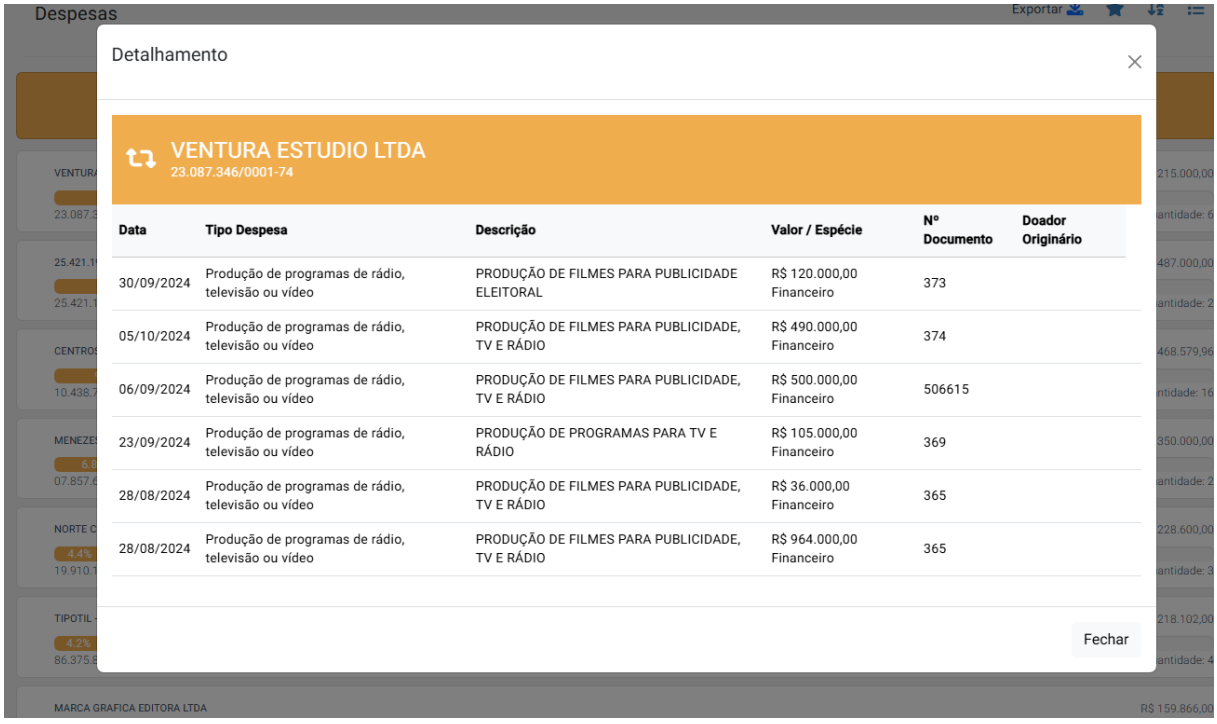
Essa fonte fornece, portanto, uma visão abrangente, permitindo cruzar informações políticas, financeiras e patrimoniais, e assim investigar possíveis irregularidades, conflitos de interesses ou indícios de corrupção entre os candidatos, sobretudo aqueles eleitos.

3.1.3 Dados dos Candidatos das Eleições de 2024

A segunda fonte utilizada neste trabalho é o sistema de Divulgação de Candidaturas e Contas Eleitorais, disponibilizado também pelo Tribunal Superior Eleitoral[2]. Essa plataforma reúne informações detalhadas sobre as candidaturas registradas e os dados financeiros das campanhas eleitorais, permitindo o acesso às prestações de contas dos candidatos que concorreram nas eleições municipais de 2024.

A fonte oferece dados essenciais para a análise da movimentação financeira dos candidatos, incluindo os valores arrecadados e pagos, bem como o limite legal de gastos para cada cargo. Também estão disponíveis informações detalhadas sobre cada despesa gasta na candidatura, assim como a identificação do fornecedor do serviço e até mesmo detalhes da nota fiscal de algumas dessas despesas, como mostra a Figura 4. Informações sobre doadores e suas doações também são fornecidos de forma semelhante em outra seção da página do candidato, onde os dados sobre cada doação e o seu doador são detalhados.

Figura 4 – Página com as despesas contratadas de um fornecedor



Fonte: Captura de tela retirada de (TRIBUNAL SUPERIOR ELEITORAL, 2024).

[2] https://divulgacandcontas.tse.jus.br/divulga/#/candidato/SUL/SC/2045202024

Notas fiscais declaradas pelo candidato também estão disponíveis em uma seção da página do candidato. A Figura 5 mostra como elas são exibidas. Porém, vale ressaltar que não temos necessariamente uma nota fiscal ligada a cada despesa, logo, o candidato pode ter uma despesa registrada sem uma nota fiscal.

Figura 5 – Página com informações das notas fiscais informadas pelo candidato

Contratante	CNPJ Emitente	Nome Emitente	Natureza Op.	Modelo	Data Emissão	Nº NF	Nº Série	Valor	UE	Unidade Arrecadadora	Ue	Chave
Titular	79.283.065/0001-41	ORBENK ADMINISTRACAO E SERVICOS LTDA.	SERV	00	10/10/2024	220811	1	R$ 22.142,28	SC	PREFEITURA MUNICIPAL DE JOINVILLE	JOINVILLE	66D596E8-9C0C-D4DE-21EA-06EB33I
Titular	51.207.912/0001-70	POSTO ILHA DA MAGIA LTDA	VEND	55	05/10/2024	35	010	R$ 144,47	BR	RFB_TSE	BRASIL	42241051207912000170550100000(
Titular	51.207.912/0001-70	POSTO ILHA DA MAGIA LTDA	VEND	55	05/10/2024	39	010	R$ 143,41	BR	RFB_TSE	BRASIL	42241051207912000170550100000(
Titular	51.207.912/0001-70	POSTO ILHA DA MAGIA LTDA	VEND	55	05/10/2024	1	011	R$ 141,46	BR	RFB_TSE	BRASIL	42241051207912000170550110000(
Titular	51.207.912/0001-70	POSTO ILHA DA MAGIA LTDA	VEND	55	05/10/2024	32	010	R$ 301,91	BR	RFB_TSE	BRASIL	42241051207912000170550100000(
Titular	51.207.912/0001-70	POSTO ILHA DA MAGIA LTDA	VEND	55	05/10/2024	37	010	R$ 149,09	BR	RFB_TSE	BRASIL	42241051207912000170550100000(
Titular	51.207.912/0001-70	POSTO ILHA DA MAGIA LTDA	VEND	55	05/10/2024	42	010	R$ 152,61	BR	RFB_TSE	BRASIL	42241051207912000170550100000(
Titular	51.207.912/0001-70	POSTO ILHA DA MAGIA LTDA	VEND	55	05/10/2024	34	010	R$ 121,70	BR	RFB_TSE	BRASIL	42241051207912000170550100000(

Fonte: Captura de tela retirada de (TRIBUNAL SUPERIOR ELEITORAL, 2024).

Além disso, também disponibiliza informações cadastrais completas dos candidatos, como nome, número de urna, partido, bens declarados e situação da candidatura. A Figura 6 ilustra como esses dados são informados na página da fonte.

Figura 6 – Página com informações de candidatura

Fonte: Captura de tela retirada de (TRIBUNAL SUPERIOR ELEITORAL, 2024).

Como a fonte não oferece nenhum mecanismo de download dos dados de cada candidato, esses dados foram extraídos a partir de um scraper, desenvolvido pelo autor.

O script automatiza a navegação por todos os municípios de SC, percorre cada tipo de cargo, consulta o perfil individual e coleta os dados de cada candidato, gerando uma lista de objetos JSON que contém os dados de cada candidato. A estrutura desse scraper dos dados coletados por ele será detalhada adiante.

Vale ressaltar que as informações sobre doadores, fornecedores e despesas estão disponíveis apenas para a eleição de 2024. O mais perto que a fonte das eleições de 2022 chega nessa parte financeira é com a disponibilização das notas fiscais, que também estão presentes na fonte de 2024.

3.2 SCRAPPING DOS DADOS DAS ELEIÇÕES DE 2024

A técnica de web scraping[3] consiste em extrair automaticamente dados estruturados de sites e aplicações web. Trata-se de um recurso essencial quando informações relevantes para análises não estão diretamente disponíveis para download ou estão apresentadas apenas em formatos não estruturados, dificultando sua utilização direta em análises automatizadas.

No contexto deste trabalho, o scraping foi empregado para coletar dados eleitorais das eleições municipais de 2024, disponíveis publicamente no sistema DivulgaCandContas do Tribunal Superior Eleitoral (TSE)[4]. Embora o TSE forneça diversas informações detalhadas sobre os candidatos, suas despesas, bens declarados, doações recebidas e notas fiscais emitidas durante a campanha eleitoral, esses dados não estão prontamente disponíveis para download em formatos estruturados, como CSV ou JSON.

Diante dessa limitação, tornou-se imprescindível a construção de um scraper personalizado utilizando Python e o framework Scrapy, que possibilitam a extração organizada e robusta dessas informações. O script automatiza a coleta de dados dos candidatos às eleições municipais de 2024 em Santa Catarina, o qual começa consultando uma lista completa dos municípios do estado e depois percorre cada cargo político disputado (prefeitos, vice-prefeitos e vereadores). O código fonte do scraper está disponível no apêndice A.

Para cada cargo em cada município, obtém uma lista dos candidatos inscritos e realiza consultas adicionais para coletar detalhes individuais, como nome, gênero, data de nascimento, partido e situação atual da candidatura. Em seguida, busca informações financeiras detalhadas, incluindo bens declarados, despesas contratadas, doações recebidas, despesas com fornecedores e notas fiscais eletrônicas.

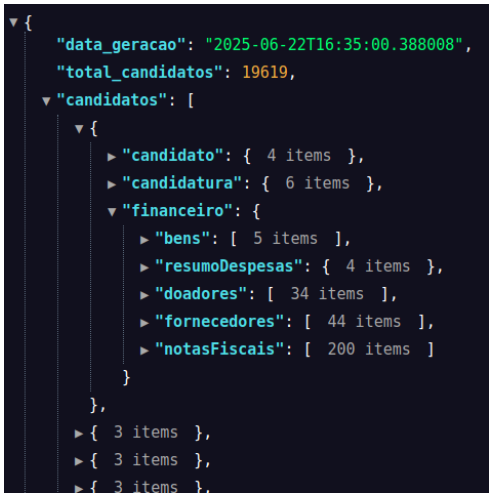
Os dados de cada candidato são organizados em objetos JSON padronizados, que estruturam os dados pessoais do candidato, da sua candidatura naquela eleição e por último as informações financeiras declaradas pelo candidato. O scraper retorna

[3] https://www.geeksforgeeks.org/blogs/what-is-web-scraping-and-how-to-use-it/

[4] https://divulgacandcontas.tse.jus.br/divulga/#/candidato/SUL/SC/2045202024

uma lista desses objetos que representam os candidatos, assim como outros detalhes da sua execução. A Figura 7 mostra a estrutura do JSON retornado pelo scraper. A seguir, os campos do objeto JSON referente a um candidato são detalhados e exemplificados.

Figura 7 – Estrutura geral do JSON retornado pelo scraper



Candidato e Candidatura

Os primeiros dados coletados pelo scraper para cada candidato são as suas informações pessoais e sobre a sua candidatura (partido, cargo, numero da urna, etc). O trecho de código abaixo (Listing 3.2.1) mostra um exemplo de como esses dados são estruturados no JSON de saída.

```
{
  "candidato": {
    "nome": "TOPAZIO SILVEIRA NETO",
    "sexo": "MASC.",
    "dataNasc": "1962-04-16",
    "tituloEleitor": "005053790906"
  },
  "candidatura": {
    "numUrna": 55,
    "cidade": "FLORIANÓPOLIS",
    "estado": "SC",
    "cargo": "Prefeito",
    "situacaoAtual": "Eleito",
    "partido": {
      "numero": 55,
      "sigla": "PSD",
      "nome": "Partido Social Democrático"
    }
  }
}
```

Listing 3.2.1 – Exemplo de dados sobre o candidato e sua candidatura no objeto JSON

Bens

Também foram coletados dados sobre os bens declarados do candidato. Cada objeto de candidato possui uma lista com os bens, como mostrado no quadro abaixo (Listing 3.2.2)


```
{
  ...
  "financeiro": {
    "bens": [
      {
        "ordem": 2,
        "descricao": "VIA BC",
        "descricaoTipoBem": null,
        "valor": 1440000.0,
        "dataUltimaAtualizacao": "2024-08-08"
      },
      {
        "ordem": 4,
        "descricao": "POUPANÇA",
        "descricaoTipoBem": null,
        "valor": 536.35,
        "dataUltimaAtualizacao": "2024-08-08"
      },
      {
        "ordem": 5,
        "descricao": "APLICAÇÃO",
        "descricaoTipoBem": null,
        "valor": 967.11,
        "dataUltimaAtualizacao": "2024-08-08"
      },
      ...
    ],
    ...
  }
}
```


Listing 3.2.2 – Exemplo de um trecho inicial da lista de bens de um candidato no objeto JSON

Doadores

Um aspecto particularmente relevante dessa fonte é a disponibilização dos CPFs ou CNPJs dos doadores pessoas físicas, o que permite a identificação precisa dos indivíduos envolvidos no financiamento das campanhas. Essa informação possibilita o cruzamento direto com outras bases públicas e privadas, ampliando a capacidade de análise sobre vínculos econômicos e relações suspeitas entre doadores e candidatos.

No quadro abaixo (Listing 3.2.3) vemos uma parte de uma lista de doadores extraída pelo scraper, que coleta também o nome do doador e o seu CPF ou CNPJ. Cada doador possui uma lista de objetos que representam uma doação feita por ele, que detalham o valor, data e outras informações sobre essa doação.

```
{
  ...
  "financeiro": {
    ...
    "doadores": [
      {
        "cpfCnpj": "13629827000100",
        "nome": "Direção Nacional - Partido Social Democrático",
        "doacoes": [
          {
            "nr_recibo_eleitoral": "000551181051SC000024E",
            "nr_documento": "13629827000100",
            "data": "26/08/2024",
            "ds_receita": null,
            "valor": 3000000,
            "especie_recurso": "Transferência eletrônica"
          }
        ]
      },
      {
        "cpfCnpj": "08517423000195",
        "nome": "Direção Nacional - Partido Liberal",
        "doacoes": [
          {
            "nr_recibo_eleitoral": "000551181051SC000028E",
            "nr_documento": "082024",
            "data": "31/08/2024",
            "ds_receita": null,
            "valor": 1850000,
            "especie_recurso": "Transferência eletrônica"
          }
        ]
      },
      ...
    ]
    ...
  }
}
```


Listing 3.2.3 – Exemplo de um trecho inicial da lista de doadores de um candidato no objeto JSON

Fornecedores e Despesas

Também relacionado com o lado financeiro, essa fonte disponibiliza informações sobre os fornecedores contratados pelo candidato e as despesas pagas a eles. O scraper coleta esse dados e forma uma lista com todos os fornecedores, que são identificados pelo nome e CPF ou CNPJ. Cada fornecedor também possui uma lista de objetos que representam despesas, os quais descrevem o valor, data e outros detalhes dessas despesas. O quadro abaixo (Listing 3.2.4) mostra exemplos de alguns fornecedores e suas despesas que foram coletados pelo scraper.

```
{
  ...
  "financeiro": {
    ...
    "fornecedores": [
      {
        "cpfCnpj": "14899615000106",
        "nome": "PARTIDO SOCIAL DEMOCRATICO - FLORIANOPOLIS- SC- MUNICIPAL",
        "despesas_pagas": [
          {
            "valor": 45.29,
            "data": "30/10/2024",
            "tipo": "Doações financeiras a outros candidatos/partidos",
            "descricao": null,
            "especie_documento_emitido": null
          }
        ]
      },
      {
        "cpfCnpj": "51207912000170",
        "nome": "POSTO ILHA DA MAGIA LTDA",
        "despesas_pagas": [
          {
            "valor": 144.11,
            "data": "27/09/2024",
            "tipo": "Combustíveis e lubrificantes",
            "descricao": "COMBUSTÍVEL PLACA RYX0H66",
            "especie_documento_emitido": "Nota Fiscal"
          },
          ...
        ]
      },
      ...
    ]
    ...
  }
}
```


Listing 3.2.4 – Exemplo de um trecho inicial da lista de fornecedores de um candidato no objeto JSON

O scraper também coleta informações resumidas com relação as despesas totais do candidato, como mostrado no quadro abaixo.


```
{
  ...
  "financeiro": {
    ...
    "resumoDespesas": {
      "limiteDeGastos": 159850.76,
      "totalDespesasContratadas": 156785.0,
      "totalDespesasPagas": 156785.0,
      "doacoesParaOutrosCandidatosOuPartidos": 0.0
    },
    ...
  }
}
```


Listing 3.2.5 – Exemplo do resumo das despesas de um candidato no objeto JSON

Notas Fiscais

A fonte também informa as notas fiscais declaradas pelo candidato durante sua campanha. São disponibilizadas diversas informações sobre as notas fiscais, como o CNPJ de quem emitiu a nota, o valor e data de emissão, a unidade eleitoral, a unidade arrecadadora, etc. A Figura 3.2.6 ilustra melhor todas essas informações e como estão estruturadas no objeto JSON.

Vale ressaltar que não temos as notas fiscais referentes a todas as despesas na maioria dos casos, e também não temos uma relação direta da despesa com a sua nota fiscal na fonte. Logo, podemos ter despesas sem nota fiscal e vice-versa.


```
{
  ...
  "financeiro": {
    ...
    "notasFiscais": [
      {
        "cnpjEmitente": "51207912000170",
        "dataEmissao": 1728097200000,
        "valor": 85.84,
        "ue": "BR",
        "unidadeArrecadadora": "RFB_TSE",
        "numeroNota": "38",
        "serie": "010",
        "chaveAcesso": "42241051207912000170550100000000381000041015",
        "urlAcesso": "https://www.nfe.fazenda.gov.br/portal/principal.aspx"
      },
      {
        "cnpjEmitente": "51207912000170",
        "dataEmissao": 1728097200000,
        "valor": 143.41,
        "ue": "BR",
        "unidadeArrecadadora": "RFB_TSE",
        "numeroNota": "39",
        "serie": "010",
        "chaveAcesso": "42241051207912000170550100000000391000041020",
        "urlAcesso": "https://www.nfe.fazenda.gov.br/portal/principal.aspx"
      },
      ...
    ]
    ...
  }
}
```


Listing 3.2.6 – Exemplo de um trecho inicial da lista de notas fiscais de um candidato no objeto JSON

3.3 ESQUEMA DO BANCO DE DADOS DESTINO

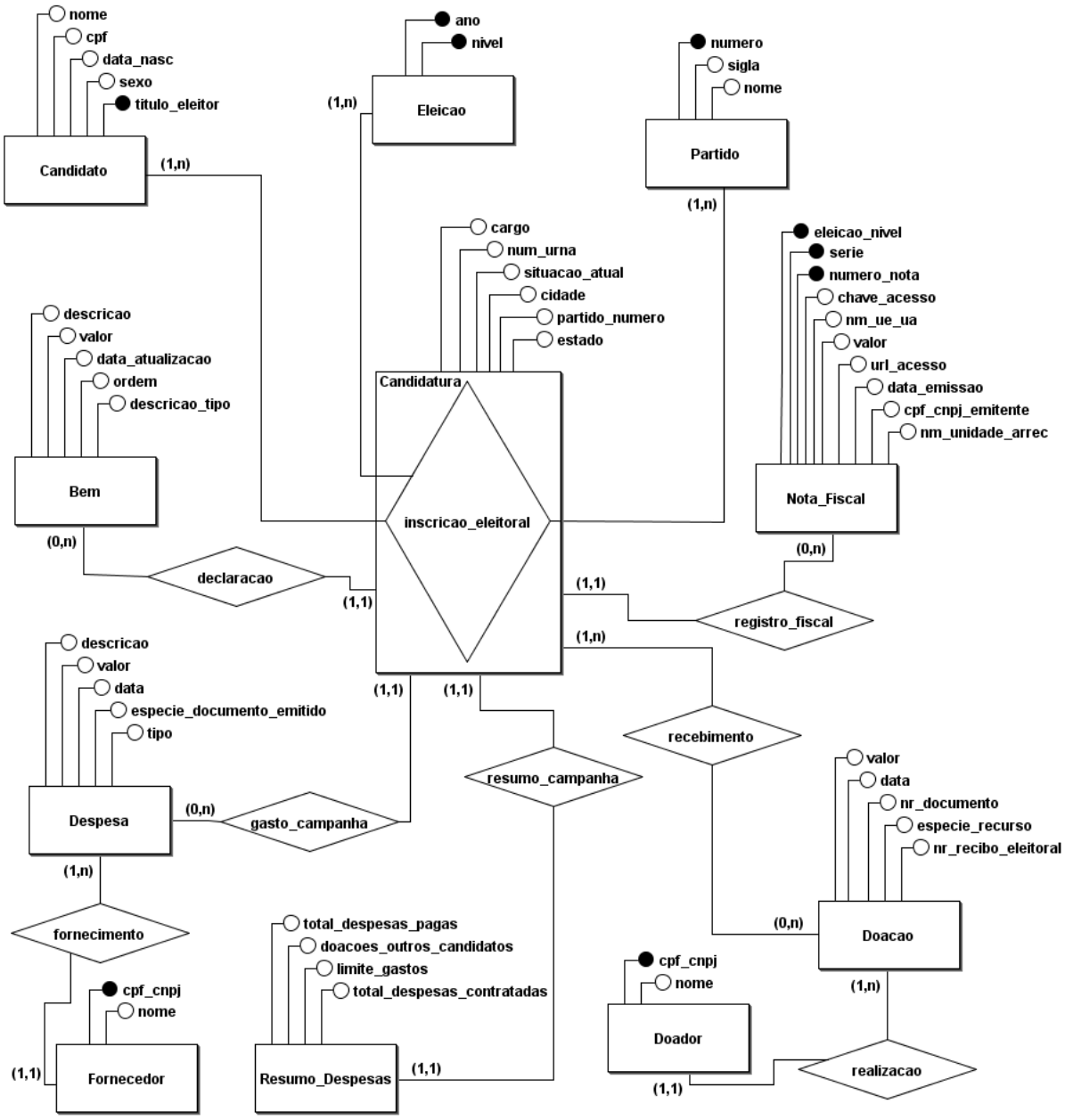
A estrutura do banco de dados é uma parte essencial no desenvolvimento de qualquer sistema que lide com grandes volumes de informações estruturadas. No contexto deste trabalho, optou-se pela utilização de um banco de dados relacional devido à sua adequação às necessidades específicas de armazenar e gerenciar dados eleitorais, incluindo informações pessoais dos candidatos, suas atividades financeiras e partidárias. Essa escolha foi feita principalmente porque os dados coletados das

fontes, como arquivos CSV fornecidos pelo Tribunal Superior Eleitoral (TSE), já se encontram estruturados em formatos tabulares, o que torna o modelo relacional a opção mais natural e eficiente. Outro motivo para essa escolha foi exemplificar um caso real em que dados originalmente não estruturados, especificamente o JSON obtido por *web scraping* das eleições de 2024, são transformados e armazenados em um banco relacional, indo ao encontro de um dos objetivos deste trabalho.

O esquema relacional apresentado na Figura 8 foi projetado para garantir compatibilidade entre as duas fontes de dados utilizadas no projeto. Campos e tabelas foram cuidadosamente elaborados de forma a refletir fielmente os dados presentes tanto nos arquivos CSV do TSE quanto nas informações obtidas pelo *scraper*, especialmente nas situações em que ambas as fontes fornecem os mesmos tipos de dados. Para garantir maior clareza e praticidade no desenvolvimento e manutenção do sistema, os campos das tabelas passaram por uma normalização, removendo caracteres especiais e padronizando a capitalização. Essa normalização contribui diretamente para a organização e legibilidade do código, facilitando as operações de leitura, escrita e manutenção no contexto do sistema.

A tabela principal deste esquema é a `Candidatura`, que conecta diversas entidades relevantes ao contexto eleitoral, como candidatos, partidos, eleições e atividades financeiras. Esta tabela centraliza informações fundamentais sobre cada candidatura e está diretamente relacionada a entidades como `Candidato`, `Partido` e `Eleicao`.

Figura 8 – Esquema do banco de dados



Fonte: O autor.

4 PROCESSOS DE ETL

Este capítulo detalha a implementação prática dos processos de Extração, Transformação e Carga (ETL) utilizando as ferramentas Apache Airflow e Dagster. O objetivo é demonstrar como cada orquestrador foi aplicado para processar os dados das eleições de 2022 (a partir de arquivos CSV) e 2024 (a partir de um arquivo JSON), carregando-os em um banco de dados relacional unificado.

Ambas as implementações seguem a mesma lógica conceitual de ETL, mas o fazem por meio de paradigmas de desenvolvimento e funcionalidades distintas. As seções a seguir descrevem a arquitetura geral do processo de ETL e também como os conceitos específicos de cada ferramenta foram utilizados para estruturar os pipelines e gerenciar dependências durante o desenvolvimento com cada ferramenta. Ao final, uma análise comparativa destaca as principais diferenças, vantagens e desvantagens observadas durante o desenvolvimento.

4.1 ARQUITETURA COMUM DO PIPELINE DE ETL

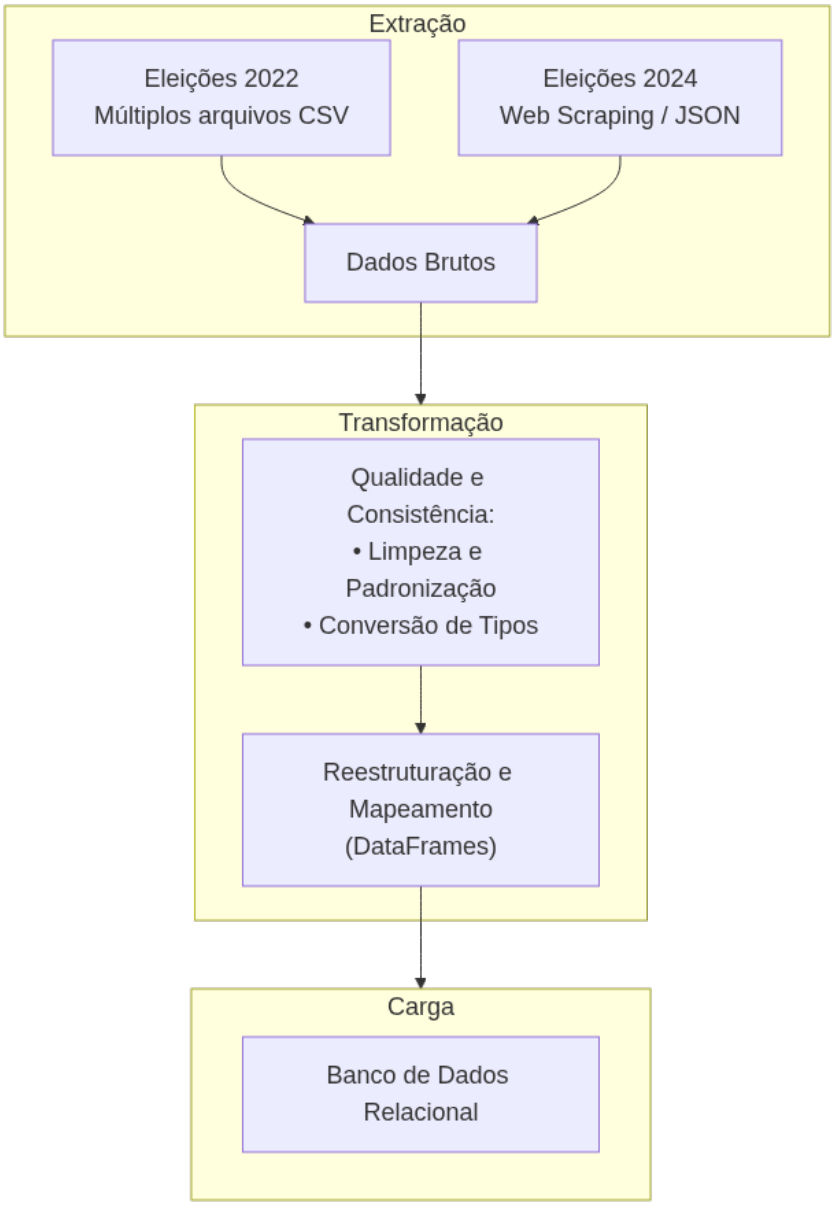
Antes de detalhar as implementações específicas em Airflow e Dagster, é fundamental apresentar a arquitetura de referência que orientou ambos os desenvolvimentos. Esta especificação comum estabelece um critério uniforme de avaliação, garantindo que a comparação entre as ferramentas se concentre em suas capacidades de orquestração, e não em diferenças na manipulação dos dados. A arquitetura foi concebida seguindo três etapas principais: as fontes, as regras de transformação e o modelo de destino, ilustrados na Figura 10.

4.1.1 Extração dos Dados

O ponto de partida do pipeline consiste na extração de dados de duas fontes heterogêneas referentes aos pleitos de 2022 e 2024: em 2022, as informações estão disponibilizadas em múltiplos arquivos no formato CSV, já em 2024, os dados são obtidos por *web scraping* e consolidados em um único arquivo no formato JSON.

No escopo deste trabalho, esses arquivos foram incluídos como recursos locais no diretório do projeto. Essa abordagem simplifica a camada de extração, permitindo que o pipeline acesse os dados diretamente do sistema de arquivos e mantendo o foco da análise nas capacidades de orquestração e transformação das ferramentas. Além disso, por conta do caráter imutável dos dados, por se tratarem de eleições passadas, viabiliza uma estratégia de extração única e completa. Isso elimina a complexidade associada a pipelines que necessitam de sincronização ou atualização incremental com a fonte.

Figura 9 – Diagrama da Arquitetura de ETL



Fonte: O autor.

4.1.2 Regras de Transformação de Dados

Uma vez extraídos, os dados brutos são submetidos a um processo de transformação padronizado, que constitui o núcleo da lógica de processamento. Esta etapa, idêntica em ambas as implementações para garantir uma comparação justa, aborda primeiramente a qualidade e a consistência dos dados. Operações de limpeza e padronização são aplicadas para converter tipos de dados brutos, como strings, em formatos adequados (datas e valores numéricos), normalizar conteúdo textual através de capitalização e remoção de espaços, e tratar valores nulos ou inválidos.

Após a garantia da qualidade dos dados individuais, a transformação foca na reestruturação e no mapeamento para o modelo de destino. Nesta fase, a estrutura

heterogênea das fontes é desmembrada em um formato tabular e relacional. Isso resulta na criação de múltiplos conjuntos de dados, cada um projetado para popular uma tabela específica do banco de dados, como `Candidato`, `Despesa` e `Doacao`.

4.1.3 Carga dos Dados

A etapa final do fluxo é carregar os dados transformados em um repositório unificado. O alvo de ambos os fluxos de trabalho é o esquema relacional apresentado na Figura 8. Para garantir a robustez do processo, a carga é realizada com uma estratégia de *upsert* (inserir ou atualizar). Esta abordagem garante a idempotência do pipeline, o que significa que reexecuções não causarão duplicatas de dados ou falhas por violação de chaves primárias, tornando o processo confiável e resiliente.

4.1.4 Arquitetura de Execução em Containers

Para garantir a consistência e a portabilidade das implementações, ambas as aplicações foram executadas em ambientes totalmente containerizados com o uso de Docker. Essa abordagem isola as dependências e a configuração de cada ferramenta, eliminando variações de ambiente e simplificando a instalação. A arquitetura de execução, embora específica para cada ferramenta, compartilha componentes e estratégias em comum.

Tanto no Airflow quanto no Dagster, a arquitetura é composta por múltiplos containers que se comunicam em uma rede Docker interna. Cada ferramenta utiliza um container para sua interface de usuário (*webserver*) e um ou mais para os componentes de orquestração (como o *scheduler* e *worker* no Airflow, e o *daemon* no Dagster).

Um ponto central da arquitetura comum é a separação dos bancos de dados. Ambos os setups utilizam dois bancos de dados PostgreSQL distintos, cada um em seu próprio container:

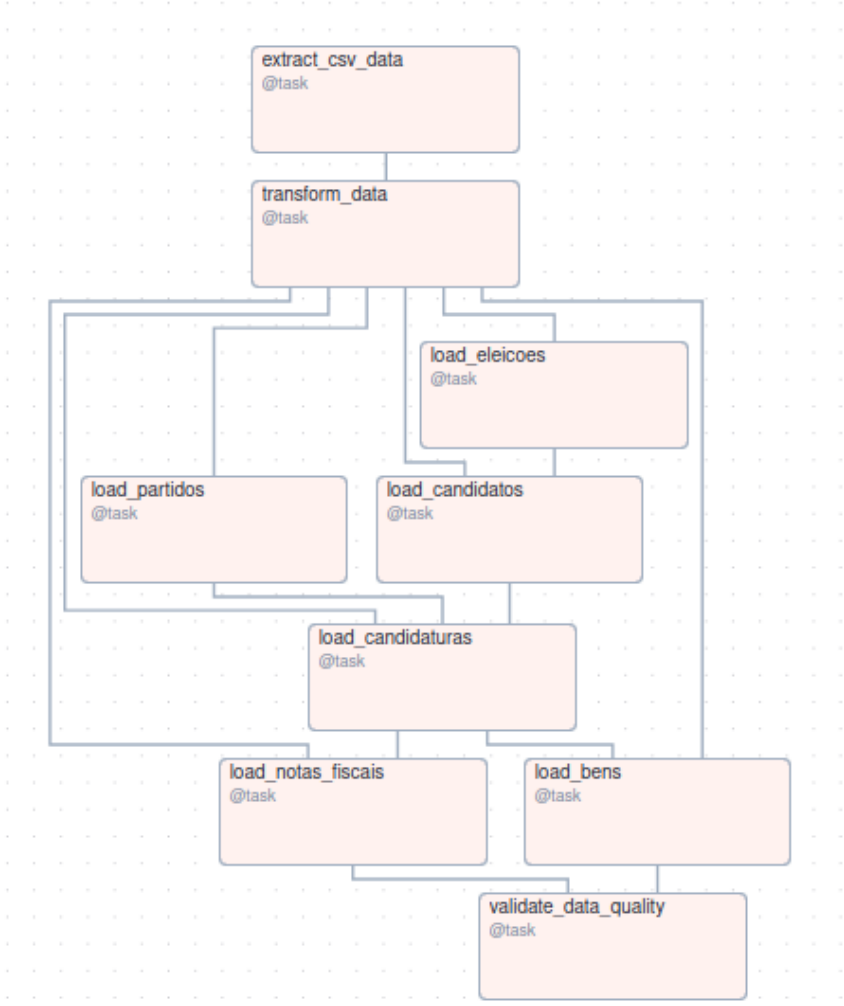
1. **Banco de Dados de Metadados**: Um banco de dados dedicado para a ferramenta de orquestração armazenar seu estado interno, como o histórico de execuções, o status das tarefas e as configurações.
2. **Banco de Dados da Aplicação**: Um segundo banco de dados, nomeado `candidatos_sc`, que serve como o destino final para os dados processados pelos pipelines de ETL.

Essa separação garante que o armazenamento de metadados da ferramenta de orquestração seja desacoplado dos dados da aplicação, sendo uma boa prática de arquitetura. Além disso, ambos os ambientes montam os diretórios de código-fonte como volumes, permitindo o desenvolvimento iterativo sem a necessidade de reconstruir as imagens dos containers a cada alteração.

4.2 AIRFLOW

A implementação com o Apache Airflow foi estruturada em torno do seu conceito central: o Grafo Acíclico Direcionado (DAG). Para atender às particularidades de cada fonte de dados, foram desenvolvidos dois DAGs distintos: um para o processamento dos múltiplos arquivos CSV das eleições de 2022 e outro para o arquivo JSON único das eleições de 2024. Essa abordagem, comum em projetos com Airflow, permite isolar as lógicas de extração e transformação, que variam entre as fontes, mantendo a modularidade.

Figura 10 – DAG para o ETL dos dados de 2022



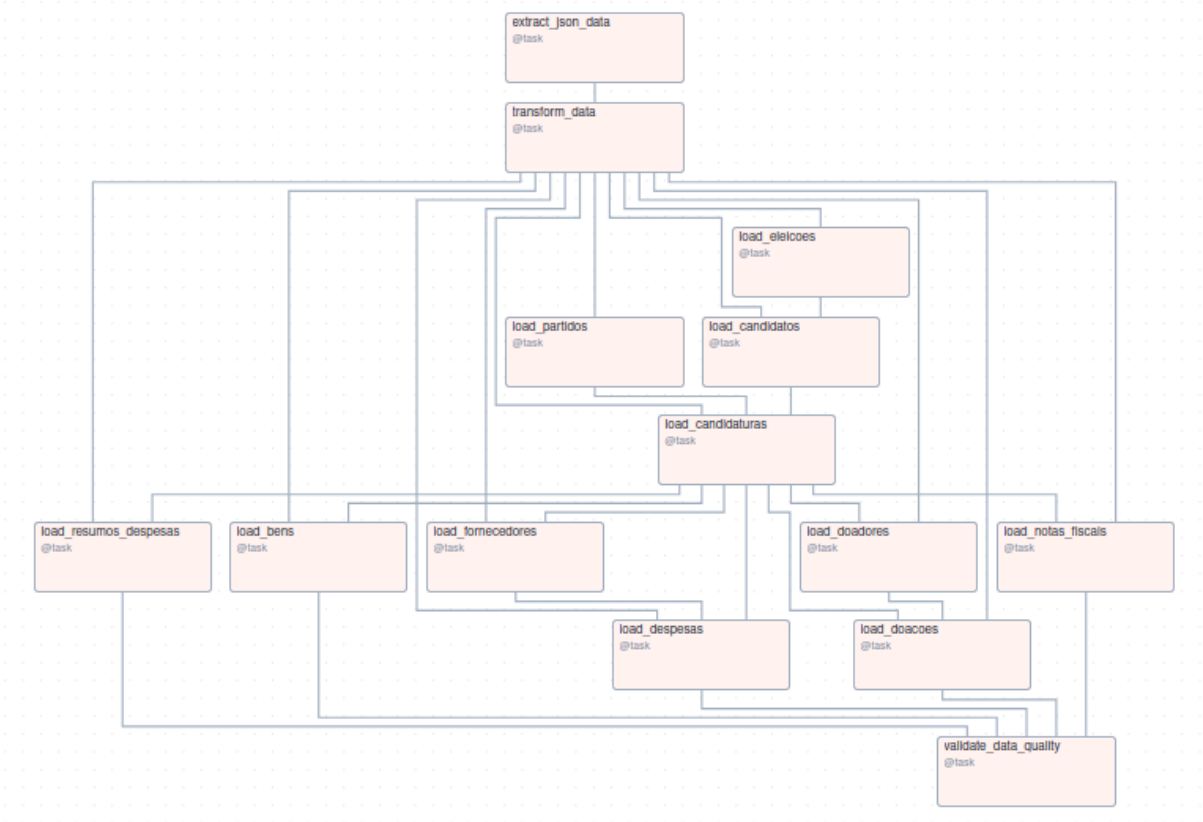
Fonte: Captura de tela da DAG na interface do Airflow.

Para a implementação das DAGs, foi utilizada a *TaskFlow API* do Airflow. Essa abordagem, detalhada na Seção 2.2, permite a criação de pipelines de forma mais intuitiva e declarativa, tratando tarefas como funções Python e gerenciando dependências por meio da passagem de seus resultados, o que simplifica o código e melhora sua legibilidade.

A arquitetura comum a ambos os fluxos inicia com uma única tarefa de extração, seguida por uma tarefa central de transformação. A partir daí, o design adota um padrão de *fan-out*, onde os dados processados são desmembrados e carregados em paralelo para as diversas tabelas do banco de dados. Finalmente, as trilhas de execução convergem para uma única tarefa de validação, que assegura a integridade e a qualidade dos dados persistidos.

A distinção crucial entre os dois pipelines reside na lógica contida na tarefa de transformação. No DAG de 2022 (Figura 10), esta etapa enfrenta o desafio de integrar fontes de dados com mecanismos de relacionamento inconsistentes. Enquanto a relação entre os arquivos de candidatos e seus bens pode ser estabelecida de forma direta através de um identificador sequencial único, a correlação com as notas fiscais é mais complexa, onde o único campo comum é o número de urna do candidato, exigindo uma lógica de mapeamento mais elaborada para garantir a correlação correta.

Figura 11 – DAG para o ETL dos dados de 2024



Fonte: Captura de tela da DAG na interface do Airflow.

Em contraste, a tarefa de transformação do DAG de 2024 (Figura 11) é logicamente mais simples. A fonte de dados, um único arquivo JSON, já possui uma estrutura aninhada que agrupa todas as informações de um candidato. Portanto, a principal responsabilidade da transformação não é a correlação, mas o "desaninha-

mento"da estrutura hierárquica do JSON, extraindo e formatando os dados para cada uma das tabelas de destino.

Outra diferença é visualmente evidente nos grafos. O DAG de 2024 exibe um *fan-out* muito mais amplo na fase de carga, já que possui uma diversidade de dados maior, como os doadores, fornecedores e despesas. Assim, enquanto a complexidade do pipeline de 2022 está concentrada na lógica da transformação, a complexidade do pipeline de 2024 se manifesta na amplitude da carga de dados e no gerenciamento correto das dependências entre as tasks dessa última etapa, garantindo que as inserções no banco sejam feitas na ordem correta.

Para executar essa etapa de carga de forma robusta e segura, a comunicação com o banco de dados foi inteiramente gerenciada pelo `PostgresHook`, um componente nativo do Airflow que abstrai e centraliza a conexão com o PostgreSQL. Seu uso foi encapsulado em funções auxiliares e utiliza o sistema de conexões centralizado do Airflow, eliminando a necessidade de armazenar credenciais sensíveis diretamente no código da DAG.

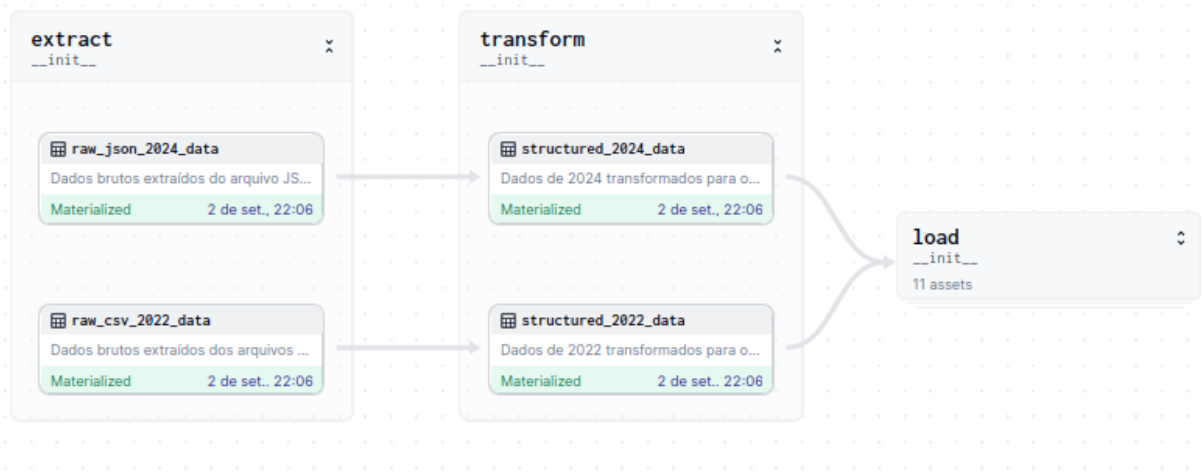
4.3 DAGSTER

A implementação com Dagster adota uma abordagem fundamentalmente diferente daquela do Airflow, centrada no conceito de *assets* (ativos de dados). Em vez de definir um grafo de tarefas, o pipeline é modelado como um grafo de ativos, onde cada ativo representa um objeto de dados persistente, como um arquivo bruto, um conjunto de dados transformado ou uma tabela no banco de dados. Essa filosofia foca no "o quê"(os dados) em vez do "como"(as tarefas), proporcionando uma visão mais clara da linhagem e do estado dos dados.

Cada ativo é uma função Python decorada com `@asset`, que produz um artefato de dados. As dependências entre os ativos são definidas pelo Dagster com base nos argumentos das funções, que se definidos com o mesmo nome, são automaticamente inferidos como dependências.

A lógica de extração e transformação, por sua vez, foi implementada em ativos distintos para cada fonte, já que cada fonte tem tipos de arquivo e necessidades de transformação diferentes. Como o grafo de ativos é único em uma instância do Dagster, foram criados grupos de ativos para cada etapa do processo, trazendo uma visão mais modular e clara da linhagem de dados. A Figura 12 mostra o grafo de ativos na interface do Dagster, destacando os grupos de ativos para cada etapa do processo.

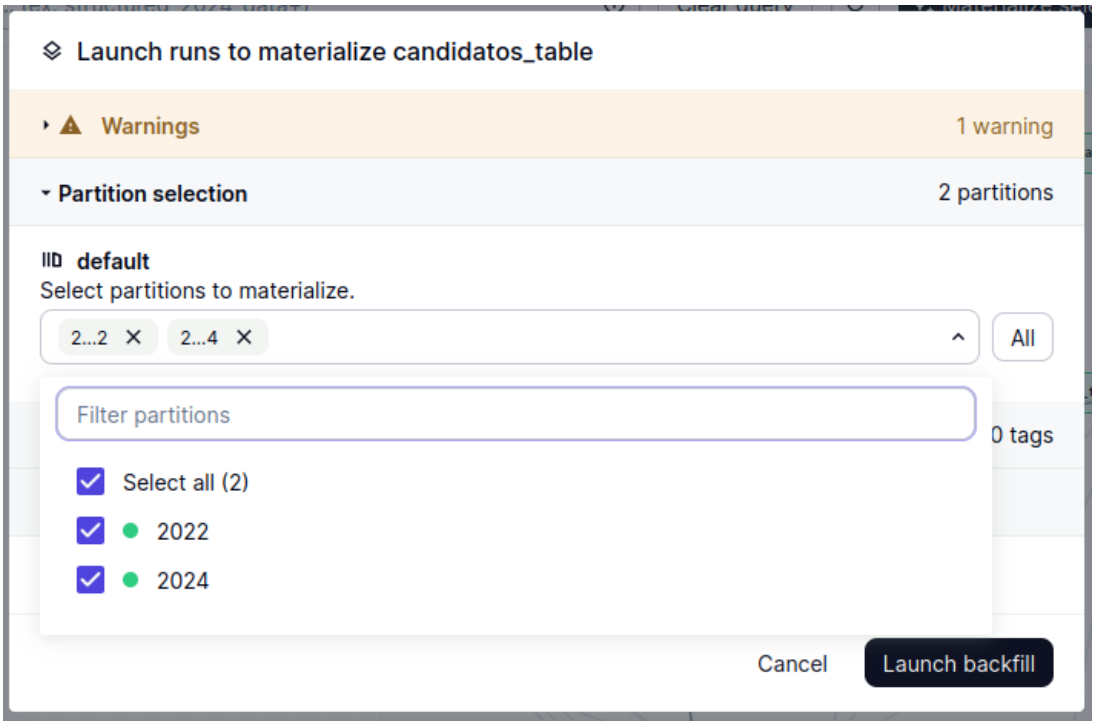
Figura 12 – Grafo de ativos global na interface do Dagster



Fonte: Captura de tela da interface do Dagster.

Uma das principais diferenças da utilização do Dagster é a capacidade de gerenciar os dados de 2022 e 2024 de forma unificada e parametrizada através do uso de *Partitions* (partições). Em vez de criar dois fluxos de trabalho distintos como no Airflow, foi definida uma única partição estática com as chaves "2022"e "2024". Os ativos de carga são particionados, permitindo que a mesma lógica de carregamento seja executada seletivamente para cada ano. A Figura 13 ilustra como a interface do Dagster permite selecionar a partição desejada para executar um asset, simplificando a operação e a manutenção dos pipelines.

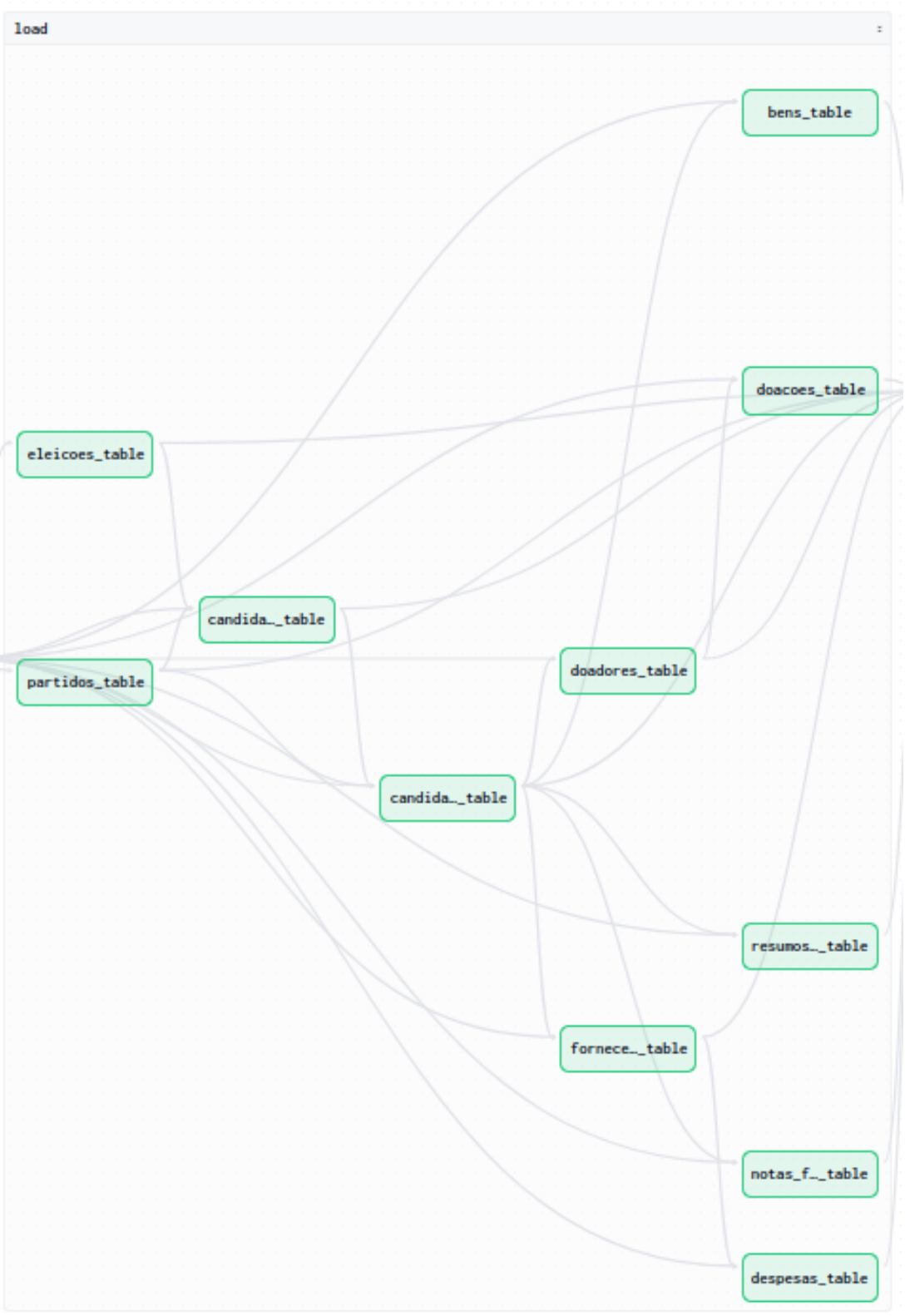
Figura 13 – Seleção de partição para execução de um asset na interface do Dagster



Fonte: Captura de tela da interface do Dagster.

A Figura 14 mostra o grupo de ativos de carga na interface do Dagster, destacando a utilização de partições para a criação de uma única lógica de carregamento para as duas fontes de dados.

Figura 14 – Grupo de ativos de load na interface do Dagster



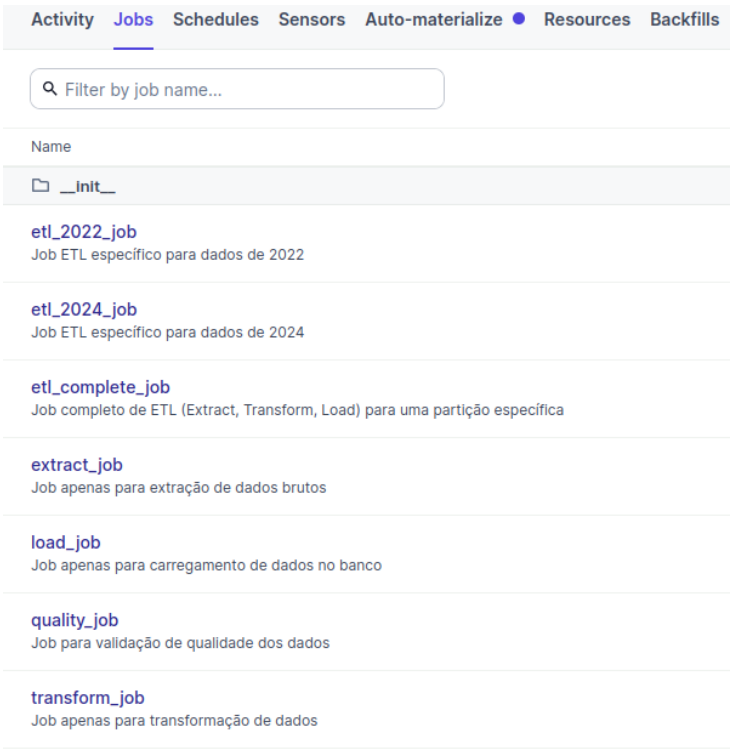
Fonte: Captura de tela da interface do Dagster.

A interação com o banco de dados foi gerenciada através de um *Resource*, um conceito do Dagster para lidar com sistemas externos. O `CandidatosPostgresResource` encapsula a lógica de conexão e execução de queries, sendo disponibilizado aos ativos através de injeção de dependência, o que promove o desacoplamento e facilita os testes.

Para automatizar ainda mais o pipeline, foi utilizada a política de `AutoMaterializePolicy`. Esta política instrui o Dagster a materializar automaticamente um ativo assim que todas as suas dependências upstream forem atualizadas com sucesso. Isso elimina a necessidade de agendamentos explícitos ou execuções manuais para a maioria do fluxo de trabalho, tornando o pipeline reativo e garantindo que os dados estejam sempre atualizados em cascata a partir de uma nova extração.

Por último, com o intuito de proporcionar uma execução mais modular e controlada dos ativos, foram definidos múltiplos *Jobs*. No Dagster, um *Job* seleciona um subconjunto específico de ativos do grafo global, servindo a um propósito análogo ao de uma `DAG` no Airflow: definir um fluxo de trabalho concreto e executável. Essa capacidade foi utilizada para criar diferentes escopos de execução, como um `etl_complete_job` para o pipeline completo, e jobs mais granulares como `extract_job` e `load_job`. Essa flexibilidade agilizou o desenvolvimento e os testes, ao permitir a execução isolada apenas dos ativos relevantes para uma determinada alteração. A Figura 15 mostra os jobs configurados na interface do Dagster.

Figura 15 – Jobs na interface do Dagster



Fonte: Captura de tela da interface do Dagster.

5 ANÁLISE COMPARATIVA

A implementação dos pipelines de ETL com Airflow e Dagster, embora seguindo a mesma arquitetura conceitual, revelou diferenças significativas nas práticas de desenvolvimento, orquestração e manutenção. Esta seção apresenta uma análise comparativa detalhada, focada na experiência prática obtida durante o desenvolvimento dos fluxos de dados para as eleições de 2022 e 2024. O objetivo é destacar os pontos fortes e as limitações de cada ferramenta no contexto deste projeto, oferecendo uma visão clara sobre em que cenários cada uma se sobressai.

5.1 EXPERIÊNCIA DE DESENVOLVIMENTO

A experiência de desenvolvimento é um fator crucial na escolha de uma ferramenta de orquestração, pois impacta diretamente a produtividade da equipe, a manutenibilidade do código e a velocidade de iteração. A seguir, são analisados os principais aspectos que influenciaram essa experiência no presente trabalho.

5.1.1 Complexidade Conceitual

Uma análise quantitativa da documentação oficial expõe diferenças notáveis na carga conceitual inicial de cada ferramenta. No Airflow, um desenvolvedor é introduzido a um núcleo de aproximadamente oito conceitos principais, que incluem desde os blocos de construção básicos como *DAG*s, *Operators* e *Tasks* (APACHE SOFTWARE FOUNDATION, 2025a), até abstrações mais recentes como a *TaskFlow API* (APACHE SOFTWARE FOUNDATION, 2025b). Em contrapartida, a documentação do Dagster detalha um conjunto de dezesseis abstrações centrais, como *Assets*, *Jobs* e *Resources* (DAGSTER. . . , 2025). Essa disparidade sugere, inicialmente, uma barreira de entrada conceitual maior no Dagster.

No entanto, para avaliar a complexidade real de implementação, podemos utilizar o Workflow Reference Model (WfRM) (HOLLINGSWORTH, 1995) como uma base teórica. Este modelo estabelece uma arquitetura e terminologia padrão para sistemas de workflow, permitindo-nos mapear e comparar as abstrações de cada ferramenta de forma neutra. Ao nos concentrarmos nos conceitos estritamente necessários para construir os pipelines de ETL propostos neste trabalho, a análise se torna mais refinada. A Tabela 2 apresenta um comparativo dos conceitos aplicados, enriquecido com seu mapeamento correspondente no WfRM.

Tabela 2 – Comparativo dos conceitos aplicados e seu mapeamento no WfRM

Conceito WfRM	**Descrição**	**Airflow**	**Dagster**
Process Activity	Unidade de trabalho	`Task`	`Asset`
Process Definition	Definição do workflow	`DAG`	`Grafo de Ativos`
Invoked Application I/F	Conexão externa	`Hook`	`Resource`

Como a tabela evidencia, no contexto específico deste projeto, ambos os frameworks exigiram a compreensão de um número similar de abstrações centrais. Isso demonstra que a complexidade real da implementação não residiu na quantidade de conceitos, mas sim na natureza paradigmática e na sofisticação de suas implementações em relação ao modelo de referência.

A primeira e mais fundamental divergência de paradigma reside na definição da lógica, correspondente ao conceito de *Process Activity* do WfRM. O Airflow adere fielmente a este modelo, onde uma *Task* representa uma unidade discreta de trabalho. O Dagster, por sua vez, estende este conceito, onde um *Asset* representa não apenas a atividade, mas também o artefato de dados que ela produz. Essa abordagem centrada em dados, embora mais distante do modelo original focado em processos, oferece uma semântica mais rica para a engenharia de dados moderna, onde a linhagem e o estado dos dados são cruciais para a observabilidade e a manutenção.

Essa diferença se estende à forma como cada ferramenta implementa a *Invoked Application Interface* do WfRM para se conectar a sistemas externos. A implementação do Airflow com *Hooks* e *Connections* é mais procedural e imperativa. Em contraste, a abordagem do Dagster com *Resources* e injeção de dependência representa uma implementação mais madura e desacoplada, alinhada com princípios modernos de engenharia de software que vão além do escopo original do WfRM.

Outro ponto é a necessidade de executar o mesmo pipeline para dados de 2022 e 2024 expôs uma lacuna no modelo clássico de workflow, que não padroniza a parametrização de execuções. O Dagster aborda isso nativamente com *Partitions*, um conceito que parametriza tanto a execução quanto a materialização dos dados. O Airflow, por não possuir uma abstração equivalente, exigiu uma implementação manual dessa lógica, resultando na duplicação do código de orquestração sob a forma de dois DAGs distintos.

Em suma, a análise revela que o Dagster, apesar de introduzir mais conceitos em sua documentação completa, implementa as abstrações fundamentais do WfRM com uma camada adicional de sofisticação e foco em dados. Sua abordagem opinativa internaliza complexidades comuns da engenharia de dados, como linhagem, particionamento e gerenciamento de recursos, que no Airflow são deixadas para o desenvolvedor resolver. Portanto, a complexidade é percebida de formas diferentes, sendo que o Dagster é conceitualmente mais denso, mas pode simplificar a implementação de padrões robustos, e já o Airflow é conceitualmente mais simples, mas pode

exigir maior complexidade no código para alcançar o mesmo nível de robustez.

5.1.2 Setup Inicial e Infraestrutura

Ambas as ferramentas foram configuradas em ambientes containerizados utilizando Docker, porém com níveis de complexidade arquitetural distintos. O Airflow, seguindo o arquivo `docker-compose.yml` oficial[1], exigiu a orquestração de cerca de 9 serviços interdependentes para habilitar sua arquitetura distribuída. Este setup inclui componentes de infraestrutura robustos como o *postgres* (banco de metadados) e o *redis* (broker de mensagens), essenciais para a comunicação assíncrona entre o *scheduler* e os *workers*. Além destes, o ambiente instancia o *webserver*, o *triggerer*, serviços de inicialização (*airflow-init*) e utilitários de linha de comando. Essa abordagem garante consistência com ambientes de produção em escala, mas resulta em um tempo de setup inicial mais longo e em um consumo de recursos locais significativamente maior.

O Dagster, por outro lado, adota uma arquitetura modular que separa o plano de controle do código do usuário, resultando em um setup local mais enxuto conforme a documentação[2]. A configuração básica utilizada envolveu três componentes principais: o *dagster-webserver* (responsável pela interface gráfica), o *dagster-daemon* (processo de longa duração que gerencia agendamentos, sensores e execuções) e o contêiner de banco de dados *postgresql* para persistência. Diferente do Airflow, o Dagster não exige nativamente um broker de mensagens (como o Redis) para o funcionamento básico local, comunicando-se diretamente com o ambiente de execução do código. Isso tornou a experiência de setup mais direta, permitindo iniciar rapidamente o ambiente de desenvolvimento, mas, em contrapartida, resulta em uma arquitetura mais restrita que pode comprometer o desempenho quando comparada à robustez distribuída do Airflow.

Essa diferença no setup inicial representa um claro *trade-off*. O Airflow expõe o desenvolvedor à complexidade de uma arquitetura de produção desde o início, o que pode ser benéfico para o aprendizado e para evitar surpresas no *deploy*, mas ao mesmo tempo impõe uma barreira de entrada maior. Por outro lado, o Dagster prioriza a velocidade de iteração e a produtividade, oferecendo um ciclo de feedback mais rápido durante a fase inicial do desenvolvimento, o que se mostrou uma vantagem significativa na etapa de construção dos pipelines deste trabalho.

5.1.3 Reutilização de Código e Modularidade

A modularidade e a reutilização de código são pontos onde as diferenças de paradigma entre as ferramentas se tornaram mais evidentes. No Airflow, a abordagem

[1] https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html
[2] https://docs.dagster.io/deployment/oss/deployment-options/docker

para processar as duas fontes de dados (2022 e 2024) exigiu a criação de duas DAG's distintas, uma para cada ano. Embora funções auxiliares de transformação tenham sido compartilhadas em módulos Python, a lógica de orquestração para cada fluxo foi definida de forma independente. Isso resultou na duplicação de várias tarefas, principalmente as de carregamento de dados, cujo código era funcionalmente idêntico em ambos os fluxos, mas precisou ser redefinido em cada DAG. Essa característica levou a um total de 23 funções de tarefas para cobrir os dois cenários. A estrutura do Airflow, com cada DAG sendo um pipeline autocontido e independente, oferece modularidade no nível do fluxo de trabalho, mas pode levar a uma maior repetição de código quando múltiplos fluxos compartilham lógicas de orquestração semelhantes.

O Dagster, por outro lado, permitiu uma solução mais unificada através do conceito de *Partitions*. Foi definida uma única partição estática com as chaves "2022"e "2024", o que permitiu parametrizar os ativos de carga. A mesma função de um ativo pôde ser utilizada para carregar os dados de ambos os anos, selecionando a fonte de dados correta com base na chave da partição fornecida no contexto da execução. Essa abordagem eliminou a necessidade de duplicar a lógica de orquestração da carga, resultando em um código mais conciso e de fácil manutenção, totalizando 17 ativos para abranger o mesmo escopo. No entanto, essa abordagem resulta em um único grafo de linhagem de dados para toda a aplicação, o que pode ser considerado menos modular do que os múltiplos grafos isolados do Airflow. A forma mais próxima de segmentar o escopo de execução neste modelo é através da definição de *Jobs*, que orquestram subconjuntos específicos do grafo global de ativos.

Outro diferencial do Dagster na modularidade é o conceito de *Resources*. A conexão com o banco de dados foi encapsulada em um recurso dedicado, que centraliza toda a lógica de interação com o PostgreSQL. Este recurso é então injetado como um argumento nos ativos que o necessitam, um padrão de injeção de dependência que desacopla a lógica de negócio da infraestrutura externa. No Airflow, essa interação foi realizada com o `PostgresHook`, que, embora eficaz, é obtido de forma imperativa dentro das tarefas, resultando em um acoplamento ligeiramente maior.

5.2 INTERFACE E MONITORAMENTO

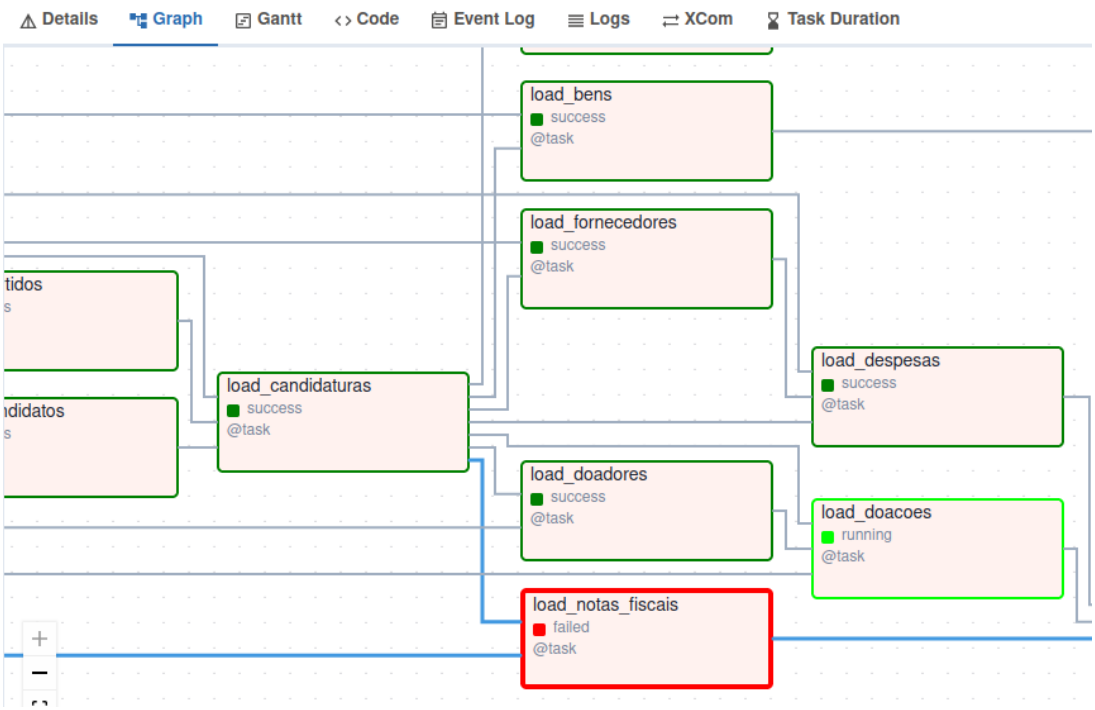
Além da experiência de desenvolvimento, a interface de usuário (UI) e as funcionalidades de observabilidade são cruciais para a operação diária, depuração e monitoramento dos pipelines. A UI funciona como o principal ponto de interação para gerenciar e visualizar os fluxos de dados. Logo, exploraremos como os princípios de arquitetura de cada ferramenta se manifestam em suas interfaces, impactando diretamente a usabilidade e a profundidade do monitoramento oferecido.

A primeira diferença fundamental reside no paradigma visual de cada ferramenta. A interface do Airflow é centrada nas DAGs, onde a tela inicial apresenta uma lista de

todos os pipelines, e o foco do usuário é direcionado para o gerenciamento de suas execuções. Em contrapartida, a UI do Dagster é projetada em torno do grafo global de assets, oferecendo uma visão completa e imediata da saúde e do estado dos dados da aplicação, como ilustrado nas Figuras 12 e 14.

Essa distinção de paradigma se torna mais evidente durante a depuração de falhas. No Airflow, o processo é segmentado e exige vários passos: o desenvolvedor precisa primeiro navegar até a DAG específica, encontrar a execução que falhou na visão de grade (Grid View), clicar na tarefa para, então, acessar seus logs em texto, como detalhado na Figura 16. A reexecução segue uma lógica igualmente operacional, onde o usuário precisa encontrar a tarefa desejada e limpar seu estado (Clear state) para que o scheduler a execute novamente.

Figura 16 – Exemplo de depuração no Airflow: visualizando task com falha

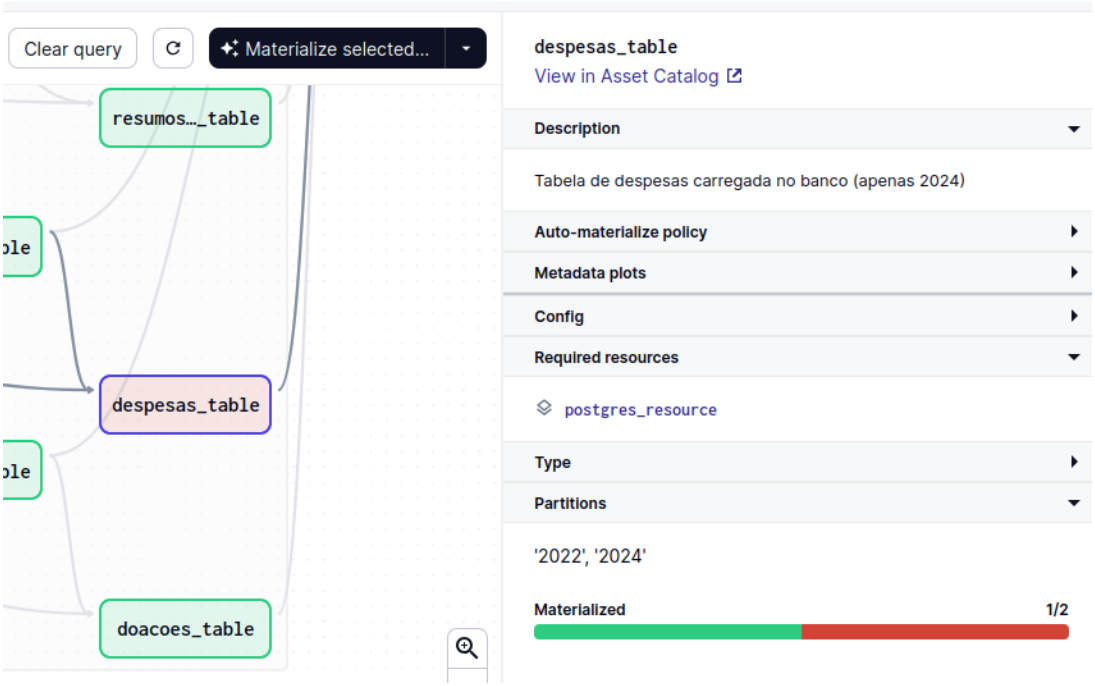


Fonte: Captura de tela da interface do Airflow.

Em contraste, a UI do Dagster, notavelmente mais moderna, oferece um fluxo de trabalho mais integrado e intuitivo para a depuração. A partir do grafo de linhagem global, a identificação de falhas é imediata. Um painel de resumo, como o da Figura 17, expõe o estado de cada partição e permite acionar a rematerialização diretamente. Para uma análise mais profunda, a página de detalhes do ativo centraliza todas as informações de depuração, como pode ser visto na Figura 18. A interface exibe a execução que causou o erro com um link direto para os logs, ao mesmo tempo que preserva o contexto dos metadados da última execução bem-sucedida. A ação de reprocessamento é realizada sobre o *asset*, simplificando drasticamente o ciclo de

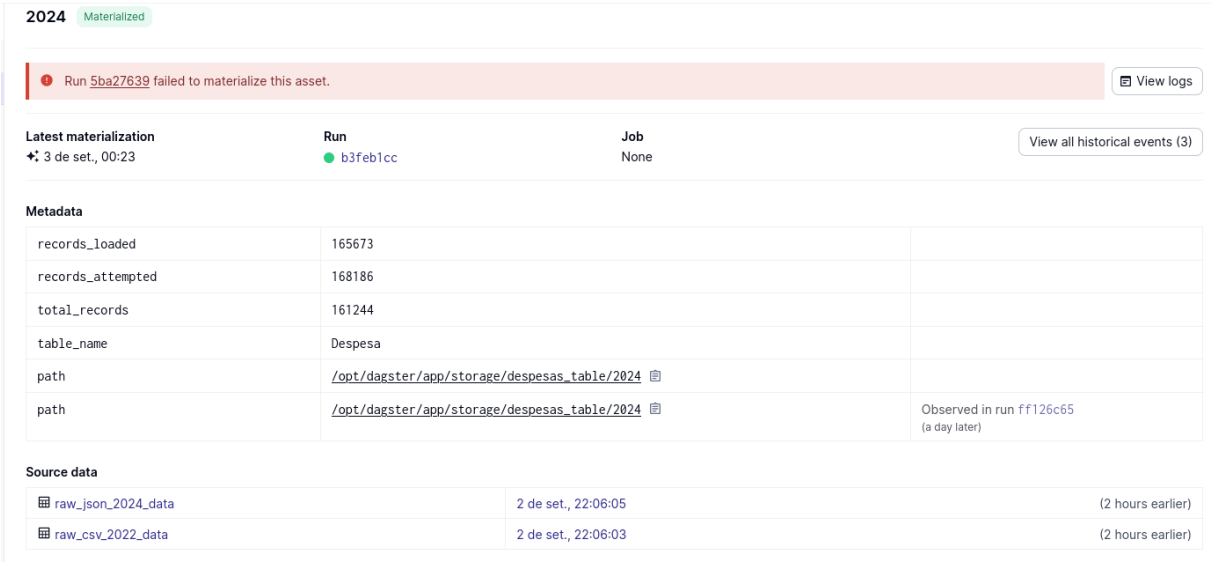
desenvolvimento iterativo e a recuperação de falhas.

Figura 17 – Exemplo de depuração no Dagster: visualizando asset com falha



Fonte: Captura de tela da interface do Dagster.

Figura 18 – Exemplo de depuração no Dagster: detalhes do asset com falha



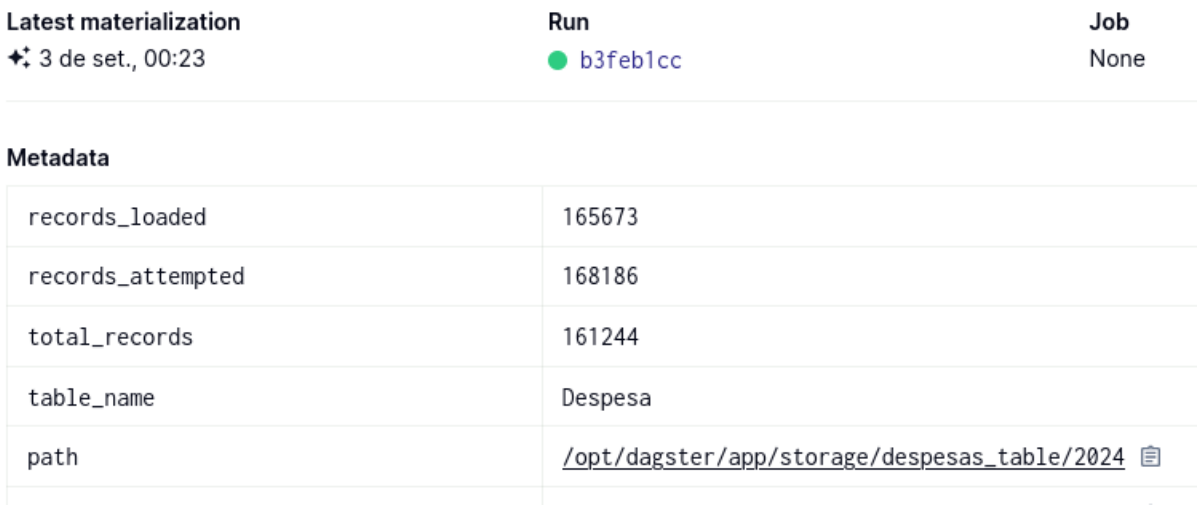
Fonte: Captura de tela da interface do Dagster.

Quando se trata de observabilidade, ambas as ferramentas oferecem mecanismos para inspecionar os resultados, mas com diferentes níveis de profundidade. No Airflow, a observabilidade é centrada no estado operacional da tarefa. Seus principais recursos, como logs detalhados e alertas de falha, são projetados para validar a integridade do processo de execução, informando se uma tarefa foi concluída com ou sem

erros. O monitoramento, portanto, está focado na saúde do pipeline do ponto de vista da execução.

O Dagster, por sua vez, amplia o conceito de observabilidade ao incluir não apenas o status da execução, mas também a qualidade dos dados produzidos. Sua principal vantagem é o suporte nativo e estruturado a metadados. Cada asset pode registrar metadados estruturados durante sua materialização, como por exemplo o número de registros processados, estatísticas descritivas ou resultados de testes de qualidade. Esses metadados são exibidos de forma organizada na UI, associados a cada versão do ativo, como mostra a Figura 19. Isso permite que o usuário monitore não apenas a ocorrência de falhas na execução, mas também se os dados produzidos atendem às expectativas de qualidade. Desse modo, a plataforma permite validar tanto a integridade do processo quanto a validade semântica do resultado. Essa abordagem consciente dos dados fornece um monitoramento muito mais rico e contextualizado do que a simples análise de logs de texto.

Figura 19 – Visualização de metadados de uma materialização de ativo no Dagster



Fonte: Captura de tela da interface do Dagster.

Logo, a experiência de operação e monitoramento em cada plataforma é bastante distinta. O Airflow provê as ferramentas necessárias para um monitoramento técnico e focado na execução, exigindo do usuário uma navegação mais segmentada para depuração. Em contrapartida, o Dagster entrega uma experiência mais moderna e unificada, onde a centralidade nos ativos de dados e seus metadados resulta em um fluxo de trabalho mais intuitivo para identificar, entender e resolver problemas, diminuindo a distância entre a falha operacional e a análise de seu impacto nos dados.

5.3 COMPARAÇÃO DE DESEMPENHO

Para além da análise qualitativa sobre a experiência de desenvolvimento e operação, foi realizada uma análise quantitativa com o objetivo de comparar o desempenho do Airflow e do Dagster na execução dos pipelines de ETL propostos. Esta seção detalha a metodologia de aferição utilizada, apresenta os resultados de tempo de execução e consumo de recursos, e discute como as características arquitetônicas de cada ferramenta impactaram esses resultados.

Para garantir uma comparação de desempenho justa e reprodutível, foi estabelecido um ambiente de teste padronizado. Todas as execuções ocorreram na mesma configuração de hardware e utilizaram a arquitetura containerizada com Docker detalhada anteriormente, assegurando que quaisquer variações de desempenho fossem atribuíveis unicamente às ferramentas. Adicionalmente, cada pipeline foi executado a partir de um estado limpo, com o banco de dados da aplicação vazio, de modo a medir o desempenho de uma carga completa e evitar a influência de dados pré-existentes ou caches.

As métricas de tempo de execução foram extraídas diretamente das interfaces de usuário de cada ferramenta, enquanto o consumo de recursos (CPU e Memória) foi monitorado durante os picos de processamento por meio do comando `docker stats`.

5.3.1 Tempo de Execução

O principal indicador de desempenho avaliado foi o tempo total necessário para a execução completa de cada pipeline, desde a extração inicial dos dados até a carga final no banco de dados. A Tabela 3 resume os resultados obtidos.

Tabela 3 – Comparativo do tempo de execução total dos pipelines (HH:MM:SS).

Pipeline	**Airflow**	**Dagster**
Eleições 2022	**00:03:32**	00:05:37
Eleições 2024	**01:16:56**	02:42:50

Os resultados obtidos revelam um cenário interessante. Primeiramente, é fundamental contextualizar que o salto de minutos em 2022 para horas em 2024 não é uma surpresa, dado que o conjunto de dados das eleições de 2024 é substancialmente maior em volume (cerca de 6900 registros para 2022 e aproximadamente 360000 em 2024) e variedade, incluindo entidades como doadores, fornecedores e um detalhamento financeiro mais completo. Neste contexto, o Airflow demonstrou um desempenho superior em ambos os cenários. No pipeline de 2022, foi aproximadamente 37% mais rápido, mas a diferença se acentua drasticamente no pipeline de 2024, onde o Airflow foi mais de duas vezes mais rápido.

Uma análise da execução de cada etapa dos fluxos revela que o gargalo não estava distribuído uniformemente. Enquanto as etapas de Extração e Transformação tiveram desempenhos comparáveis e muito eficientes (menos de 30 segundos em ambas as ferramentas), a principal discrepância ocorreu na etapa de Carga. Especificamente, a lentidão se concentrou na tabela de *Despesas*, a maior do conjunto de dados, onde foram carregados 144218 itens. A tarefa de carga desta tabela no Airflow foi concluída em aproximadamente 52 minutos, enquanto a materialização do *asset* equivalente no Dagster levou cerca de 2 horas, respondendo pela maior parte da diferença de tempo total.

Uma hipótese para essa diferença de velocidade está na forma como o *PostgresHook* do Airflow e o *Resource* do Dagster gerenciam a conexão com o banco de dados. A abordagem do Airflow parece manter a conexão aberta por mais tempo, o que é mais rápido para inserir um grande volume de dados de uma só vez. Já o Dagster prioriza a organização e segurança do código, o que pode adicionar passos extras a cada operação de escrita. Em uma tarefa tão massiva, essa sobrecarga, mesmo que pequena a cada passo, pode ter se acumulado e tornado o processo mais lento.

Outra hipótese considerada para a performance superior do Airflow reside na própria arquitetura de serviços utilizada por padrão. Enquanto o setup do Dagster é mais centralizado, o ambiente do Airflow foi instanciado com uma infraestrutura distribuída que inclui o Redis atuando como *message broker*. Embora essa arquitetura adicione complexidade e eleve o consumo de recursos (como será detalhado na Seção 5.3.2), ela oferece um mecanismo de fila altamente otimizado para o agendamento e execução de tarefas. A presença do Redis permite que a comunicação entre o *scheduler* e os *workers* ocorra com latência mínima e alto *throughput*, garantindo que a lógica de orquestração não se torne um gargalo, mesmo diante do alto volume de dados processados no cenário de 2024.

5.3.2 Consumo de Recursos (CPU e Memória)

Para aprofundar a análise de desempenho, vamos investigar o consumo de CPU e memória, conectando os tempos de execução observados com a eficiência de cada arquitetura. A análise é dividida em dois momentos distintos, onde primeiro o consumo em repouso é avaliado para quantificar o *overhead* inerente de cada sistema e, em seguida, o comportamento durante a execução de pico é detalhado para identificar os gargalos e as dinâmicas de uso de recursos sob carga máxima.

Consumo em repouso

A análise do consumo de recursos em repouso, realizada com os pipelines inativos, vai de encontro com a hipótese de que a arquitetura do Airflow possui um consumo de recursos inerentemente maior por conta do maior número de serviços

no seu container. A Tabela 4 resume o consumo de memória RAM de cada sistema, calculado a partir da soma dos seus respectivos containers em execução.

Tabela 4 – Comparativo de consumo de memória em repouso.

Ferramenta	Consumo Total (Aprox.)	Componentes de Maior Consumo
Dagster	870 MiB	daemon (434 MiB), webserver (259 MiB)
Airflow	**3.44 GiB**	webserver (**1.33 GiB**), worker (**1.32 GiB**)

Essa diferença é uma consequência direta das distintas abordagens de arquitetura. No Airflow, a arquitetura distribuída por natureza, composta por múltiplos serviços especializados, resulta em um consumo de base elevado. Notavelmente, os componentes webserver e worker são os maiores consumidores, cada um utilizando mais de 1.3 GiB de RAM. Provavelmente, isso ocorre porque ambos os serviços precisam carregar as definições de todos os DAGs na memória para operar, sendo o webserver para exibir a interface e o worker para estar pronto para executar tarefas. A presença de outros serviços, como o Redis, também contribui para o overhead total.

Em contrapartida, a arquitetura do Dagster é mais enxuta e integrada. Embora o daemon seja o principal consumidor de memória, provavelmente por pré-carregar as definições de código para monitorar execuções, o consumo total do sistema permanece abaixo de 1 GiB. Isso reflete uma arquitetura que, para cenários locais, é otimizada para ser mais leve e iniciar mais rapidamente, validando a percepção de que sua configuração inicial é mais simples e menos exigente em termos de recursos.

Consumo em execução

Para compreender as causas por trás das diferenças nos tempos de execução, foi realizada uma análise detalhada do consumo de recursos durante a fase de maior estresse de cada pipeline: a etapa de Carga, no momento em que o maior número de tarefas ou *assets* estavam sendo executados em paralelo. A arquitetura de contêineres utilizada, com bancos de dados dedicados para metadados e armazenamento (dos dados de candidatos) em ambos os sistemas, permitiu um diagnóstico preciso dos gargalos de desempenho. A Tabela 5 consolida os dados de pico coletados.

A análise dos dados de pico revela um padrão de comportamento consistente e distinto para cada ferramenta. No Airflow, observa-se uma carga de trabalho integrada e bem distribuída. Durante a execução, tanto o *Executor* quanto os bancos de dados da *Aplicação* e de *Metadados* operam em alta performance de forma equilibrada. Isso demonstra uma arquitetura eficiente, onde os componentes responsáveis pela execução e pela persistência dos dados trabalham em harmonia, um padrão que se manteve e escalou de forma coesa mesmo sob a carga massiva do pipeline de 2024.

Tabela 5 – Comparativo de consumo de recursos de pico durante a etapa de Carga.

Pipeline	**Componente**	**Airflow**		**Dagster**	
		CPU %	**Memória**	**CPU %**	**Memória**
Eleições 2022	Executor (*Worker*/*Webserver*)	75.11%	**1.76 GiB**	**156.81%**	1.45 GiB
	BD de Candidatos	**92.01%**	31 MiB	38.06%	**93 MiB**
	BD de Metadados	85.45%	74 MiB	**115.78%**	**85 MiB**
Eleições 2024	Executor (*Worker*/*Webserver*)	304.84%	**4.44 GiB**	**383.11%**	2.44 GiB
	BD de Candidatos	**130.19%**	**47 MiB**	75.64%	45 MiB
	BD de Metadados	150.04%	**222 MiB**	**240.44%**	72 MiB

Em contrapartida, o perfil de execução do Dagster expõe um gargalo claramente deslocado para a sua lógica de orquestração interna. O *Executor* e o *BD de Metadados* tornam-se consistentemente sobrecarregados, absorvendo a vasta maioria dos recursos de CPU. Simultaneamente, o *BD de Candidatos* opera com uma fração da carga de seu equivalente no Airflow. Essa dinâmica, que se intensificou drasticamente no pipeline de 2024, evidencia que o custo de desempenho não está na interação com o banco de dados da aplicação.

Os dados coletados sugerem fortemente que o gargalo de desempenho do Dagster não reside em sua capacidade de interagir com o PostgreSQL para a carga de dados, mas sim em um imenso *overhead* imposto por sua própria lógica de orquestração. Uma interpretação plausível é que o processo de materializar *assets* em paralelo gera uma cascata de eventos, registros de linhagem e atualizações que sobrecarrega o executor e o banco de metadados. Este comportamento leva à subutilização do banco de dados da aplicação, oferecendo uma explicação consistente para os maiores tempos de execução observados. A análise, portanto, evidencia um *trade-off* fundamental: a boa observabilidade e o paradigma orientado a *assets* do Dagster impõem uma sobrecarga substancial, enquanto a arquitetura do Airflow, embora consuma mais recursos devido a serviços auxiliares como o Redis, consegue converter essa infraestrutura adicional em uma execução de tarefas significativamente mais eficiente e escalável.

6 ANÁLISE EXPLORATÓRIA DOS DADOS COLETADOS

Este capítulo apresenta uma análise exploratória dos dados eleitorais coletados através dos pipelines de ETL implementados usando as ferramentas Apache Airflow e Dagster. O objetivo é compreender as características gerais dos dados, identificar padrões relevantes e fornecer uma visão quantitativa do cenário eleitoral em Santa Catarina nas eleições de 2022 e 2024.

Os dados coletados abrangem períodos distintos conforme o tipo de informação. As transações financeiras de doações foram registradas entre 10 de fevereiro de 2024 e 28 de dezembro de 2024, totalizando 322 dias. As despesas de campanha cobrem o período de 1º de abril de 2024 a 16 de novembro de 2024, resultando em 229 dias. Já as notas fiscais possuem registros de 12 de agosto de 2022 a 10 de novembro de 2024, abrangendo 821 dias.

É importante ressaltar que os registros de doações e despesas estão disponíveis apenas para a eleição de 2024. Para a eleição de 2022, apenas as notas fiscais foram coletadas. Esta assimetria nos dados deve ser considerada na interpretação dos resultados financeiros, restringindo análises comparativas diretas entre os dois pleitos no aspecto de receitas e despesas.

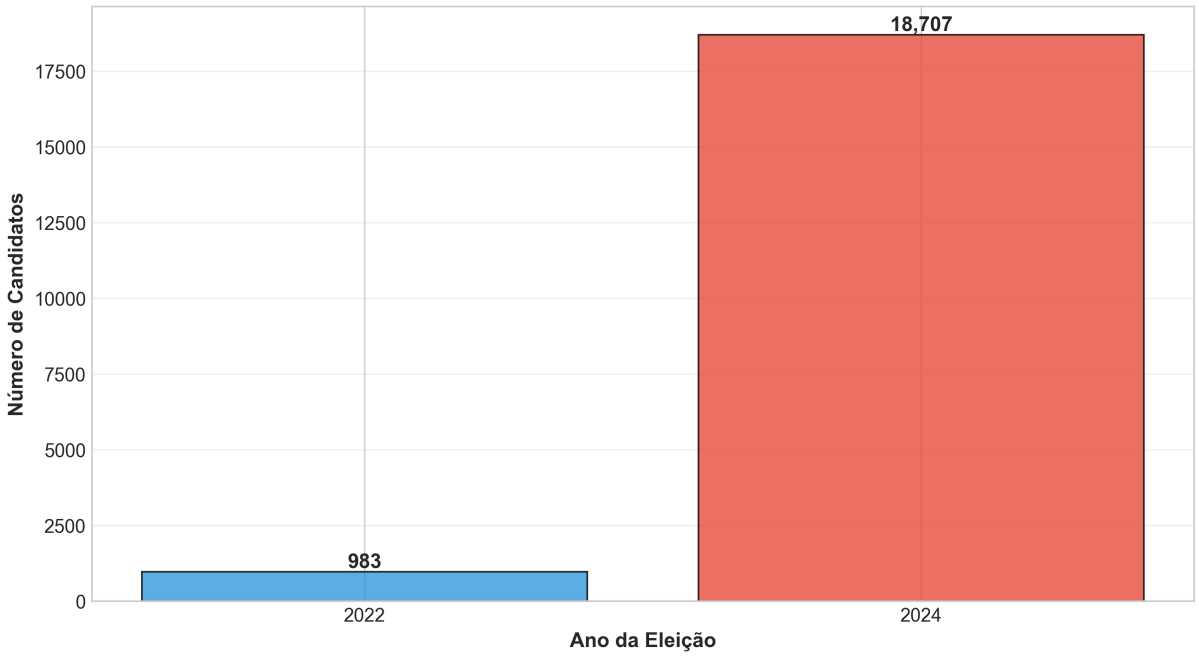
A análise foi conduzida utilizando consultas *SQL* sobre o banco de dados e scripts Python com as bibliotecas Pandas, Matplotlib e Seaborn para processamento e visualização dos dados. Os resultados aqui apresentados não apenas caracterizam o conjunto de dados coletado, mas também demonstram a eficácia dos workflows de ETL desenvolvidos em ambas as ferramentas.

6.1 PERFIL DAS CANDIDATURAS

6.1.1 Candidatos

Ao todo, foram coletados e integrados dados de 19.339 candidatos únicos, sendo 983 candidatos na eleição de 2022 e 18.707 candidatos na eleição de 2024. A diferença significativa no volume de candidaturas entre as duas eleições reflete a natureza descentralizada das eleições municipais, que, por possuir uma distribuição geográfica mais ampla, mobilizam um número muito maior de candidatos. A Figura 20 ilustra essa diferença de grande de candidatos entre os dois períodos.

Figura 20 – Número de candidatos por eleição em Santa Catarina



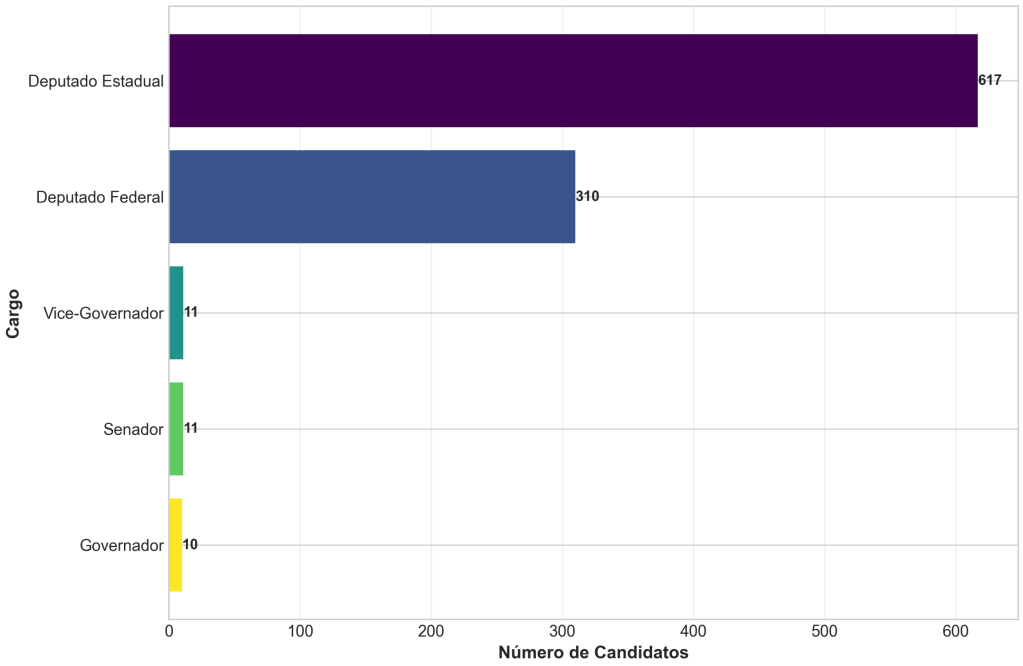
Fonte: O autor.

6.1.2 Distribuição por Cargo

A distribuição de candidatos por cargo reflete as características estruturais de cada tipo de eleição. Na eleição de 2022, os cargos legislativos concentraram a maior parte das candidaturas, onde 617 candidatos concorreram ao cargo de Deputado Estadual (62,8% do total) e 310 ao cargo de Deputado Federal (31,5%). Os cargos executivos e para o Senado apresentaram menor volume, com 10 candidatos a Governador, 11 a Senador, 11 a Vice-Governador, como ilustra a Figura 21

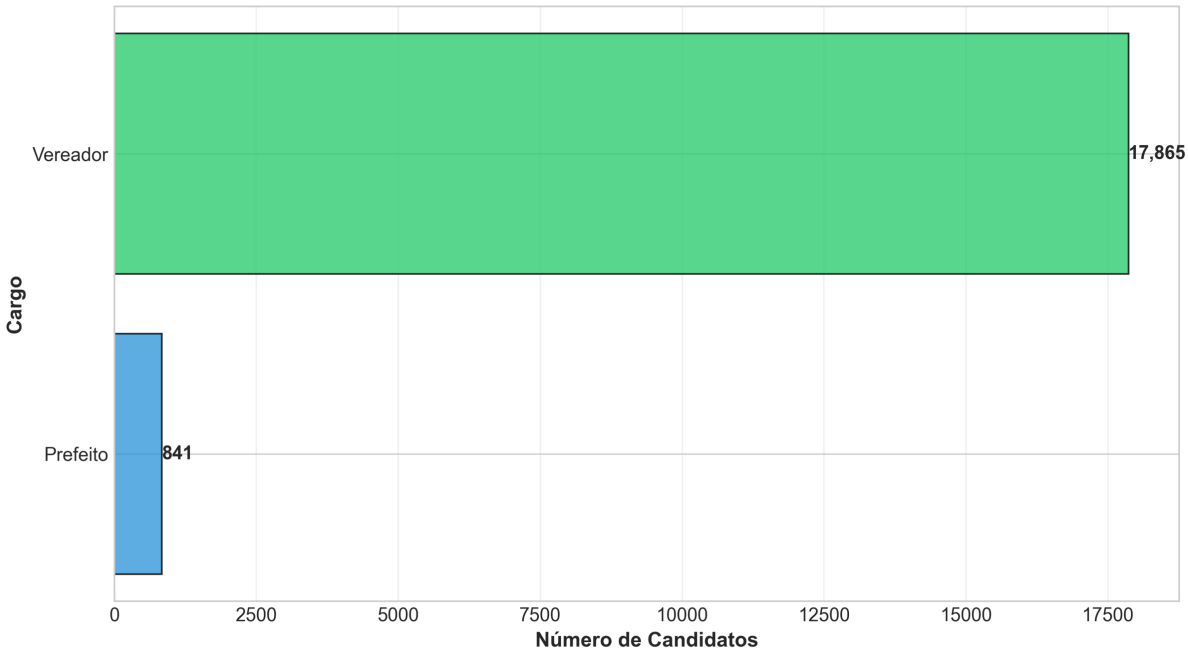
Na eleição municipal de 2024, a distribuição foi ainda mais concentrada, onde tivemos 17.865 candidatos que disputaram vagas para Vereador, representando 95,5% de todas as candidaturas, enquanto 841 candidatos (4,5%) concorreram ao cargo de Prefeito. Esta distribuição, ilustrada na Figura 22, evidencia a alta competitividade das eleições para o legislativo municipal, que representa a principal porta de entrada na política representativa para muitos candidatos.

Figura 21 – Distribuição de candidatos por cargo nas eleições de 2022



Fonte: O autor.

Figura 22 – Distribuição de candidatos por cargo nas eleições de 2024



Fonte: O autor.

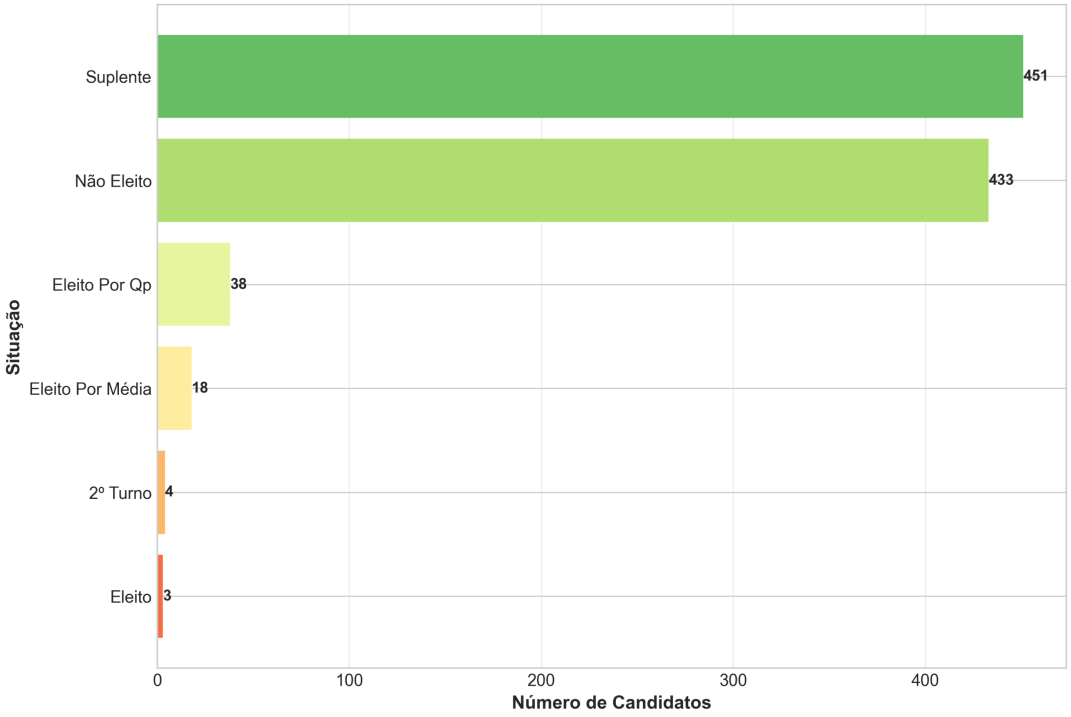
6.1.3 Representatividade de Gênero

A análise da representatividade de gênero nas candidaturas revela um padrão consistente entre as duas eleições analisadas. Na eleição de 2022, foram registradas 654 candidaturas masculinas (66,5%) e 329 femininas (33,5%). Em 2024, observou-se uma proporção semelhante, sendo 12.118 candidatos do sexo masculino (64,8%) e 6.589 do sexo feminino (35,2%).

6.1.4 Situação Final das Candidaturas

A análise da situação final dos candidatos após o processo eleitoral oferece insights sobre a competitividade e as dinâmicas de cada eleição. Em 2022, 451 candidatos (45,9%) foram classificados como suplentes, 433 (44,0%) não foram eleitos, 38 (3,9%) foram eleitos por quociente partidário (QP), 18 (1,8%) foram eleitos por média, 4 candidatos participaram de segundo turno, e apenas 3 foram eleitos diretamente no primeiro turno para cargos majoritários. A Figura 23 ilustra esses dados.

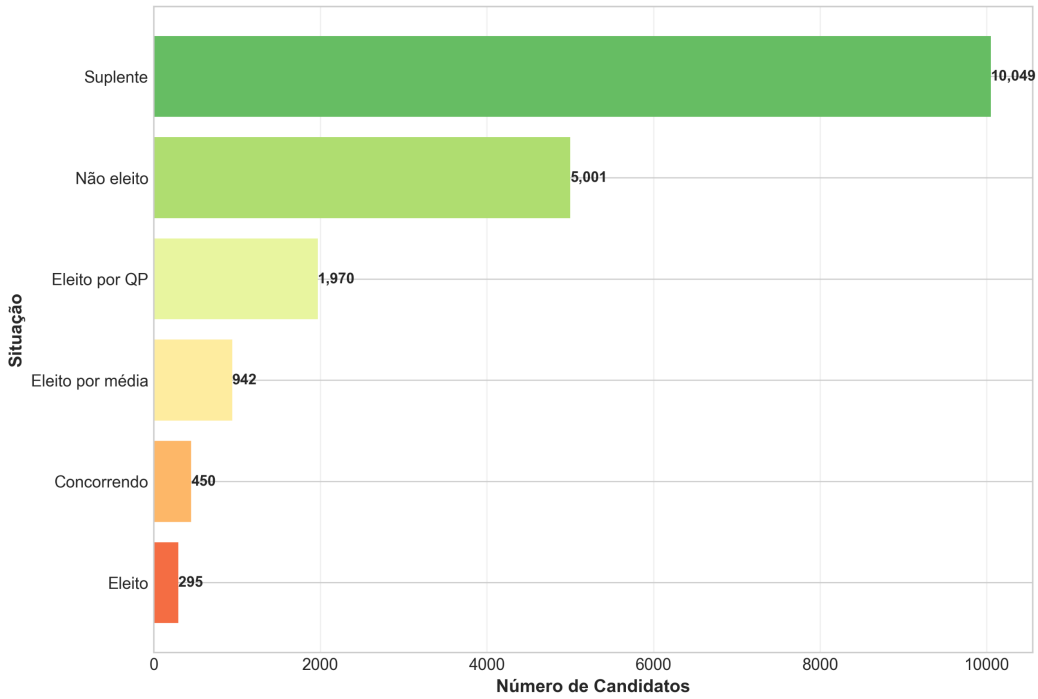
Figura 23 – Situação Final das Candidaturas nas eleições de 2022



Fonte: O autor.

Em 2024, a distribuição foi distinta, onde 10.049 candidatos (53,7%) ficaram como suplentes, 5.001 (26,7%) não foram eleitos, 1.970 (10,5%) foram eleitos por quociente partidário, 942 (5,0%) foram eleitos por média e 295 (1,6%) foram eleitos diretamente. A Figura 24 ilustra esse cenário.

Figura 24 – Situação Final das Candidaturas nas eleições de 2024



Fonte: O autor.

A taxa efetiva de sucesso eleitoral, considerando todas as formas de eleição (direta, por QP e por média), foi de aproximadamente 6,0% em 2022 e 17,1% em 2024, refletindo a maior competitividade das eleições estaduais e federais em comparação com as municipais, principalmente por decidir cargos mais exclusivos e com menos representantes, como governador e senadores, por exemplo.

6.2 ANÁLISE FINANCEIRA

A análise financeira das campanhas eleitorais foi conduzida com base nos dados de doações, despesas e notas fiscais registradas pelos candidatos. Conforme mencionado anteriormente, os dados completos de receitas e despesas estão disponíveis apenas para a eleição de 2024, o que limita comparações diretas com 2022.

6.2.1 Receitas de Campanha

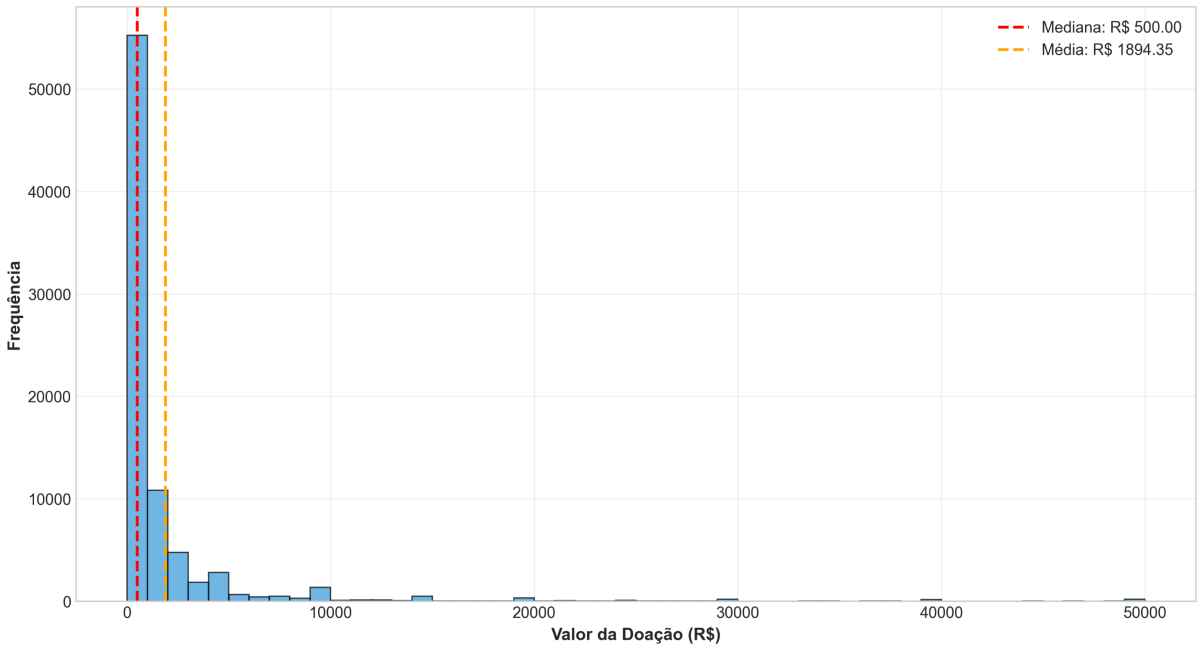
Na eleição de 2024, foram registradas 81.678 doações, totalizando R$ 259.616.828,47 arrecadados pelos candidatos. O valor médio por doação foi de R$ 3.178,54, com transações variando de R$ 0,01 até o limite máximo observado de R$ 4.000.000,00. Esta variação expressiva evidencia a heterogeneidade no perfil dos doadores e no porte das doações recebidas.

A base de doadores identificada compreende 23.257 pessoas físicas e jurídicas únicas. Em média, cada doador contribuiu com aproximadamente R$ 11.164,68,

embora esta distribuição seja altamente assimétrica, com poucos doadores realizando contribuições significativas enquanto a maioria contribui com valores menores. O valor mínimo de R$ 0,01 observado em algumas transações pode indicar ajustes contábeis ou doações de valor simbólico.

A análise da distribuição de valores de doações, apresentada na Figura 25, revela uma concentração expressiva em valores baixos e médios, com uma cauda longa indicando doações de grande porte. Vemos que a maioria das transações envolve valores moderados, mas algumas poucas doações de alto valor representam parcela significativa do montante total arrecadado.

Figura 25 – Distribuição dos valores de doações na eleição de 2024 (Limitado a R$ 50.000 para melhor visualização)



Fonte: O autor.

6.2.2 Despesas de Campanha

No âmbito das despesas, foram contabilizadas 144.218 transações em 2024, totalizando R$ 225.323.593,31 gastos pelos candidatos. O valor médio por despesa foi de R$ 1.562,38, consideravelmente inferior à média das doações, com gastos variando de R$ 0,01 a R$ 1.952.000,00.

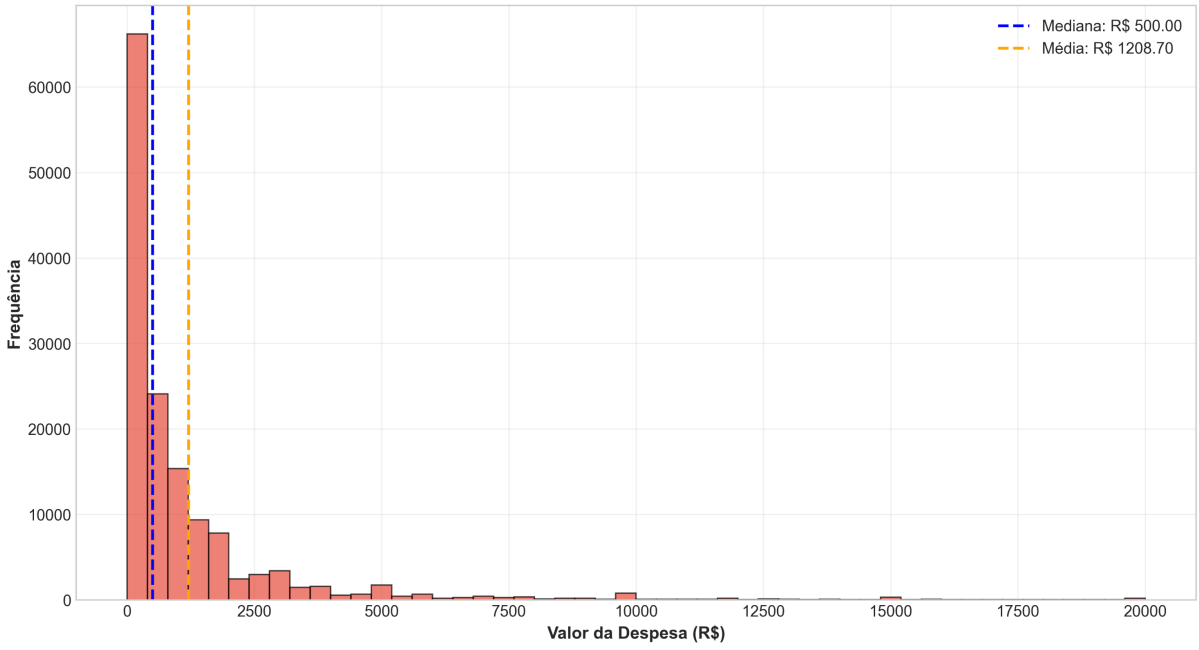
Um aspecto notável é que o número de transações de despesas é significativamente maior que o de doações, indicando que, em média, as despesas são mais diluídas e de menor valor em comparação com as doações. A razão entre o número de despesas e receitas é de aproximadamente 1,77:1, demonstrando que para cada doação recebida, são realizadas quase duas transações de despesa. Este padrão

sugere uma estratégia de campanha baseada em múltiplos pequenos gastos, possivelmente refletindo a natureza descentralizada das campanhas municipais, com despesas distribuídas entre diversos fornecedores locais.

O ecossistema de prestadores de serviços para campanhas eleitorais é amplo, sendo que foram identificados 42.797 fornecedores únicos. Esta diversidade de fornecedores evidencia a complexidade logística e operacional das campanhas eleitorais, que demandam uma variedade de serviços como gráficas, transportes, alimentação, comunicação, marketing, entre outros. Em média, cada fornecedor recebeu R$ 5.263,18, valor significativamente inferior a média de contribuição por doador, indicando que os candidatos ganham mais do que gastam na candidatura.

A distribuição de valores de despesas, ilustrada na Figura 26, apresenta características semelhantes à das doações, com concentração em valores baixos e médios e presença de outliers de alto valor.

Figura 26 – Distribuição dos valores de despesas na eleição de 2024 (Limitado a R$ 20.000 para melhor visualização)



Fonte: O autor.

6.2.3 Notas Fiscais

Complementando os dados de receitas e despesas, foram analisadas 57.530 notas fiscais emitidas durante as campanhas. Destas, 8.964 referem-se à eleição de 2022, totalizando R$ 98.491.197,94, com valor médio de R$ 10.987,42 por nota, e 48.566 à eleição de 2024, totalizando R$ 97.487.061,21, com valor médio de R$ 2.007,35.

É interessante notar que, embora a eleição de 2024 tenha gerado um número muito maior de notas fiscais, o valor total foi praticamente equivalente ao de 2022. Esta diferença no valor médio das notas sugere que as campanhas estaduais e federais tendem a realizar transações de maior porte, enquanto as campanhas municipais trabalham com valores menores e mais fragmentados.

6.3 PATRIMÔNIO DECLARADO

A análise dos bens declarados pelos candidatos oferece uma visão sobre o perfil patrimonial dos candidatos. Ao todo, foram declarados 58.144 bens, somando um patrimônio total de aproximadamente R$ 7,48 bilhões.

A distribuição entre as eleições apresenta diferenças significativas. Em 2022, foram declarados 4.301 bens, totalizando R$ 1,19 bilhão, com valor médio de R$ 276.333,39 por bem. Em 2024, foram declarados 53.843 bens, totalizando R$ 6,30 bilhões, com valor médio de R$ 116.921,57 por bem.

O patrimônio médio por candidato, considerando o total declarado dividido pelo número de candidatos únicos, é de aproximadamente R$ 386.938,81. A Tabela 6 apresenta um resumo comparativo do patrimônio declarado nas duas eleições.

Tabela 6 – Resumo do patrimônio declarado pelos candidatos

Eleição	**Bens Declarados**	**Valor Total (R$)**	**Valor Médio (R$)**
2022	4.301	1.188.510.000,00	276.333,39
2024	53.843	6.295.408.000,00	116.921,57
Total	58.144	7.483.918.000,00	–

Fonte: O autor.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo final tem como objetivo consolidar os resultados e aprendizados obtidos ao longo deste trabalho. A primeira seção apresenta as conclusões extraídas do estudo comparativo entre as ferramentas Apache Airflow e Dagster, com especial atenção às descobertas da análise prática detalhada no Capítulo 5. Serão destacados os pontos fortes e fracos de cada ferramenta, os cenários de aplicação em que cada uma se sobressai, as diferenças observadas entre a fundamentação teórica e a implementação empírica e as dificuldades enfrentadas no desenvolvimento do trabalho. Adicionalmente, será apresentada uma reflexão sobre a experiência de utilizar Inteligência Artificial generativa como assistente no processo de desenvolvimento, destacando tanto sua utilidade como acelerador quanto as limitações observadas. Por fim, a última seção é dedicada a explorar possíveis trabalhos futuros.

7.1 CONCLUSÕES

O objetivo central deste trabalho foi realizar uma comparação empírica entre o Apache Airflow e o Dagster, duas das mais proeminentes ferramentas de orquestração de dados. A implementação de pipelines de ETL para coleta e processamento de dados eleitorais de 2022 e 2024 permitiu uma avaliação aprofundada que transcendeu a teoria, revelando nuances práticas em experiência de desenvolvimento, monitoramento e desempenho. A análise comparativa confirmou e, em alguns aspectos, desafiou as percepções teóricas discutidas no Capítulo 2, provando ser uma etapa fundamental para uma compreensão assertiva das ferramentas.

Apache Airflow

A análise prática reforçou a posição do Airflow como uma ferramenta robusta, madura e altamente performática. Seu principal destaque foi o desempenho superior na carga de dados, sendo mais de duas vezes mais rápido que o Dagster no cenário mais intensivo, o que demonstra uma arquitetura de execução otimizada para operações em lote. A maturidade da ferramenta também se reflete em seu paradigma centrado em *Tasks* e *DAGs*, que, por ser um padrão de mercado, pode facilitar a adoção por equipes já familiarizadas com conceitos tradicionais de orquestração.

Contudo, essa maturidade vem acompanhada de uma maior complexidade inicial e overhead de infraestrutura. O setup local, com múltiplos contêineres e alto consumo de memória em repouso, impõe uma curva de aprendizado mais íngreme. A implementação prática também revelou uma menor flexibilidade para pipelines parametrizados, exigindo a duplicação de código e da lógica de orquestração para cenários similares. Adicionalmente, sua interface de usuário, embora funcional, mostrou-se

menos integrada para depuração. Por essas características, o Airflow se destaca em ambientes de produção de grande escala, onde o desempenho bruto e a confiabilidade são críticos.

Dagster

Por outro lado, o Dagster provou ser uma ferramenta moderna, com um forte foco na experiência de desenvolvimento e na governança de dados. Seu grande diferencial é a abordagem centrada em *assets*, que proporciona uma linhagem de dados clara e uma observabilidade superior. A capacidade nativa de registrar e visualizar metadados estruturados para cada materialização é um recurso poderoso para monitorar a qualidade dos dados, e não apenas o status da execução. A experiência de desenvolvimento foi notavelmente mais fluida, beneficiada por um setup local simplificado, um ciclo de feedback rápido e uma excelente reutilização de código através de conceitos como *Partitions* e *Resources*. Sua interface de usuário, mais moderna e intuitiva, facilita drasticamente a depuração e o monitoramento.

Essa sofisticação, no entanto, apresentou contrapartidas diretas em performance, sendo que o principal ponto fraco identificado foi o desempenho em cargas de dados massivas, onde a camada de abstração do Dagster impôs um overhead significativo que sobrecarregou seu executor e banco de metadados, resultando em tempos de execução consideravelmente mais longos. Dessa forma, o Dagster é ideal para projetos onde a qualidade, a linhagem e a observabilidade dos dados são prioritárias em relação à velocidade de execução. É uma excelente escolha para equipes que valorizam a clareza sobre como os ativos são produzidos e consumidos é fundamental para a governança.

Análise Teórica vs. Prática

A aplicação empírica deste trabalho foi fundamental para confrontar o conhecimento teórico da literatura com a realidade da implementação. A análise teórica, revisada no Capítulo 2, já indicava corretamente a maturidade do Airflow e a abordagem intuitiva e baseada em ativos do Dagster. A prática confirmou que a experiência de desenvolvimento no Dagster é de fato mais fluida, e que a complexidade de infraestrutura do Airflow é maior.

Contudo, a análise prática revelou nuances críticas que a teoria não captura. A primeira foi a complexidade conceitual: enquanto a literatura aponta o Dagster como mais simples, a análise deste trabalho mostrou que ele possui, na verdade, um número maior de abstrações. Sua aparente simplicidade deriva do fato da ferramenta possuir diversas soluções nativas que solucionam complexidades comuns da engenharia de dados. Conforme mapeado com o WfRM, o Airflow implementa os conceitos clássicos de workflow de forma mais direta, mas deixa a cargo do desenvolvedor a solução para

problemas como a parametrização, o que levou à duplicação de código. O Dagster, por sua vez, estende esses conceitos com abstrações como *Partitions*, resolvendo o problema de forma nativa.

A descoberta mais contundente, no entanto, foi sobre o desempenho. A literatura apenas sugere que o Dagster poderia ser menos escalável, mas a análise de consumo de recursos deste trabalho diagnosticou um severo overhead na sua própria lógica de orquestração. A prática demonstrou que o gargalo não estava na interação com o banco de dados da aplicação, mas sim na sobrecarga do executor e do banco de metadados do Dagster, responsáveis por gerenciar a linhagem e os eventos de materialização. Esse custo de desempenho, inerente à sua rica camada de observabilidade, é um trade-off fundamental que apenas a análise prática pôde quantificar e explicar. Assim, este trabalho prova que a implementação empírica é indispensável para ir além da teoria das ferramentas e revelar seus comportamentos concretos, informando de maneira assertiva a decisão técnica.

Dificuldades enfrentadas

Embora o objetivo central deste trabalho fosse a comparação entre as ferramentas de orquestração Airflow e Dagster, uma parcela significativa da complexidade do projeto residiu na etapa de engenharia de dados, especialmente no desenvolvimento do web scraper para as eleições de 2024. Diferentemente dos dados de 2022, disponibilizados em arquivos CSV estruturados, as informações de 2024 exigiram a construção de uma solução personalizada para extrair dados do sistema DivulgaCandContas, processo que demandou um tempo considerável de análise para definir a estrutura dos objetos JSON, filtrar campos relevantes das páginas web dos candidatos e definir estratégias eficientes de navegação e captura que garantissem a completude da extração.

Adicionalmente, a heterogeneidade das fontes impôs desafios na padronização e integração, exigindo um esforço de modelagem para mapear a semântica de atributos distintos em um esquema de banco de dados relacional unificado e coerente. Esse processo foi complementado por uma etapa crítica de validação da qualidade dos dados persistidos, que envolveu verificações rigorosas para assegurar a consistência do armazenamento. Tais medidas foram fundamentais para prevenir a perda de dados relevantes, evitar duplicações indevidas e garantir a idempotência dos pipelines desenvolvidos.

7.2 USO DE IA NO DESENVOLVIMENTO DOS PIPELINES

Para o desenvolvimento dos pipelines de ETL deste trabalho, foi utilizada Inteligência Artificial generativa por meio do editor de código Cursor, uma experiência que se revelou uma estratégia extremamente proveitosa e que acelerou significamente o processo de implementação. A ferramenta demonstrou grande utilidade na geração de códigos de base, tanto para a estrutura de DAGs em Airflow quanto para a definição de assets em Dagster, o que permitiu superar a inércia inicial de configuração de cada projeto. Além disso, o Cursor atuou como uma ferramenta de consulta ágil, ajudando a esclarecer dúvidas conceituais sobre as particularidades de cada framework e a resolver rapidamente problemas pontuais de codificação, otimizando o tempo e a produtividade.

Por outro lado, a experiência também evidenciou de forma clara as limitações atuais da tecnologia. Ao longo do processo, a IA integrada ao Cursor cometeu diversos erros e apresentou alucinações, sugerindo soluções funcionalmente incorretas, desnecessárias para o contexto ou que não aderiam às melhores práticas de cada framework. Em alguns casos, as respostas geradas introduziam complexidade excessiva, duplicação de lógica ou padrões de projeto inadequados, o que exigiu retrabalho e refinamento manual. A principal deficiência observada foi a incapacidade da IA de raciocinar sobre o projeto como um todo, oferecendo trechos de código aparentemente otimizados para um problema isolado, mas sem considerar as consequências indiretas, a consistência arquitetural ou a manutenibilidade futura da solução integrada. Nesse sentido, o uso da ferramenta implicou também custos adicionais de validação, revisão e depuração, tornando indispensável um esforço humano constante de supervisão e discernimento para garantir a qualidade e a pertinência das sugestões. O papel do desenvolvedor permaneceu, portanto, central tanto para assegurar a coesão da arquitetura quanto para pensar estrategicamente sobre o ciclo de vida do projeto.

7.3 TRABALHOS FUTUROS

Este estudo forneceu uma comparação detalhada do Airflow e do Dagster em um contexto de ETL em lote com dados estruturados e semiestruturados. No entanto, o vasto campo da engenharia de dados oferece diversas oportunidades para expandir esta pesquisa. A seguir, são sugeridos alguns caminhos para trabalhos futuros:

1. **Análise de Desempenho em Ambiente Distribuído:** A presente análise foi realizada em um ambiente local containerizado. Um passo natural seria replicar os pipelines em um ambiente de nuvem distribuído, permitindo uma avaliação mais realista da escalabilidade, resiliência a falhas e complexidade operacional de cada ferramenta em um cenário de produção.
2. **Expansão da Base de Dados Histórica:** Uma evolução natural deste trabalho seria a expansão do conjunto de dados para abranger um espectro temporal mais amplo, desenvolvendo novos fluxos de ETL para eleições passadas e futuras. O resultado seria a criação de um repositório histórico unificado e robusto, viabilizando análises sobre financiamento de campanha, evolução patrimonial de candidatos e padrões políticos ao longo do tempo.
3. **Otimização de Desempenho do Dagster:** O gargalo de desempenho identificado no Dagster abre uma frente de investigação, onde trabalhos futuros poderiam explorar diferentes configurações de executores, estratégias de paralelismo e otimizações na interação com o banco de dados para verificar se o overhead de orquestração pode ser reduzido.

REFERÊNCIAS

AIRFLOW, Apache. **Apache Airflow Documentation**. Acesso em: 6 de mai. 2025. Apache Software Foundation. 2025. Disponível em: https://airflow.apache.org/docs/apache-airflow/stable/index.html.

AIRFLOW, Apache. **Apache Airflow User Survey 2022 Results**. Acesso em: 27 mar. 2025. Apache Software Foundation. 2022. Disponível em: https://airflow.apache.org/blog/airflow-survey-2022/.

APACHE SOFTWARE FOUNDATION. **Apache Airflow Documentation: Core Concepts**. Acessado em: 30 de setembro de 2025. 2025. Disponível em: https://airflow.apache.org/docs/apache-airflow/2.11.0/core-concepts/index.html#.

APACHE SOFTWARE FOUNDATION. **Apache Airflow Documentation: TaskFlow API Tutorial**. Acessado em: 30 de setembro de 2025. 2025. Disponível em: https://airflow.apache.org/docs/apache-airflow/2.11.0/tutorial/taskflow.html.

BENDRIKOV, Anton. **Navigating Data Workflows with Dagster: Exploration for Beginners**. Acesso em: 31 mar. 2025. 2024. Disponível em: https://medium.com/@antongw1p/navigating-data-workflows-with-dagster-exploration-for-beginners-f92bf07c2cc6.

DAGSTER Concepts - Getting Started. Acesso em: 31 mar. 2025. Dagster Labs. 2025. Disponível em: https://docs.dagster.io/getting-started/concepts.

DAGSTER Docs. Acesso em: 31 mar. 2025. Dagster Labs. 2025. Disponível em: https://docs.dagster.io/.

DATACAMP. **Dagster vs Airflow: Which Workflow Orchestration Tool Is Better?** [*S.l.*: *s.n.*], 2024. https://www.datacamp.com/blog/dagster-vs-airflow. Acessado em: 12/11/2024.

HOLLINGSWORTH, David. **The Workflow Reference Model**. Winchester, Hampshire, UK, 1995. Document Number TC00-1003, Issue 1.1. Disponível em: http://www.wfmc.org.

MEIRELES, Fernando; SILVA, Denisson; COSTA, Beatriz. **electionsBR: R Functions to Download and Clean Brazilian Electoral Data**. [*S.l.*: *s.n.*], 2021. Acesso em: 20 de junho de 2025. Disponível em: https://cran.r-project.org/package=electionsBR.

PETRAITYTĖ, Ernesta. **Exploring Efficient Workflow Frameworks for Data Management: Optimising Data Management for Sponsoring Consortium for Open Access Publishing in Particle Physics (SCOAP³)**. 2024. MBA in Modern Software and Computing Solutions – Oulu University of Applied Sciences, Oulu, Finland. Thesis examiner(s): Teemu Leppänen.

ROACH, Jake. **Getting Started with Apache Airflow**. Acesso em: 27 mar. 2025. DataCamp. 2022. Disponível em: https://www.datacamp.com/tutorial/getting-started-with-apache-airflow.

TRIBUNAL SUPERIOR ELEITORAL. **DivulgaCandContas 2024 - Informações de Candidatos**. [*S.l.*: *s.n.*], 2024. https://divulgacandcontas.tse.jus.br/divulga/#/candidato/SUL/SC/2045202024. Acesso em: 11 abr. 2025.

VASCONCELOS, Felipe F.; TAVARES, João V. S.; RIBEIRO, Murilo U.; COUTINHO, Fabio J.; CLARINDO, João Paulo. CandiDATA: um dataset para análise das eleições no Brasil. *In*: ANAIS do XXXVI Simpósio Brasileiro de Bancos de Dados (SBBD). Maceió, AL, Brasil: [*s.n.*], 2021. Acesso em: 20 de junho de 2025.

VASSILIADIS, Panos; SIMITSIS, Alkis. Extraction, Transformation, and Loading. *In*: LIU, Ling; ÖZSU, M. Tamer (Ed.). **Encyclopedia of Database Systems**. Boston, MA: Springer, 2009.

VASSILIADIS, Panos; SIMITSIS, Alkis; SKIADOPOULOS, Spiros. Conceptual modeling for ETL processes. *In*: PROCEEDINGS of the 5th ACM international workshop on Data Warehousing and OLAP. [*S.l.*]: ACM, 2002.

Apêndices

APÊNDICE A – CÓDIGO FONTE

O código-fonte do scraper desenvolvido neste trabalho está publicamente disponível no Codigos@UFSC, no endereço: https://codigos.ufsc.br/tcc-victor-gouvea/scraper-candidatos-sc. Da mesma forma, seguem os repositórios com o código fonte dos workflows desenvolvidos usando Airflow e Dagster, respectivamente: https://codigos.ufsc.br/tcc-victor-gouvea/airflow-candidatos-sc e https://codigos.ufsc.br/tcc-victor-gouvea/dagster-candidatos-sc.

Todos os repositórios contém um arquivo `README.md` contendo informações sobre o código desenvolvido e como executar as aplicações.

APÊNDICE B – ARTIGO NO FORMATO SBC SOBRE ESTE TCC

Estudo comparativo de ferramentas de orquestração: Dagster e Airflow

Victor Rodrigues Gouvêa[1], **Renato Fileto**[1]

[1] Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística - INE, Florianópolis, SC – Brasil

victor.gouvea@grad.ufsc.br, r.fileto@ufsc.br

Resumo. *A crescente complexidade dos ecossistemas de dados torna as ferramentas de orquestração essenciais para a automação e governança de fluxos de trabalho. Este trabalho apresenta um estudo comparativo empírico entre o Apache Airflow e o Dagster. Foram desenvolvidos e executados pipelines de Extração, Transformação e Carga (ETL) em ambas as ferramentas, processando dados das eleições de 2022 (arquivos CSV) e 2024 (coleta via web scraping de JSON) em Santa Catarina. A análise comparativa abrangeu três eixos: experiência de desenvolvimento, interface/monitoramento e desempenho. O Dagster destacou-se pela experiência de desenvolvimento, promovendo reutilização de código através de ativos particionados e maior observabilidade. Em contrapartida, o Airflow mostrou-se mais eficiente em cargas massivas, sendo mais de duas vezes mais rápido no cenário mais intensivo. Conclui-se que a escolha envolve um trade-off entre produtividade/governança (Dagster) e desempenho bruto (Airflow).*

Abstract. *The growing complexity of data ecosystems makes orchestration tools essential for workflow automation and governance. This work presents an empirical comparative study between Apache Airflow and Dagster. Extract, Transform, and Load (ETL) pipelines were developed and executed using both tools, processing election data from 2022 (CSV files) and 2024 (web scraping of JSON data) in Santa Catarina, Brazil. The comparative analysis covered three axes: development experience, interface/monitoring, and performance. Dagster stood out for its development experience, promoting code reuse through partitioned assets and greater observability. In contrast, Airflow proved more efficient in massive data loads, being over twice as fast in the most intensive scenario. It is concluded that the choice involves a trade-off between productivity/governance (Dagster) and raw performance (Airflow).*

1. Introdução

O aumento exponencial do volume de dados consolidou os processos de ETL (*Extraction, Transformation, and Load*) como componentes críticos em arquiteturas modernas. Para gerenciar esses fluxos, ferramentas de orquestração dedicadas tornaram-se indispensáveis para garantir execução eficiente e resiliente [Vassiliadis and Simitsis 2009].

Atualmente, o Apache Airflow e o Dagster destacam-se como soluções líderes, porém com paradigmas distintos. O Airflow, lançado em 2014, utiliza Grafos Acíclicos Direcionados (DAGs) para orquestrar tarefas imperativas (*Tasks*), focando na execução

de atividades [Apache Software Foundation 2025]. Já o Dagster introduz o conceito de *Software-Defined Assets*, onde a orquestração é centrada na produção e manutenção de ativos de dados (tabelas, arquivos), permitindo que a ferramenta conheça a linhagem e o tipo dos dados antes da execução [Bendrikov 2024].

Enquanto a literatura aponta o Airflow como uma solução madura e o Dagster como uma alternativa focada na experiência do desenvolvedor [DataCamp 2024], há uma escassez de comparações empíricas em cenários de implementação real. O objetivo deste trabalho é preencher essa lacuna, comparando as ferramentas sob a ótica da experiência de desenvolvimento, usabilidade da interface e desempenho computacional (tempo e recursos) ao processar cargas de dados heterogêneas das eleições de Santa Catarina.

2. Cenário e Organização dos Dados

O estudo de caso integra dados de candidatos de Santa Catarina referentes às eleições de 2022 e 2024. A heterogeneidade das fontes foi intencional para avaliar a flexibilidade das ferramentas:

- **Eleições 2022:** Dados oficiais do TSE em arquivos CSV estáticos. A coleta consistiu no download e armazenamento local, representando um fluxo de baixa complexidade estrutural.
- **Eleições 2024:** Dados coletados do sistema *DivulgaCandContas* via *web scraping* personalizado. O resultado desta coleta foi consolidado em arquivos JSON aninhados, onde cada objeto representa um candidato e contém listas de suas transações financeiras, resultando em uma estrutura hierárquica complexa e um volume de dados significativamente maior e mais diverso que o de 2022.

3. Implementação dos Processos de ETL

Para assegurar uma comparação justa, definiu-se uma arquitetura de referência padronizada (Figura 1). O fluxo segue as etapas clássicas: Extração (leitura local dos arquivos CSV ou JSON), Transformação (limpeza e normalização) e Carga (inserção em um banco de dados relacional).

Ambos os ambientes foram executados em *containers* Docker isolados, separando o banco de metadados da ferramenta do banco de dados de destino.

3.1. Orquestração no Apache Airflow

Utilizando a *TaskFlow API*, a implementação exigiu a criação de duas DAGs distintas para cada eleição. O design adotou um padrão de *fan-out* na etapa de carga para paralelismo, a qual foi gerenciada pelo `PostgresHook`, um componente nativo do Airflow que abstrai e centraliza a conexão com o PostgreSQL.

3.2. Orquestração no Dagster

O Dagster utilizou o grafo de ativos (*Asset Graph*). Diferente do Airflow, não houve duplicação de fluxos: utilizou-se o conceito de *Partitions* (chaves "2022"e "2024") e *Resources* (para conexão com banco). O mesmo código de ativos processa ambos os anos, adaptando-se dinamicamente.

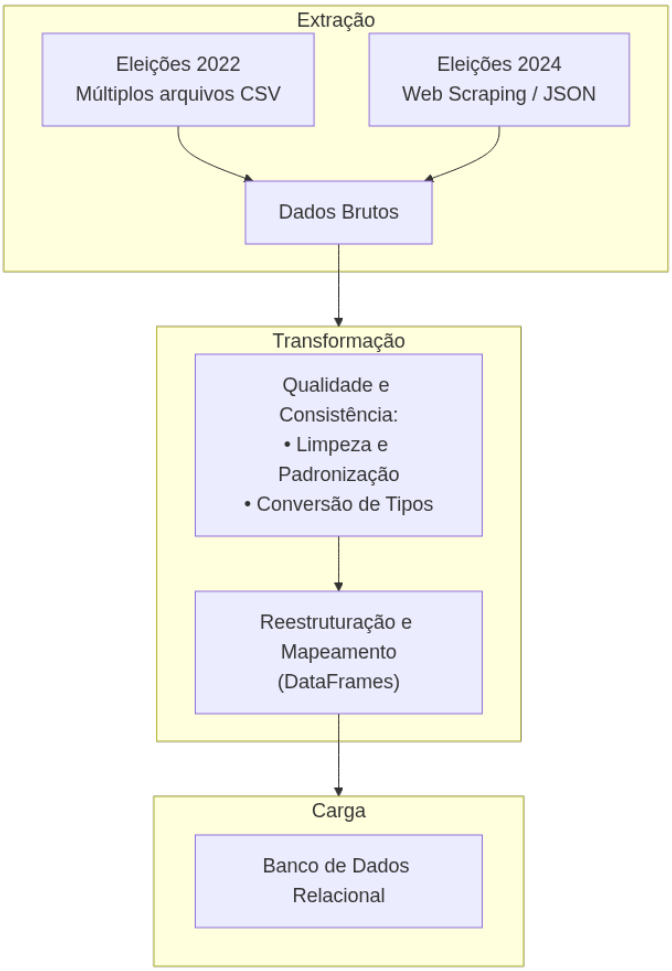


Figura 1. Arquitetura de ETL comum entre as implementações.

4. Análise Comparativa

A implementação dos pipelines de ETL revelou diferenças significativas nas práticas de desenvolvimento e no comportamento em execução de cada ferramenta. Esta seção detalha a análise comparativa baseada na experiência prática e nas métricas coletadas.

4.1. Experiência de Desenvolvimento

A experiência foi avaliada sob três aspectos: complexidade conceitual, infraestrutura e modularidade.

4.1.1. Complexidade Conceitual

Uma análise inicial da documentação sugere uma barreira de entrada maior no Dagster, que apresenta cerca de 16 abstrações centrais (*Assets, Jobs, Resources, IO Managers*, etc), contra aproximadamente 8 no Airflow (*DAGs, Operators, Tasks*, etc). No entanto, ao mapear esses conceitos para o *Workflow Reference Model* (WfRM) [Hollingsworth 1995], percebe-se que a complexidade real reside na sofisticação da implementação e não na quantidade de conceitos.

A divergência fundamental ocorre na definição da atividade (*Process Activity*). O Airflow adere ao modelo tradicional, onde uma *Task* é uma unidade discreta de processamento. O Dagster estende esse conceito: um *Asset* representa não apenas a computação, mas também o artefato de dados persistido. Essa abordagem orientada a dados enriquece a semântica do pipeline, permitindo que a ferramenta gerencie nativamente a linhagem e o estado dos dados, algo que no Airflow depende inteiramente da implementação do desenvolvedor.

Além disso, a abstração de conexões externas difere significativamente. O Airflow utiliza *Hooks*, que são instanciados de forma imperativa dentro das tarefas. O Dagster utiliza *Resources*, que seguem um padrão de injeção de dependência, promovendo um desacoplamento mais robusto entre a lógica de negócio e a infraestrutura.

4.1.2. Setup e Infraestrutura

Ambas as ferramentas foram configuradas em ambientes containerizados utilizando Docker, porém com níveis de complexidade arquitetural distintos.

O **Airflow**, para simular uma arquitetura distribuída fiel à de produção, exigiu a orquestração de uma vasta gama de serviços interdependentes. O *setup* padrão incluiu componentes robustos de infraestrutura, como um banco de dados para metadados (PostgreSQL) e um *broker* de mensagens (Redis), essenciais para a comunicação assíncrona entre o *scheduler* e os *workers*. Além destes, foram instanciados o *webserver*, o *triggerer* e serviços de inicialização. Essa abordagem garante consistência com ambientes produtivos em escala, mas resulta em um tempo de inicialização mais longo e impõe uma barreira de entrada maior para o desenvolvimento local.

O **Dagster**, por outro lado, adota uma arquitetura modular que separa o plano de controle do código do usuário. O *setup* local mostrou-se significativamente mais enxuto, exigindo apenas três componentes principais: o servidor web, o *daemon* (gerenciador de agendamentos) e o banco de dados para persistência. Diferentemente do Airflow, o Dagster não exige nativamente um *broker* de mensagens para o funcionamento básico local, comunicando-se diretamente com o ambiente de execução. Isso tornou a experiência de *setup* mais direta e o ciclo de *feedback* durante o desenvolvimento notavelmente mais rápido.

Essa diferença representa um *trade-off* claro: o Airflow prioriza a paridade com a arquitetura distribuída de produção desde o início, enquanto o Dagster prioriza a velocidade de iteração e a produtividade do desenvolvedor.

4.1.3. Reutilização de Código e Modularidade

A necessidade de processar duas fontes de dados distintas (2022 e 2024) com lógicas similares evidenciou as diferenças de modularidade.

No **Airflow**, a arquitetura baseada em DAGs exigiu a criação de dois fluxos separados. Embora funções Python auxiliares tenham sido compartilhadas, a definição da orquestração (as tarefas e suas dependências) precisou ser duplicada. Isso resultou na definição de **23 tarefas** distintas para cobrir o escopo do projeto, gerando código repetitivo

para operações idênticas, como a carga no banco de dados.

No **Dagster**, a modularidade foi superior devido ao uso nativo de *Partitions*. Foi definida uma única lógica de ativos parametrizada por chaves estáticas ("2022", "2024"). O mesmo código de ativo processou ambos os anos, adaptando-se dinamicamente ao contexto da partição. Isso reduziu o escopo para apenas **17 ativos**, eliminando a duplicação da lógica de orquestração. Além disso, a utilização de *Resources* permitiu que a lógica de conexão com o banco de dados fosse injetada nos ativos.

4.2. Interface e Observabilidade

A interface do usuário (UI) reflete os paradigmas de cada ferramenta e impacta diretamente a eficiência na depuração de erros.

No **Airflow**, a interface centrada em tarefas (*Tasks*) torna a depuração um processo segmentado. A identificação de falhas exige navegar até a DAG específica e localizar o erro na visualização de grade (*Grid View*). O acesso aos logs demanda múltiplos cliques em abas distintas, e a ferramenta não apresenta outras opções de depuração ou monitoramento de falhas mais acessíveis.

Em contraste, a interface do **Dagster** é centrada no grafo de ativos de dados. A observabilidade é integrada: ao detectar uma falha, a ferramenta destaca o ativo problemático diretamente na linhagem global. Isso permite entender imediatamente o impacto do erro nos dados dependentes (*downstream*), uma vez que o grafo visualiza a interdependência dos artefatos.

Além disso, ao selecionar o ativo falho, o Dagster apresenta uma visão detalhada contendo o rastreamento do erro e metadados da execução na mesma tela. Essa integração de informações agiliza o ciclo de correção, permitindo ao desenvolvedor re-materializar o ativo diretamente dessa visualização e preservando o contexto do erro.

Essa comparação evidencia que, enquanto o Airflow oferece um monitoramento funcional focado no processo, o Dagster entrega uma experiência mais moderna e contextualizada, reduzindo a fricção entre a identificação do erro e sua resolução.

4.3. Comparação de Desempenho

Para complementar a análise qualitativa, realizou-se uma avaliação quantitativa focada em tempo de execução e consumo de recursos. O objetivo foi medir o comportamento das ferramentas sob diferentes cargas de trabalho em um ambiente controlado e isolado.

Todas as execuções ocorreram na mesma configuração de hardware e utilizaram a arquitetura containerizada com Docker, assegurando que quaisquer variações de desempenho fossem atribuíveis unicamente às ferramentas. Adicionalmente, cada pipeline foi executado a partir de um estado limpo, com o banco de dados da aplicação vazio, de modo a medir o desempenho de uma carga completa e evitar a influência de dados pré-existentes ou caches.

As métricas de tempo de execução foram extraídas diretamente das interfaces de usuário de cada ferramenta, enquanto o consumo de recursos (CPU e Memória) foi monitorado durante os picos de processamento por meio do comando `docker stats`.

4.3.1. Tempo de Execução

O tempo total de processamento, da extração à carga final, foi o principal indicador de eficiência. A Tabela 1 apresenta os resultados.

Tabela 1. Comparativo do tempo de execução total dos pipelines.

Cenário	Airflow (HH:MM:SS)	Dagster (HH:MM:SS)
Eleições 2022	**00:03:32**	00:05:37
Eleições 2024	**01:16:56**	02:42:50

No cenário de 2024, que envolveu o processamento massivo de aproximadamente 360.000 registros, o Airflow mostrou-se mais de duas vezes mais rápido que o Dagster. A análise detalhada das etapas revelou que o gargalo do Dagster concentrou-se na fase de carga, especificamente na materialização da tabela de Despesas. Enquanto o `PostgresHook` do Airflow gerenciou eficientemente a inserção em lote, a camada de abstração de *Resources* e o gerenciamento de eventos do Dagster parecem ter introduzido uma latência significativa.

Outro fator determinante para a performance superior do Airflow é sua arquitetura distribuída, que utiliza o Redis como *message broker*. Esse mecanismo de fila altamente otimizado permite uma comunicação de baixa latência entre o agendador e os executores, garantindo que a lógica de orquestração não se torne um gargalo durante o processamento massivo, ao contrário do *setup* mais centralizado do Dagster.

4.3.2. Consumo de Recursos (CPU e Memória)

A investigação do consumo de CPU e memória permitiu identificar a origem das discrepâncias de desempenho. A análise foi dividida em dois estados: repouso e execução de pico.

Consumo em Repouso: Com os pipelines inativos, mediu-se o impacto da arquitetura de cada ferramenta. A Tabela 2 resume os resultados.

Tabela 2. Comparativo de consumo de memória em repouso.

Ferramenta	Total (Aprox.)	Componentes de Maior Consumo
Dagster	870 MiB	daemon (434 MiB), webserver (259 MiB)
Airflow	**3.44 GiB**	webserver (1.33 GiB), worker (1.32 GiB)

A arquitetura do Airflow, composta por múltiplos serviços (Redis, Worker, Scheduler, Webserver), apresentou um consumo significativamente maior em comparação ao *setup* mais enxuto do Dagster. Isso confirma que o Airflow exige mais recursos apenas para manter sua infraestrutura operacional disponível.

Consumo em Execução: Durante a etapa de Carga, no momento em que o maior número de tarefas ou *assets* estavam sendo executados em paralelo, monitorou-se o pico de utilização para identificar gargalos. A Tabela 3 apresenta os dados comparativos para ambos os cenários.

Tabela 3. Comparativo de consumo de recursos de pico durante a etapa de Carga.

Pipeline	Componente	Airflow		Dagster	
		CPU %	**Memória**	**CPU %**	**Memória**
Eleições 2022	Executor (*Worker*/*Webserver*)	75.11%	**1.76 GiB**	**156.81%**	1.45 GiB
	BD de Candidatos	**92.01%**	31 MiB	38.06%	**93 MiB**
	BD de Metadados	85.45%	74 MiB	**115.78%**	**85 MiB**
Eleições 2024	Executor (*Worker*/*Webserver*)	304.84%	**4.44 GiB**	**383.11%**	2.44 GiB
	BD de Candidatos	**130.19%**	**47 MiB**	75.64%	45 MiB
	BD de Metadados	150.04%	**222 MiB**	**240.44%**	72 MiB

Esses valores capturados evidenciam comportamentos distintos em relação ao consumo de recursos das duas ferramentas:

- **Airflow:** Apresentou um uso equilibrado de todos os componentes. O alto uso de CPU no Banco de Dados de Candidatos indica que a ferramenta conseguiu enviar dados rápido o suficiente para estressar o banco de destino, maximizando o *throughput*.
- **Dagster:** O Executor e o Banco de Metadados ficaram sobrecarregados, enquanto o Banco da Aplicação foi subutilizado.

Isso demonstra que o Dagster sofre de um ***overhead* de orquestração**: o custo computacional para gerenciar a linhagem, registrar eventos de materialização e metadados estruturados é tão alto que se torna o gargalo do processo, impedindo que a inserção de dados ocorra na velocidade máxima suportada pelo banco. Em suma, o Airflow consome mais memória para manter sua arquitetura distribuída, mas converte esse custo em desempenho bruto de execução, enquanto o Dagster paga um preço em performance pela sua rica camada de governança.

5. Considerações Finais

A análise comparativa empírica entre Apache Airflow e Dagster, realizada através da implementação de pipelines de ETL com dados eleitorais de 2022 e 2024, revelou distinções fundamentais que orientam a escolha da ferramenta adequada conforme as prioridades do projeto.

O **Apache Airflow** reafirmou sua posição como uma solução robusta e madura, destacando-se significativamente no quesito desempenho. Nos cenários de carga massiva, o Airflow mostrou-se mais de duas vezes mais rápido que o concorrente, evidenciando uma arquitetura otimizada para execução bruta em lote. No entanto, essa eficiência cobra seu preço na complexidade de infraestrutura e na experiência de desenvolvimento, onde o setup exige múltiplos contêineres com alto consumo de memória em repouso e a criação de pipelines parametrizados demandou duplicação de código, expondo uma menor flexibilidade nativa para padrões de reutilização.

Em contrapartida, o **Dagster** demonstrou ser superior na experiência de desenvolvimento e na governança de dados. Sua abordagem centrada em *Assets* proporcionou uma linhagem de dados clara e uma observabilidade enriquecida por metadados estruturados, facilitando drasticamente a depuração e o monitoramento da qualidade dos dados. Recursos como *Partitions* e *Resources* permitiram uma alta reutilização de código e um

ciclo de feedback ágil. Contudo, essa sofisticação introduziu um overhead de desempenho notável, já que a camada de abstração e gerenciamento de metadados do Dagster gerou tempos de execução consideravelmente maiores em cargas volumosas, indicando que a ferramenta prioriza a produtividade e a governança em detrimento da velocidade bruta de processamento.

Um ponto crucial deste estudo foi o confronto entre a expectativa teórica e a realidade prática. Embora a literatura previsse a maturidade do Airflow e a modernidade do Dagster, a implementação revelou nuances não documentadas. Ao contrário da simplicidade teórica sugerida, o Dagster apresentou um número maior de abstrações conceituais; entretanto, a prática demonstrou que essas abstrações resolvem nativamente complexidades de engenharia que, no Airflow, exigem implementação manual e repetitiva por parte do desenvolvedor. Além disso, a análise prática diagnosticou que o gargalo de desempenho do Dagster não reside apenas na escalabilidade do banco de dados, mas em um severo overhead intrínseco à sua lógica de orquestração e gestão de eventos, uma descoberta que apenas a experimentação empírica pôde quantificar e que desafia as percepções superficiais sobre a escalabilidade da ferramenta.

Como direcionamento para trabalhos futuros, sugerem-se as seguintes frentes de pesquisa:

- **Análise em Ambientes Distribuídos:** Expansão desta análise para ambientes distribuídos em nuvem, visando avaliar a resiliência e a escalabilidade operacional das ferramentas em cenários de produção real.
- **Ampliação da Base de Dados Histórica:** Extensão do conjunto de dados para abranger um espectro temporal maior de eleições, consolidando um repositório robusto para análises políticas longitudinais.
- **Otimização de Desempenho do Dagster:** Investigações focadas na otimização de desempenho, explorando diferentes configurações de executores e estratégias de paralelismo para verificar a possibilidade de mitigar o overhead de orquestração identificado.

Referências

[Apache Software Foundation 2025] Apache Software Foundation (2025). *Apache Airflow Documentation*. Acesso em: mai. 2025.

[Bendrikov 2024] Bendrikov, A. (2024). Navigating data workflows with dagster: Exploration for beginners. Acesso em: mar. 2025.

[DataCamp 2024] DataCamp (2024). Dagster vs airflow: Which workflow orchestration tool is better? https://www.datacamp.com/blog/dagster-vs-airflow. Acessado em: 12/11/2024.

[Hollingsworth 1995] Hollingsworth, D. (1995). The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition, Winchester, UK.

[Vassiliadis and Simitsis 2009] Vassiliadis, P. and Simitsis, A. (2009). Extraction, transformation, and loading. In Liu, L. and Özsu, M. T., editors, *Encyclopedia of Database Systems*. Springer, Boston, MA.

Anexos

ANEXO A – DESCRIÇÃO DETALHADA DOS ARQUIVOS CSV DAS ELEIÇÕES DE 2022

Abaixo estão as descrições detalhadas das estruturas dos arquivos CSV fornecidos pelo Tribunal Superior Eleitoral (TSE), no site do Governo Federal[1], referentes às eleições de 2022. Essas descrições foram extraídas diretamente do documento oficial denominado "Leiame", disponibilizado pelo próprio TSE juntamente aos arquivos CSV.

[1] https://dados.gov.br/dados/conjuntos-dados/candidatos-2022

LEIAME

Este arquivo contém o leiaute das tabelas existentes no repositório de dados eleitorais. Antes de trabalhar os dados é importante ler as seguintes considerações:

- A codificação de caracteres dos arquivos é "Latin 1";
- Os campos estão entre aspas e separados por ponto e vírgula, inclusive os campos numéricos;
- Campos preenchidos com #NULO significam que a informação está em branco no banco de dados. O correspondente para #NULO nos campos numéricos é -1;
- Campos preenchidos com #NE significam que naquele ano a informação não era registrada em banco de dados pelos sistemas eleitorais. O correspondente para #NE nos campos numéricos é -3;
- O campo UF, além das unidades da federação, pode conter alguma das seguintes situações:
    - o BR: quando se tratar de informação a nível nacional;
    - o VT: quando se tratar de voto em trânsito;
    - o ZZ: quando se tratar de Exterior.
- Os arquivos estão em constante processo de atualização e aperfeiçoamento. Alguns arquivos podem estar em branco ou com mensagem de erro devido à indisponibilidade temporária na base de algum estado ou à inexistência daquele arquivo para a época pretendida.

O § 2º do art. 33 da Resolução TSE nº 23.609/2019 estabelece que:

" Os endereços informados para atribuição de CNPJ, comunicações processuais e do Comitê Central de Campanha, telefone pessoal, e-mail pessoal, número do CPF e o documento pessoal de identificação não serão divulgados no DivulgaCandContas e serão juntados como documento sigiloso no processo de registro de candidatura no PJe. (incluído pela Resolução nº 23.729/2024) "

Portanto, são considerados como não divulgáveis, para as Eleições 2024, as informações registradas na variável NR_ CPF_ CANDIDATO dos arquivos de candidaturas do Portal de Dados Abertos.

Agradecemos todas as críticas e sugestões recebidas de pesquisadores e usuários que estão colaborando para a melhoria na qualidade da prestação das informações.

Qualquer sugestão ou dúvida deve ser encaminhada ao e-mail estatistica@tse.jus.br.

I CONSULTA_ CAND

NOTAÇÃO:

CONSULTA_ CAND_ <ANO>_<UF>

CONSULTA_ CAND_ <ANO>_BRASIL

Para o caso de candidatos não divulgáveis, seus dados pessoais são preenchidos com:

- -4, em caso de campos numéricos, exceto campo de idade;
- "NÃO DIVULGÁVEL", em caso de campos textuais;
- "", no caso de campos relativos à idade do candidato e sua data de nascimento.

Variável	Descrição
DT_ GERACAO	Data da extração dos dados para geração do arquivo.
HH_ GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.
ANO_ ELEICAO	Ano de referência da eleição para geração do arquivo. Observação: para eleições suplementares, o ano de referência da eleição é o ano da eleição ordinária correspondente. Por exemplo: em 2016 houve eleições ordinárias. Após a data desta eleição ordinária e antes da próxima, houve eleições suplementares em 2017, 2018 e 2019. As informações destas eleições suplementares estarão divulgadas no arquivo gerado para as eleições 2016.
CD_ TIPO_ ELEICAO	Código do tipo de eleição. Pode assumir os valores: . 1: Eleição Suplementar; . 2: Eleição Ordinária; e . 3: Consulta Popular.
NM_ TIPO_ ELEICAO	Nome do tipo de eleição. Observação: as eleições ordinárias são previstas em Lei, possuem data certa para serem realizadas, ocorrem em anos pares e possuem a periodicidade de 04 em 04 anos. Nas eleições gerais ordinárias são eleitos os cargos de Presidente, Governador, Deputado (Federal e Estadual) e Senador. Nas eleições ordinárias municipais são eleitos os cargos de Prefeito e Vereador. As eleições suplementares são aquelas que não têm periodicidade pré-determinada ou definida e ocorrem quando, eventualmente, se fizerem necessárias. As consultas populares ocorrem sempre que a população é convocada

	a opinar diretamente sobre um assunto específico e importante. Ela pode ser realizada de duas formas: plebiscito (quando o cidadão opina previamente sobre a possível criação de uma lei) e referendo (quando uma lei aprovada por um órgão legislativo é submetida à aceitação ou não das eleitoras e dos eleitores) .
NR_ TURNO	Número do turno da eleição. Observação: no Brasil, as eleições realizam-se por meio de dois sistemas: o sistema majoritário (aplicado aos cargos de Presidente, Vice-Presidente, Governador, Vice-Governador, Prefeito, Vice-Prefeito e Senador) e o sistema proporcional (aplicado aos cargos de Deputado Federal, Deputado Estadual, Deputado Distrital e Vereador) . O sistema majoritário consiste em declarar eleita a candidata ou o candidato que tenha recebido a maioria dos votos a concorrentes (votos válidos, votos anulados e votos anulados sub judice) . Caso a candidata ou o candidato ao cargo indicado no sistema majoritário, com exceção do cargo de Senador, não alcance maioria absoluta destes votos a concorrentes no primeiro turno (mínimo de 50% + 1) , haverá segundo turno em que concorrerão apenas os dois candidatos mais votados. O segundo turno das eleições no Brasil ocorre para os cargos de Presidente, Vice-Presidente da República, Governador e Vice-Governador dos Estados e do Distrito Federal e para Prefeito e Vice-Prefeito de Municípios com mais de 200 mil eleitores. Nos municípios cujo eleitorado é igual ou menor que 200 mil e para o cargo de Senador elege-se a candidata ou o candidato que tenha alcançado a maioria simples dos votos.
CD_ ELEICAO	Código único da eleição no âmbito da Justiça Eleitoral. Observação: este código é único por eleição e por turno, ou seja, cada turno possui seu código de eleição.
DS_ ELEICAO	Descrição da eleição.
DT_ ELEICAO	Data da ocorrência da eleição.
TP_ ABRANGENCIA	Abrangência da eleição. Pode assumir os valores: . M: Municipal; . E: Estadual; e . F: Federal.

	Observação: a abrangência territorial da eleição está diretamente relacionada aos cargos eletivos e suas circunscrições eleitorais. As eleições realizadas na circunscrição Municipal são as eleições para os cargos de Prefeito, Vice-Prefeito e Vereador; as realizadas na circunscrição Estadual são para os cargos de Governador, Vice-Governador, Senador, Deputado Estadual, Deputado Federal e Deputado Distrital e; as realizadas na circunscrição Federal são para os cargos de Presidente e Vice-Presidente da República.
SG_ UF	Sigla da unidade da federação na qual a candidata ou candidato concorre na eleição.
SG_ UE	Sigla da unidade eleitoral na qual a candidata ou candidato concorre na eleição. A unidade eleitoral representa a unidade da federação ou o município em que a candidata ou o candidato concorre na eleição e é relacionada à abrangência territorial desta candidatura. Em caso de abrangência federal (cargo de Presidente e Vice-Presidente) a sigla é BR. Em caso de abrangência estadual (cargos de Governador, Vice-Governador, Senador, Deputado Federal, Deputado Estadual e Deputado Distrital) a sigla é a UF da candidatura. Em caso de abrangência municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código TSE de identificação do município da candidatura.
NM_ UE	Nome da unidade eleitoral da candidata ou candidato. Em caso de abrangência nacional, é igual a "Brasil". Em caso de abrangência estadual, é o nome da UF em que a candidata ou candidato concorre. Em caso de abrangência municipal, é o nome do município em que a candidata ou candidato concorre.
CD_ CARGO	Código do cargo ao qual a candidata ou candidato concorre.
DS_ CARGO	Descrição do cargo ao qual a candidata ou candidato concorre.
SQ_ CANDIDATO	Número sequencial da candidata ou candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha da candidata ou candidato.

NR_ CANDIDATO	Número da candidata ou candidato na urna.
NM_ CANDIDATO	Nome completo da candidata ou candidato.
NM_ URNA_ CANDIDATO	Nome da candidata ou candidato que aparece na urna.
NM_ SOCIAL_ CANDIDATO	Nome social da candidata ou candidato. Observação: nome social é o nome pelo qual pessoas travestis ou transexuais preferem ser chamadas cotidianamente, em contraste com o nome oficialmente registrado, que não reflete sua identidade de gênero. A identidade do nome social é vinculada com a identidade civil original. Em âmbito federal, o Decreto nº 8.727 de 2016, garante o direito ao uso do nome social e reconhecimento da identidade de gênero de pessoas travestis e transexuais no âmbito da administração pública federal direta, autárquica e fundacional
NR_ CPF_ CANDIDATO	Número do CPF da candidata ou candidato.
DS_ EMAIL	Endereço de e-mail da candidata ou candidato.
CD_ SITUACAO_ CANDIDATURA	Código da situação do registro de candidatura da candidata ou candidato. Aplicável somente para os registros das eleições até 2022.
DS_ SITUACAO_ CANDIDATURA	Descrição da situação do registro de candidatura da candidata ou candidato. Aplicável somente para os registros das eleições até 2022.
TP_ AGREMIACAO	Tipo de agremiação da candidatura, ou seja, forma como a candidata ou o candidato concorrerá nas eleições. Pode assumir os valores: . Coligação: quando a candidata ou candidato concorre por coligação; . Federação: quando a candidata ou candidato concorre por uma federação; e . Partido Isolado: quando a candidata ou candidato concorre somente pelo partido.
NR_ PARTIDO	Número do partido da candidata ou candidato.
SG_ PARTIDO	Sigla do partido da candidata ou candidato.
NM_ PARTIDO	Nome do partido da candidata ou candidato.
NR_ FEDERACAO	Número da federação pela qual o partido da candidata ou candidato concorre na eleição.
NM_ FEDERACAO	Nome da federação pela qual o partido da candidata ou candidato concorre na eleição.
SG_ FEDERACAO	Sigla da federação pela qual o partido da candidata ou

	candidato concorre na eleição.
DS_ COMPOSICAO_ FEDERACAO	Composição da federação pela qual o partido da candidata ou candidato concorre na eleição. A informação da federação é composta pela concatenação das siglas dos partidos federados intercaladas com o símbolo "/".
SQ_ COLIGACAO	Sequencial da coligação pela qual o partido da candidata ou candidato concorre na eleição.
NM_ COLIGACAO	Nome da coligação pela qual o partido da candidata ou candidato concorre na eleição.
DS_ COMPOSICAO_ COLIGACAO	Composição da coligação pela qual o partido da candidata ou candidato concorre na eleição. A informação da coligação é composta pela concatenação das siglas dos partidos coligados intercaladas com ",". Observação: quando o tipo de agremiação é por partido isolado ou federação a coluna DS_ COMPOSICAO_ COLIGACAO é preenchida, respectivamente, com a sigla do partido e a sigla da federação.
SG_ UF_ NASCIMENTO	Sigla da unidade da federação de nascimento da candidata ou candidato.
DT_ NASCIMENTO	Data de nascimento da candidata ou candidato.
NR_ TITULO_ ELEITORAL_ CANDIDATO	Número do título eleitoral da candidata ou candidato.
CD_ GENERO	Código do gênero da candidata ou candidato. Pode assumir os valores: . 2: Masculino; e . 4: Feminino.
DS_ GENERO	Descrição do gênero da candidata ou candidato.
CD_ GRAU_ INSTRUCAO	Código do grau de instrução da candidata ou candidato. Pode assumir os valores: . 1: Analfabeto; . 2: Lê e escreve; . 3: Ensino fundamental incompleto; . 4: Ensino fundamental completo; . 5: Ensino médio incompleto; . 6: Ensino médio completo; . 7: Superior incompleto; e . 8: Superior completo.
DS_ GRAU_ INSTRUCAO	Descrição do grau de instrução da candidata ou candidato.
CD_ ESTADO_ CIVIL	Código do estado civil da candidata ou candidato. Pode

	assumir os valores: . 1: Solteiro(a) ; . 3: Casado(a) ; . 5: Viúvo(a) ; . 7: Separado(a) judicialmente; e . 9: Divorciado(a) .
DS_ ESTADO_ CIVIL	Descrição do estado civil da candidata ou candidato.
CD_ COR_ RACA	Código da cor/raça da candidata ou candidato. Pode assumir os valores: . 01: Branca; . 02: Preta; . 03: Parda; . 04: Amarela; . 05: Indígena; e . 06: Não Informado.
DS_ COR_ RACA	Descrição da cor/raça da candidata ou candidato.
CD_ OCUPACAO	Código da ocupação da candidata ou candidato.
DS_ OCUPACAO	Descrição da ocupação da candidata ou candidato.
CD_ SIT_ TOT_ TURNO	Código da situação de totalização da candidata ou candidato no turno.
DS_ SIT_ TOT_ TURNO	Descrição da situação de totalização da candidata ou candidato no turno.

LEIAME

Este arquivo contém o leiaute das tabelas existentes no repositório de dados eleitorais. Antes de trabalhar os dados é importante ler as seguintes considerações:

- A codificação de caracteres dos arquivos é "Latin 1";
- Os campos estão entre aspas e separados por ponto e vírgula, inclusive os campos numéricos;
- Campos preenchidos com #NULO significam que a informação está em branco no banco de dados. O correspondente para #NULO nos campos numéricos é -1;
- Campos preenchidos com #NE significam que naquele ano a informação não era registrada em banco de dados pelos sistemas eleitorais. O correspondente para #NE nos campos numéricos é -3;
- O campo UF, além das unidades da federação, pode conter alguma das seguintes situações:
  - o BR: quando se tratar de informação a nível nacional;
  - o VT: quando se tratar de voto em trânsito;
  - o ZZ: quando se tratar de Exterior.
- Os arquivos estão em constante processo de atualização e aperfeiçoamento. Alguns arquivos podem estar em branco ou com mensagem de erro devido à indisponibilidade temporária na base de algum estado ou à inexistência daquele arquivo para a época pretendida.

Agradecemos todas as críticas e sugestões recebidas de pesquisadores e usuários que estão colaborando para a melhoria na qualidade da prestação das informações. Em especial à Associação Brasileira de Ciência Política (ABCP) que, por meio de acordo de cooperação técnica firmado com o TSE, está auxiliando na verificação dos arquivos gerados e informando o TSE quanto às necessidades de dados dos pesquisadores.

Qualquer sugestão ou dúvida deve ser encaminhada ao e-mail estatistica@tse.jus.br.

I BEM_CANDIDATO

NOTAÇÃO:

BEM_ CANDIDATO_<ANO>_<UF>

BEM_ CANDIDATO_ <ANO>_BRASIL

Variável	Descrição
DT_ GERACAO	Data da extração dos dados para geração do arquivo.
HH_ GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.
ANO_ ELEICAO	Ano de referência da eleição para geração do arquivo. Observação: para eleições suplementares, o ano de referência da eleição é o ano da eleição ordinária correspondente. Por exemplo: em 2016 houve eleições ordinárias. Após a data desta eleição ordinária e antes da próxima, houve eleições suplementares em 2017, 2018 e 2019. As informações destas eleições suplementares estarão divulgadas no arquivo gerado para as eleições 2016.
CD_ TIPO_ ELEICAO	Código do tipo de eleição. Pode assumir os valores: . 1: Eleição Suplementar; . 2: Eleição Ordinária; e . 3: Consulta Popular.
NM_ TIPO_ ELEICAO	Nome do tipo de eleição. Observação: as eleições ordinárias são previstas em Lei, possuem data certa para serem realizadas, ocorrem em anos pares e possuem a periodicidade de 04 em 04 anos. Nas eleições gerais ordinárias são eleitos os cargos de Presidente, Governador, Deputado (Federal e Estadual) e Senador. Nas eleições ordinárias municipais são eleitos os cargos de Prefeito e Vereador. As eleições suplementares são aquelas que não têm periodicidade pré-determinada ou definida e ocorrem quando, eventualmente, se fizerem

	necessárias. As consultas populares ocorrem sempre que a população é convocada a opinar diretamente sobre um assunto específico e importante. Ela pode ser realizada de duas formas: plebiscito (quando o cidadão opina previamente sobre a possível criação de uma lei) e referendo (quando uma lei aprovada por um órgão legislativo é submetida à aceitação ou não das eleitoras e dos eleitores) .
CD_ ELEICAO	Código único da eleição no âmbito da Justiça Eleitoral. Observação: este código é único por eleição e por turno, ou seja, cada turno possui seu código de eleição.
DS_ ELEICAO	Descrição da eleição.
DT_ ELEICAO	Data da ocorrência da eleição.
SG_ UF	Sigla da unidade da federação na qual a candidata ou candidato concorre na eleição.
SG_ UE	Sigla da unidade eleitoral na qual a candidata ou candidato concorre na eleição. A unidade eleitoral representa a unidade da federação ou o município em que a candidata ou o candidato concorre na eleição e é relacionada à abrangência territorial desta candidatura. Em caso de abrangência federal (cargo de Presidente e Vice-Presidente) a sigla é BR. Em caso de abrangência estadual (cargos de Governador, Vice-Governador, Senador, Deputado Federal, Deputado Estadual e Deputado Distrital) a sigla é a UF da candidatura. Em caso de abrangência municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código TSE de identificação do município da candidatura.
NM_ UE	Nome da unidade eleitoral da candidata ou candidato. Em caso de abrangência nacional, é igual a "Brasil". Em caso de abrangência estadual, é o nome da UF

	em que a candidata ou candidato concorre. Em caso de abrangência municipal, é o nome do município em que a candidata ou candidato concorre.
SQ_ CANDIDATO	Número sequencial da candidata ou candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha da candidata ou candidato.
NR_ ORDEM_ BEM_ CANDIDATO	Número de ordenação do bem patrimonial material da candidata ou candidato de acordo com a sequência de bens declarados pela(o) mesma(o) .
CD_ TIPO_ BEM_ CANDIDATO	Código do tipo do bem patrimonial material declarado pela candidata ou candidato.
DS_ TIPO_ BEM_ CANDIDATO	Tipo do bem patrimonial material declarado pela candidata ou candidato.
DS_ BEM_ CANDIDATO	Descrição detalhada do bem material patrimonial da candidata ou candidato.
VR_ BEM_ CANDIDATO	Valor, em reais, do bem patrimonial material declarado pela candidata ou candidato.
DT_ ULT_ ATUAL_ BEM_ CANDIDATO	Data da última atualização, no sistema, do bem material patrimonial declarado pela candidata ou candidato.
HH_ ULT_ ATUAL_ BEM_ CANDIDATO	Hora da última atualização, no sistema, do bem material patrimonial declarado pela candidata ou candidato.

LEIA-ME

Este arquivo contém o leiaute das tabelas existentes no repositório de dados eleitorais. Antes de trabalhar os dados é importante ler as seguintes considerações:

- A codificação de caracteres dos arquivos é "Latin 1";
- Os campos estão entre aspas e separados por ponto e vírgula, inclusive os campos numéricos;
- Campos preenchidos com #NULO significam que a informação está em branco no banco de dados. O correspondente para #NULO nos campos numéricos é -1;
- Campos preenchidos com #NE significam que naquele ano a informação não era registrada em banco de dados pelos sistemas eleitorais. O correspondente para #NE nos campos numéricos é -3;
- O campo UF, além das unidades da federação pode conter alguma das seguintes situações:
  - BR: quando se tratar de informação a nível nacional;
  - VT: quando se tratar de voto em trânsito;
  - ZZ: quando se tratar de Exterior.
- Os arquivos estão em constante processo de atualização e aperfeiçoamento. Alguns arquivos podem estar em branco ou com mensagem de erro devido a indisponibilidade temporária na base de algum estado ou à inexistência daquele arquivo para a época pretendida.

Agradecemos todas as críticas e sugestões recebidas de pesquisadores e usuários que estão colaborando para a melhoria na qualidade da prestação das informações. Em especial à Associação Brasileira de Ciência Política (ABCP), que por meio de acordo de cooperação técnica firmado com o TSE está auxiliando na verificação dos arquivos gerados e informando o TSE quanto às necessidades de dados dos pesquisadores.

Qualquer sugestão ou dúvida deve ser encaminhada ao e-mail estatistica@tse.jus.br.

NOTA_ FISCAL_CANDIDATO

NOTAÇÃO:
<NOTA_FISCAL_ CANDIDATO>_<ANO>_<UF>

Variável	Descrição
DT_GERACAO	Data de geração do arquivo (data da extração dos dados).
HH_GERACAO	Hora de geração do arquivo (hora da extração) - Horário de Brasília.
NR_ANO_REFERENCIA	Ano de referência.
CD_ELEICAO	Código da eleição.
NM_URNA	Nome do candidato que aparece na urna.
NR_CANDIDATO	Número do candidato na urna.
SG_UF_UA	Sigla UF da unidade arrecadadora.
SG_UE	Sigla da Unidade Eleitoral (UE) em que o candidato concorre na eleição. A Unidade Eleitoral representa a Unidade da Federação ou o Município em que o candidato concorre na eleição e é relacionada à abrangência territorial desta candidatura. Em caso de abrangência Federal (cargo de Presidente e Vice-Presidente) a sigla é BR. Em caso de abrangência Estadual (cargos de Governador, Vice-Governador, Senador, Deputado Federal, Deputado Estadual e Deputado Distrital) a sigla é a UF da candidatura. Em caso de abrangência Municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código de identificação do município da candidatura.
NM_UE_UA	Nome da UE da unidade arrecadadora.
CONTRATANTE	Contratante: Titular, Vice.
NR_CPF_CNPJ_EMITENTE	Número do CPF ou CNPN do emissor da nota fiscal.
CD_NATUREZA_OPERACAO	Código da natureza da operação que envolve um código conhecido como CFOP com o qual hoje é informado item a item no detalhamento da nota fiscal eletrônica. Exemplo: 5102, 5929, SERV, DEVO.

CD_MODELO	Modelo da Nota Fiscal, exemplo: 00, 55, UN.
NM_UNIDADE_ARRECADADORA	Nome da unidade arrecadadora.
DT_EMISSAO	Data da emissão da nota fiscal.
NR_NOTA_FISCAL	Número da nota fiscal.
NR_SERIE	Número da série da nota fiscal.
VR_NOTA_FISCAL	Valor da nota fiscal.
NR_CHAVE_ACESSO	Número da chave de acesso da nota fiscal.
NM_URL_ACESSO	Número da URL de acesso da nota fiscal.