PM

Arquitetura para Gestão Eficiente de Dados Turísticos

PROJETO DE MESTRADO

José Filipe Machado Figueira
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

setembro | 2025

Arquitetura para Gestão Eficiente de Dados Turísticos

PROJETO DE MESTRADO

José Filipe Machado Figueira
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Eduardo Miguel Dias Marques

COORIENTAÇÃO
Karolina Baras

Faculdade de Ciências Exatas e da Engenharia

Master of Science Degree in Informatics Engineering

Arquitetura para Gestão Eficiente de Dados Turísticos

Supervised by: Eduardo Marques e Karolina Baras

Constituição do júri de provas públicas:

Leonel Domingos Telo Nóbrega (categoria), Presidente

David Sardinha Andrade de Aveiro (categoria), Vogal

Karolina Baras (categoria), Vogal

30 de dezembro de 2025

Resumo

Nos últimos anos, a digitalização tem vindo a revolucionar progressivamente a indústria do turismo, encaminhando-a para se tornar num setor cada vez mais inteligente. Tecnologias como IoT, redes sociais e plataformas online personalizaram serviços, melhoraram a experiência do cliente e otimizaram operações. Como consequência, surgiu um elevado volume de dados que se tem vindo a tornar num ativo cada vez mais valioso. No entanto, este crescimento apresenta grandes desafios, como a complexidade de armazenamento e processamento, a normalização de dados heterogéneos e a análise em massa de forma eficiente.

Com foco nos dados turísticos, foi criada uma plataforma no âmbito do projeto SMARTDEST, dedicada à recolha, gestão e disponibilização de dados, possibilitando uma análise detalhada das tendências e dinâmicas de vários indicadores do turismo na Madeira. Contudo, a plataforma apresenta algumas limitações, como falhas no processo de recolha de dados, elevada complexidade na integração de novas fontes, ausência de mecanismos de segurança e soluções de armazenamento pouco escaláveis. Este trabalho propõe-se a reconstruir a arquitetura da plataforma, com especial foco na recolha, tratamento e armazenamento de dados. O objetivo é superar as limitações anteriormente identificadas, adotando tecnologias mais robustas e escaláveis que assegurem uma maior eficiência e segurança do processo da gestão dos dados.

Keywords: *Big Data*, *ETL(Extract,Transform,Load)*, Dados Turísticos, Arquitetura de *Big Data*, Plataforma SMARTDEST

Abstract

In recent years, digitization has progressively revolutionized the tourism industry, leading it to become an increasingly intelligent sector. Technologies such as IoT, social media, and online platforms have personalized services, improved the customer experience, and optimized operations. Consequently, a large volume of data has emerged, which has become an increasingly valuable asset. However, this growth presents major challenges, such as the complexity of storage and processing, the normalization of heterogeneous data, and efficient mass analysis.

Focusing on tourism data, a platform was created within the scope of the SMARTDEST project, dedicated to collecting, managing, and providing data, enabling a detailed analysis of the trends and dynamics of various tourism indicators in Madeira. However, the platform has some limitations, such as failures in the data collection process, high complexity in integrating new sources, the absence of security mechanisms, and poorly scalable storage solutions. This work proposes to rebuild the platform's architecture, with a special focus on data collection, processing, and storage. The goal is to overcome the previously identified limitations by adopting more robust and scalable technologies that ensure greater efficiency and security in the data management process.

Keywords: *Big Data*, *ETL*(Extract, Transform, Load), Tourism Data, Big Data Architecture, SMARTDEST Platform

Acknowledgments

Sinto uma enorme gratidão a todos os que me acompanharam e apoiaram nesta caminhada.

Em primeiro lugar, à minha família, o meu muito obrigado pelo amor, incentivo e apoio incondicionais. A vossa crença em mim foi a base para que esta conquista se tornasse realidade.

Aos meus amigos, agradeço a companhia, os momentos de diversão e por estarem sempre ao meu lado, independentemente das circunstâncias. A vossa amizade tem sido uma fonte de força e felicidade.

Um agradecimento especial ao meu orientador, Eduardo Marques. A sua orientação, conhecimento e paciência foram essenciais para o sucesso da minha tese.

Estou também grato à minha coorientadora, Karolina Baras, pelo seu acompanhamento e pela oportunidade de crescer e aprender ao longo deste trabalho.

Por fim, a todos os que se cruzaram no meu percurso académico, obrigado. Cada um de vocês, à sua maneira, contribuiu para a minha evolução e aprendizagem, e por isso estou imensamente grato.

Obrigado a todos por fazerem parte desta jornada comigo.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

3NF Third Normal Form

API Application Programming Interface

AWS Amazon Web Services

CPU Central Processing Unit

CRM Customer Relationship Management

DAG Directed Acyclic Graph

DW Data Warehouse

EDW Enterprise Data Warehouse

ELT Extract, Load, Transform

ETL Extract, Transform, Load

FaaS Function as a Service

OLAP Online Analytical Processing

RDBMS Relational Database Management System

S3 Simple Storage Service

SQL Structured Query Language

UI User Interface

1 Introdução

Neste capítulo introdutório, contextualiza-se o problema em análise, apresentando o enquadramento do setor do turismo e a crescente relevância do *big data*. Subsequentemente, detalha-se a motivação subjacente a esta dissertação, os objetivos específicos delineados para o estudo e, por fim, a estrutura do documento.

1.1 Contexto

A indústria do turismo desempenha um papel crucial na promoção do crescimento e desenvolvimento económico. Em 2019, este setor contribuiu significativamente, representando 10,4 por cento do Produto Interno Bruto (PIB) mundial [1].

Antes da digitalização, o setor do turismo dependia fortemente de métodos tradicionais, como agências de viagens físicas e brochuras impressas. A pesquisa de informações era limitada e as reservas eram frequentemente feitas por telefone ou pessoalmente [2]. A introdução da tecnologia transformou drasticamente o setor, proporcionando maior conveniência e acesso a informação para os viajantes [3]. A Internet consolidou-se como uma ferramenta indispensável, tanto para os viajantes como para as empresas do setor resultando na geração de um volume elevado de dados, que se tornaram uma fonte essencial de informação e conhecimento no domínio do turismo [4].

A relevância destes dados deve-se, em grande medida, à transformação significativa que a web sofreu, onde os utilizadores passaram de meros consumidores a criadores de informação [5]. Estes dados são recolhidos em grandes volumes a partir de atividades quotidianas como reservas online e sistemas de recomendação online [6]. A análise destes dados é, frequentemente, uma tarefa complexa e exigente, porque envolve uma série de desafios que decorrem das características intrínsecas dos dados [7]. Estes desafios são geralmente descritos pelos "Vs"do *Big Data*, nomeadamente o Volume, que representa a enorme quantidade de dados gerados e exige soluções avançadas, a Velocidade, que requer sistemas capazes de processar informação em tempo real, a Variedade, relacionada com a diversidade de formatos de dados, e a Veracidade, que sublinha a importância de garantir a precisão e a fiabilidade dos dados [8]. Estas características refletem os desafios na gestão e utilização eficaz de grandes volumes de dados de diversas fontes sendo necessário utilizar ferramentas adequadas para ser possível retirar informação com qualidade [8].

A análise e gestão dos grandes volumes de dados, com as suas características desafiadoras, têm sido aplicadas em diversos setores. No caso do turismo, isso tem levado ao desenvolvimento de soluções inovadoras que ajudam a otimizar a gestão do setor. No arquipélago da Madeira, foi criada uma plataforma no âmbito do projeto SMARTDEST com o objetivo de melhorar a compreensão e a gestão do turismo na região. Esta plataforma reúne elevadas quantidades de dados provenientes de diversas fontes relacionadas com o setor, permitindo a análise, disponibilização e visualização dessas informações [9]. Através desta ferramenta, é possível obter uma visão de alguns padrões e preferências dos turistas, facilitando a análise de dados turísticos e contribuindo para a tomada de decisões mais informadas e eficazes.

1.2 Motivação

A plataforma SMARTDEST foi um avanço na consolidação de dados turísticos da Madeira, mas a sua arquitetura atual tem fragilidades que limitam a sua evolução a longo prazo. O sistema não é escalável nem flexível o suficiente para lidar com o crescimento exponencial e a diversidade das fontes de dados.

A dificuldade em integrar novas fontes, a falta de mecanismos de tolerância a falhas e a capacidade limitada para processar dados não estruturados impedem que se extraia todo o potencial analítico da informação disponível.

Esta dissertação é, portanto, motivada pela necessidade de reestruturar a arquitetura da plataforma SMARTDEST. O objetivo é criar uma solução mais robusta, escalável e eficiente, alinhada com as melhores práticas de engenharia de dados. Pretendemos construir um ecossistema de dados capaz de suportar a ingestão, o processamento e o armazenamento de grandes volumes de dados turísticos, e que inclua um sistema de monitorização abrangente para garantir a fiabilidade e a qualidade de todo o processo.

1.3 Objetivo do Estudo

Para responder à motivação apresentada, foram definidos os seguintes objetivos:

- Analisar criticamente as plataformas e ferramentas de *big data* e orquestração de *workflows* disponíveis no mercado, com foco nas suas capacidades, arquiteturas e modelos de implementação.

- Selecionar um conjunto de tecnologias maduras e adequadas ao contexto do projeto, avaliando critérios como escalabilidade, robustez, flexibilidade e ecossistema de integração.
- Desenvolver um novo ambiente de *big data*, projetado para suportar a ingestão, transformação e armazenamento de dados turísticos em larga escala, provenientes de fontes heterogéneas.
- Implementar processos de extração, transformação e carregamento otimizados, capazes de processar eficientemente tanto dados estruturados como não estruturados.
- Avaliar o desempenho da arquitetura proposta através de cenários de validação práticos, analisando a sua resiliência a falhas, a eficiência do processamento e a capacidade de integração de novas fontes de dados.
- Comparar o desempenho analítico e as capacidades de monitorização da nova arquitetura com a solução original da plataforma SMARTDEST, quantificando as melhorias obtidas.

1.4 Estrutura do Documento

O presente documento está dividido em sete capítulos, organizados de forma a guiar o leitor desde o enquadramento teórico e a contextualização do problema até à apresentação detalhada do trabalho desenvolvido e das conclusões alcançadas.

O Capítulo 1, Introdução, estabelece o ponto de partida do estudo, apresentando o contexto geral em que a pesquisa se insere, a motivação que impulsionou o projeto, o objetivo específico a ser alcançado e, por fim, uma visão geral da estrutura do próprio documento, preparando o leitor para o que será abordado nas secções seguintes.

O Capítulo 2, Estado da Arte, aborda a fundamentação teórica da tese. Nele, são exploradas as tecnologias e conceitos de *big data* aplicados ao setor do turismo, bem como os desafios específicos inerentes à gestão de dados neste contexto. Este capítulo também apresenta as arquiteturas de referência e as soluções de armazenamento de dados mais relevantes, fornecendo o conhecimento necessário para a compreensão das escolhas metodológicas.

A Metodologia é detalhada no Capítulo 3, que descreve o percurso para alcançar os objetivos. Inicia-se com a análise da plataforma SMARTDEST, identificando os seus pontos fracos, seguida pela definição dos requisitos do novo sistema. Posteriormente, são apresentadas a arquitetura conceptual e as tecnologias que foram selecionadas para a construção da solução.

O Capítulo 4, Desenvolvimento, foca-se na implementação prática do projeto. Aqui, é explicada a configuração dos diferentes componentes da nova arquitetura e a implementação do *pipeline* de extração, orquestração e armazenamento de dados, detalhando como os dados turísticos são processados.

No Capítulo 5, Validação da Arquitetura, são apresentados e analisados os resultados obtidos. Este capítulo inclui cenários de validação para diferentes tipos de recolha de dados, discute a forma como o sistema lida com falhas, aborda a integração de novas fontes e a observabilidade da plataforma, culminando numa análise comparativa com a arquitetura original do SMARTDEST.

O Capítulo 6, Conclusão, sintetiza as principais conclusões do trabalho. Neste capítulo, são resumidas as conclusões mais importantes e adicionalmente, são propostas direções para trabalhos futuros, que poderão dar continuidade.

O documento é complementado pelos Anexos, onde se encontram alguns detalhes de código e de configuração relevantes para a implementação.

2 Estado da Arte

Este capítulo tem como objetivo apresentar uma visão abrangente do estado da arte no que respeita à aplicação de *big data* no setor do Turismo, destacando os principais desafios associados à gestão e tratamento dos dados turísticos. A crescente complexidade e volume destes dados evidenciam a necessidade de infraestruturas tecnológicas robustas e de metodologias eficazes para a sua exploração e análise.

Neste contexto, são descritas diferentes arquiteturas de referência para sistemas orientados ao *big data*, com especial foco nos seus principais componentes e funcionalidades. Adicionalmente, é realizada uma análise comparativa de três ferramentas de ETL (Extract, Transform, Load), avaliando a sua adequação a cenários que envolvem a integração e o processamento de grandes volumes de dados. Esta análise visa fornecer uma base teórica e tecnológica sólida que sustenta a proposta metodológica desenvolvida ao longo do presente trabalho.

2.1 Big Data no Turismo

Com o rápido avanço da tecnologia, são gerados, registados, armazenados e acumulados continuamente volumes massivos de dados, estruturados e não estruturados, dando origem ao fenómeno conhecido como *big data* [10]. Apesar da sua crescente relevância, a comunidade científica ainda não alcançou um consenso quanto à definição deste conceito [11]. O artigo em [11] descreve o *big data* com base no modelo dos "4Vs", nomeadamente o Volume, que representa a enorme quantidade de dados gerados, a Velocidade, que requer sistemas capazes de processar informação em tempo real, a Variedade, relacionada com a diversidade de formatos de dados, e a Veracidade, que sublinha a importância de garantir a precisão e a fiabilidade dos dados. Estas particularidades exigem o recurso a tecnologias e metodologias analíticas específicas, capazes de dar resposta aos desafios inerentes para que seja possível retirar informações com qualidade.

Neste contexto, a análise de *big data* tem se revelado uma ferramenta poderosa, oferecendo uma ampla gama de benefícios para organizações de diversos setores. Esta permite a análise eficiente de grandes volumes de dados, auxiliando as empresas na identificação de padrões, tendências e informações valiosas que sustentam a tomada de decisões estratégicas [7].

No setor do turismo, a investigação baseada em *big data* é uma área em crescimento acelerado, que tem despertado um interesse cada vez maior. Esta tendência é visível no aumento significativo

do número de publicações sobre o tema, que passou de apenas três em 2007 para trinta em 2016 [12]. Este campo de estudo utiliza dados provenientes de múltiplas fontes com o objetivo de compreender melhor o comportamento dos turistas e otimizar a gestão do setor [12]. Exemplos de aplicação incluem a identificação de padrões de mobilidade dos turistas [13]; a utilização de modelos de *machine learning* que integram dados de motores de busca, tráfego em websites e redes sociais para prever a procura turística [14]; e a análise de dados que permite às agências de viagens online (OTAs) personalizar preços, otimizar a apresentação de ofertas hoteleiras e disponibilizar informações relevantes sobre os destinos [15].

2.1.1 Desafios Relacionados aos Dados Turísticos

No contexto da utilização de grandes volumes de dados no setor do turismo, continuam a emergir desafios significativos, nomeadamente no que respeita à qualidade dos dados, aos custos associados e às questões de privacidade [16].

De acordo com o estudo apresentado em [12], foi realizada uma análise abrangente de diversos trabalhos sobre a aplicação de *big data* no domínio do turismo. Através dessa análise, o autor conseguiu agrupar as fontes de dados em três grandes categorias, identificando, para cada uma delas, os contextos de investigação em que são normalmente aplicadas, bem como as suas principais vantagens e limitações.

Na primeira linha da Tabela 1, encontram-se os dados pertencentes ao grupo "Utilizadores". Este grupo engloba conteúdos gerados pelos próprios turistas, como avaliações, blogs e fotografias partilhadas online, que refletem as suas experiências e opiniões. Estes dados são amplamente utilizados em estudos de análise de sentimentos, na formulação de recomendações personalizadas e na compreensão do comportamento dos turistas.

Na segunda linha da Tabela 1, são apresentados os dados do grupo "Dispositivos". Estes provêm de sensores e equipamentos diversos, como sistemas GPS, dados de roaming e tecnologia Bluetooth. Pela sua capacidade de rastrear deslocações, são frequentemente utilizados para estudar os padrões espaciais e temporais de mobilidade dos turistas.

Por fim, a terceira linha da Tabela 1 corresponde ao grupo "Operações". Este conjunto de dados resulta das interações dos turistas com plataformas digitais, incluindo pesquisas online, visitas a websites e reservas efetuadas na Internet. Trata-se de uma fonte valiosa para a análise do

comportamento digital dos turistas, com especial relevância para a previsão da procura turística, uma vez que permite captar tendências e padrões de interação com o mercado turístico online.

Comparison among different types of big data in tourism research.

Data source	Data type	Research focuses	Advantages	Disadvantages
Users	Online textual data	Tourist sentiment analysis; Tourism recommendation	Low cost; Conveying tourist sentiment	Reliability concerns
	Online photo data	Tourist behavior analysis; Tourism marketing; Tourism recommendation	Low cost; Containing multi-metadata	Low precision of location;
Devices	GPS data	Data feasibility in tourism; Tourist spatial-temporal behavior analysis; Tourism recommendation	Global; High precision	High cost; Privacy concerns
	Mobile roaming data		Availability in less visited places; Allowing tracking tourist origins	Privacy concerns; Low precision of location
	Bluetooth data		Availability in crowed indoor scenes; Allowing unannounced tracking	Small-range coverage; Privacy concerns
	RFID data		Low cost; High precision; Availability in crowed indoor scenes	Small coverage range; Privacy concerns
	WIFI data		Availability in crowed indoor scenes; Allowing unannounced tracking	Small coverage range; Privacy concerns
	Meteorological data	Effect estimation of weather on tourism; Tourism recommendation	Containing weather factors	Difficulty in processing multi-format data
Operations	Web search data	Tourism demand prediction; Search engine optimization	Low cost; Reflecting public attention	Possible estimation biases
	Other transaction data	Tourist behavior analysis; Tourism marketing	Recording tourists' operations in tourism markets	Privacy concerns

Tabela 1: Comparação entre diferentes tipos de *Big Data* no turismo [12]

O autor em [12] identificou ainda três categorias principais de desafios associados à gestão de dados no setor do turismo, designadamente: a qualidade dos dados, os custos envolvidos na sua obtenção e as questões relacionadas com a privacidade. Estes aspetos encontram-se sintetizados na última coluna da Tabela 1.

Relativamente à qualidade dos dados, embora o setor do turismo beneficie de um elevado volume de informação, nem sempre esta se revela precisa ou fidedigna. Um exemplo paradigmático diz respeito aos dados textuais disponíveis online, cuja fiabilidade pode ser posta em causa, dado que algumas opiniões publicadas por visitantes podem ser falsas ou manipuladas [17].

No que se refere ao custo dos dados, sobretudo aqueles provenientes de dispositivos tecnológicos, o acesso a este tipo de informação implica frequentemente um investimento elevado. Este custo inclui não só a aquisição de sensores e equipamentos, como também a mobilização de voluntários para recolha e partilha dos dados [12].

Por fim, a privacidade constitui uma barreira significativa à partilha de dados no contexto turístico. As preocupações associadas à proteção da informação pessoal dificultam a colaboração entre os diversos intervenientes do setor, como turistas, agências de viagens online, unidades hoteleiras, entidades governamentais e organizações responsáveis por atrações turísticas. Esta limitação

compromete a capacidade de implementar uma gestão mais integrada, eficiente e baseada em dados [18].

Para além destas questões de natureza dos dados turísticos, importa considerar também os desafios técnicos inerentes ao tratamento e análise de grandes volumes de dados. A complexidade não reside apenas na obtenção ou partilha da informação, mas também na capacidade de processá-la de forma eficaz. O artigo em [19] destaca como principais obstáculos a tolerância a falhas, a escalabilidade, a qualidade dos dados e a heterogeneidade dos dados.

A tolerância a falhas é um requisito essencial nos sistemas de *big data*, dada a sua escala massiva e arquitetura distribuída. A fiabilidade e a disponibilidade dos serviços dependem fortemente da capacidade do sistema para continuar a operar mesmo perante falhas. Para garantir essa resiliência, são implementados mecanismos como a replicação de dados, a deteção e recuperação de falhas, e a utilização de redundância, assegurando a continuidade das operações [20].

Outro desafio crítico é a escalabilidade. Com o crescimento exponencial do volume de dados, torna-se imperativo garantir um processamento eficiente e ágil. Os sistemas tradicionais de gestão de bases de dados relacionais (RDBMS) revelam-se inadequados para lidar com a dimensão e complexidade dos dados gerados atualmente. Assim, os sistemas de *big data* devem permitir escalabilidade tanto vertical (aumento da capacidade de um único servidor) como horizontal (adição de servidores ao sistema). Importa ainda salientar que a escalabilidade não se refere apenas ao volume de dados, mas também à capacidade de processamento em tempo real ou quase em tempo real. Para responder a estas exigências, têm sido adotadas tecnologias como a computação em nuvem, sistemas de ficheiros distribuídos, bases de dados NoSQL e o modelo de processamento MapReduce [20].

A qualidade dos dados é igualmente um fator determinante. Devido à multiplicidade de fontes, formatos e origens, os dados podem apresentar ruído, redundâncias, inconsistências e erros, comprometendo a fiabilidade das análises. Adicionalmente, é comum que os dados estejam incompletos, desatualizados ou em formatos incompatíveis, o que dificulta a sua integração. O pré-processamento dos dados é, por isso, essencial para melhorar a sua qualidade, embora envolva técnicas complexas de limpeza e integração, que podem introduzir atrasos nos processos analíticos [21].

Por fim, a heterogeneidade dos dados representa um dos desafios mais exigentes. A informação disponível pode assumir formas estruturadas (como em bases de dados relacionais), semiestruturadas (como ficheiros JSON ou XML) ou não estruturadas (como textos livres, imagens ou vídeos). Esta diversidade requer sistemas capazes de integrar, interpretar e processar dados com diferentes formatos, estruturas e semânticas. A variabilidade na granularidade e na representação dos dados complica ainda mais a sua combinação e análise. Como resposta, tem-se vindo a recorrer a abordagens baseadas em *data lakes*, que permitem armazenar dados em bruto de forma centralizada, facilitando posteriormente a sua análise e integração [22].

Em suma, a aplicação de *big data* no setor do turismo revela um enorme potencial para melhorar a compreensão deste domínio, mas enfrenta ainda desafios significativos. A diversidade de fontes de dados desde conteúdos gerados por utilizadores a dispositivos e operações digitais oferece múltiplas oportunidades de análise, mas também levanta questões complexas relacionadas com a qualidade, os custos e a privacidade da informação. Para além disso, os obstáculos técnicos, como a necessidade de sistemas tolerantes a falhas, escaláveis e capazes de lidar com dados heterogéneos, exigem soluções tecnológicas robustas e adaptativas.

2.2 Arquitetura de Referência

As arquiteturas de referência para plataformas de *Big Data* desempenham um papel fundamental na normalização e na otimização do processamento e análise de grandes volumes de dados [23]. Estas arquiteturas fornecem um enquadramento geral que facilita tanto a conceção como a implementação de sistemas de *Big Data*, permitindo uma melhor compreensão global das suas funcionalidades típicas e orientando a seleção das tecnologias mais adequadas [24]. Neste contexto, foi realizada uma análise de diversas arquiteturas de referência, com base nos estudos apresentados em [25], [26], [27] e [24]. O principal objetivo desta análise é estabelecer diretrizes e identificar boas práticas para o desenvolvimento de um sistema robusto, escalável e eficiente, adaptado à plataforma SMARTDEST.

As arquiteturas de referência apresentadas em [25], [26] e [24] seguem abordagens semelhantes, destacando-se pela inclusão de componentes principais como a recolha de dados (em modo *batch* ou em *streaming*), o respetivo processamento (incluindo tarefas como limpeza, compressão, replicação e fusão) e o carregamento para diferentes bases de dados, consoante a fase do *pipeline* em execução.

Para além disso, os autores referem ainda a existência de componentes dedicados ao agendamento de tarefas, bem como a aplicação de técnicas de *machine learning* para fins de análise e visualização de dados.

Em contraste, a arquitetura proposta em [27] adota uma abordagem distinta, assente na computação *serverless* através do modelo FaaS (Function-as-a-Service). Esta arquitetura organiza-se em quatro planos lógicos: utilizador, controlo, execução e dados. Cada um com responsabilidades específicas, como a definição de tarefas, orquestração, processamento e armazenamento. Esta solução transfere a responsabilidade da gestão de infraestrutura para o fornecedor da plataforma, permitindo assim uma escalabilidade automática e uma utilização otimizada dos recursos, incluindo o *scaling* até zero nos períodos sem tarefas ativas.

Nesta secção, será realizada uma análise detalhada da arquitetura de referência desenvolvida pelo autor do artigo em [24]. A escolha desta arquitetura deve-se ao facto de, em comparação com as abordagens descritas em [25] e [26], ser a que abrange um maior número de casos de uso de plataformas de *Big Data*, incluindo exemplos de empresas como o Facebook, LinkedIn e Netflix.

Relativamente à arquitetura apresentada em [27], apesar das suas vantagens, foi descartada devido à adoção de uma abordagem *serverless*, e os custos consequentemente associados a essa.

A arquitetura descrita em [24] é apresentada na forma de um *pipeline*, no qual os dados fluem da esquerda para a direita. Esta arquitetura integra componentes essenciais que abrangem a aquisição, o processamento, a análise e a visualização dos dados, conforme ilustrado na Figura 1.

A arquitetura tem início no componente Fontes de Dados, como pode ser visto na primeira coluna da Figura 1, sendo este dividido em duas dimensões principais para a classificação das fontes de dados: mobilidade e estrutura. A dimensão de mobilidade distingue entre dados in situ (que permanecem na sua origem) e dados provenientes de streaming (um fluxo contínuo processado em tempo real). Já a dimensão estrutural classifica os dados em três categorias: estruturados, não estruturados e semiestruturados (uma categoria intermédia com alguma organização, mas maior flexibilidade, como, por exemplo, os documentos XML). Esta distinção é fundamental para determinar a forma como os dados são adquiridos e processados num sistema de *Big Data*.

A seguir, passamos à componente de Extração de Dados, localizada na segunda coluna da Figura 1, a qual é responsável pela entrada de dados no sistema. Os dados podem ser armazenados

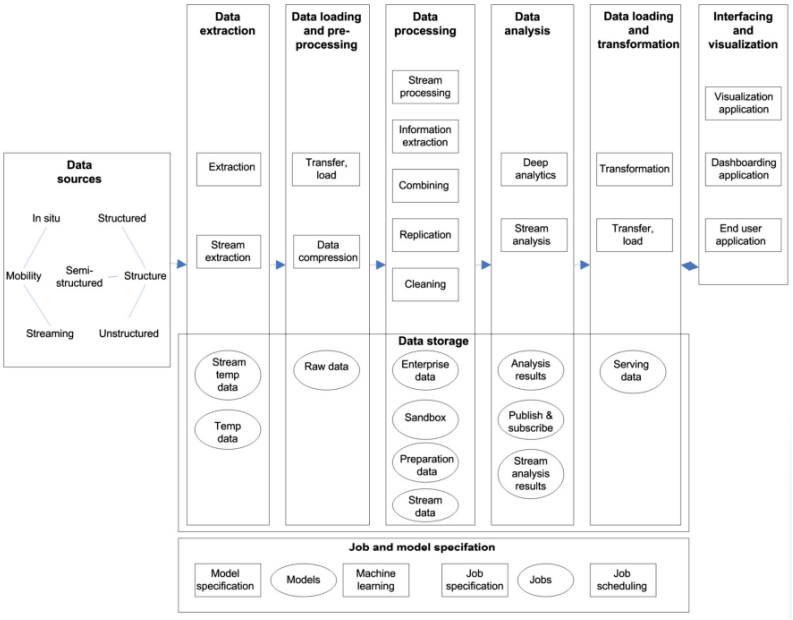

Figura 1: Arquitetura de referência [24]

temporariamente no *Temp Data Store* ou transferidos para o *raw data store*. O *Temp Data Store* é utilizado para armazenar dados de forma temporária durante o processamento ou transferência de informações dentro de um sistema. Ele serve como um local de armazenamento intermediário, onde os dados podem ser retidos antes de serem processados ou transferidos para um repositório definitivo. Por outro lado, o *raw data store* é destinado à conservação dos dados na sua forma original, sem processamento, possibilitando análises e extração de informações adicionais posteriormente.

Após os dados estarem armazenados, passamos para os componentes *pre processing* e *processing*, representados na terceira e quarta coluna da Figura 1. Estes são responsáveis por comprimir e converter diferentes tipos de dados para um formato único, armazenando-os numa área de *staging*. A agregação dos dados na área de *staging* tem como objetivo apresentar os dados brutos de forma adequada para análises mais aprofundadas, ou, alternativamente, transformar os dados para um formato apropriado para aplicações de visualização.

Depois de armazenados na área de *staging*, os dados serão processados pelos componentes *Data analysis*, *Data loading and transformation*, representados na quinta e sexta coluna da Figura 1, com o objetivo de armazenar os resultados num Data Warehouse, num formato compatível com as plataformas de visualização.

Por fim, uma vez armazenados no *data warehouse*, os dados serão acedidos pelo componente de *Interfacing and visualization* representado na ultima coluna da Figura 1, que os busca para

apresentar as informações analisadas de forma clara e acessível aos utilizadores, através de diversos dashboards.

A adequação desta arquitetura ao contexto do SMARTDEST torna-se evidente quando se analisa a forma como os seus módulos dão resposta à natureza heterogénea dos dados, característica intrínseca do domínio do turismo.

Adicionalmente, a inclusão de uma camada dedicada ao pré-processamento e ao armazenamento de dados brutos revela-se particularmente relevante no setor do turismo, onde a preservação da informação histórica é essencial para a análise de padrões e tendências ao longo do tempo. Ao manter os dados na sua forma original, antes de qualquer transformação, a arquitetura assegura a existência de uma fonte de verdade imutável, possibilitando reprocessamentos futuros sempre que necessário.

Em síntese, a análise da arquitetura de referência proposta em [24] demonstra que esta constitui uma base sólida e teoricamente bem fundamentada para servir de roteiro funcional à plataforma SMARTDEST. Esta conclusão é sustentada não apenas pela sua capacidade de responder aos requisitos específicos do domínio turístico, mas também pelos casos de uso bem-sucedidos em organizações de grande escala, tais como o Facebook, o LinkedIn e a Netflix.

2.3 Arquiteturas de Armazenamento de Dados

Arquiteturas de armazenamento de dados, especialmente no contexto de *big data*, referem-se a ferramentas e mecanismos desenvolvidos para armazenar volumes de dados em rápido crescimento, com alta velocidade e diversidade. Isto inclui os métodos de armazenamento (como armazenamento em blocos, ficheiros ou objetos), a disposição física e lógica dos sistemas (em ambientes locais, na nuvem ou híbridos) e a infraestrutura necessária para garantir desempenho, escalabilidade, fiabilidade e segurança [28].

2.3.1 Data Warehouse

Com o intuito de resolver o problema da dispersão de dados por múltiplos sistemas operacionais, como ERP, CRM e bases de dados transacionais, surgiram os *data warehouses* (DWs). Um *data warehouse* é, fundamentalmente, um repositório centralizado de dados, concebido para integrar informação proveniente de diversas fontes operacionais, convertendo-a para um formato uniforme e

consistente. Esta estrutura é otimizada tanto para o armazenamento de grandes volumes de dados como para o processamento de consultas complexas [29].

A adoção dos *data warehouses* generalizou-se na década de 1990, como resposta aos desafios associados ao acesso, integração e análise eficiente de dados dispersos e heterogéneos. Estes sistemas oferecem um ponto centralizado de acesso à informação, permitindo uma visão consolidada e detalhada da organização, com o objetivo de apoiar e melhorar o processo de tomada de decisão e os respetivos resultados.

Para que um *data warehouse* possa cumprir eficazmente o seu papel enquanto ferramenta de suporte à decisão, é necessário que obedeça a um conjunto de características estruturais fundamentais tais como [29] [30]:

- **Orientado por Assunto**: Os dados são organizados por temas específicos, reunindo toda a informação relevante sobre um determinado assunto.
- **Integrado**: Os dados são uniformizados, com nomes, medidas e formatos consistentes, mesmo que provenham de sistemas distintos.
- **Não Volátil**: A informação é estável e não se altera com cada novo processo operacional; permanece consistente ao longo do tempo.
- **Variável no Tempo**: Inclui tanto dados históricos como atuais, abrangendo um período alargado (tipicamente entre 5 a 10 anos), ao contrário dos dados operacionais que se limitam a intervalos temporais mais curtos.

Estas características estruturais constituem a base para o funcionamento eficaz de um *data warehouse*, assegurando a fiabilidade e relevância da informação disponibilizada para análise. No entanto, a utilidade de um *data warehouse* não depende apenas das suas propriedades conceptuais, mas também da forma como é concebido e implementado.

A criação de um *data warehouse* é um processo complexo que requer a integração de diversos componentes de *hardware* e software, concebidos para possibilitar a análise eficiente de grandes volumes de dados [29].

Importa salientar que o processo de *data warehousing* vai muito além da simples inserção de dados num repositório central. Envolve a definição de uma arquitetura robusta e a utilização de

ferramentas especializadas que possibilitem a recolha, transformação, armazenamento, integração e análise de dados oriundos de múltiplas fontes heterogéneas. [29].

A arquitetura define como os dados fluem dos sistemas operacionais para os ambientes analíticos, bem como a forma como são geridos e acedidos. A maioria das arquiteturas tradicionais de *data warehouses* estão organizadas em quatro camadas principais como é possível observar na Figura 2 [31] [32]:

- Data Source Layer : É nesta camada que os dados têm origem, geralmente provenientes de várias bases de dados operacionais, fluxos de dados externos ou outros sistemas transacionais. Estas fontes podem ser heterogéneas e apresentar diferentes formatos e níveis de latência.
- Data *Staging* Layer (ETL/ELT): Os dados provenientes dos sistemas de origem são extraídos, transformados e carregados (ETL), ou extraídos, carregados e depois transformados (ELT) para o armazém de dados. Esta camada trata da limpeza, integração e transformação dos dados, garantindo a sua consistência e qualidade.
- Data Storage Layer: Os dados limpos e integrados são armazenados num repositório central, frequentemente estruturado num formato multidimensional (esquema em estrela ou em floco de neve), de modo a facilitar consultas e análises eficientes. Esta camada está optimizada para cargas de trabalho analíticas predominantemente de leitura.
- Data Presentation/Access Layer: Esta camada fornece acesso aos dados armazenados para os utilizadores finais, normalmente através de ferramentas de relatórios, painéis de controlo, sistemas OLAP ou aplicações de mineração de dados.

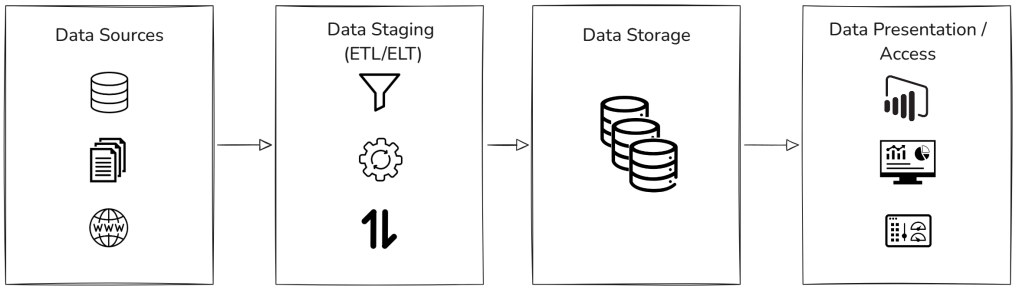

Figura 2: Arquitetura Data Warehouse [31]

Dada a importância da arquitetura na organização e no fluxo de dados dentro de um *data warehouse*, ao longo dos anos foram desenvolvidas diferentes abordagens arquiteturais para responder

a diversas necessidades organizacionais e operacionais. Entre as mais influentes encontram-se as abordagens clássicas propostas por Bill Inmon e Ralph Kimball, que oferecem perspetivas distintas sobre como estruturar e implementar um *data warehouse* [33].

A abordagem de Inmon, também conhecida como "Top-Down", defende a construção inicial de um Data Warehouse Empresarial (EDW) centralizado e altamente normalizado, geralmente na terceira forma normal (3NF), com o objetivo de garantir a consistência e integração dos dados em toda a organização. Bill Inmon, considerado o "pai do *data warehousing*", propõe que, após a criação deste armazém centralizado, sejam desenvolvidos *data marts* específicos por assunto, derivados diretamente do EDW. Esta estrutura assegura que todas as visões departamentais dos dados sejam coerentes e integradas. Entre as principais vantagens desta abordagem destacam-se a forte consistência e integridade dos dados em toda a organização, a facilitação da análise a nível empresarial graças a uma única fonte de verdade e a adequação a organizações com necessidades complexas de relatórios interdepartamentais. No entanto, esta metodologia apresenta desafios, como um tempo de implementação inicial mais prolongado, uma vez que o EDW deve ser construído antes da criação de qualquer *data mart*, e custos e complexidade mais elevados na fase inicial do projeto [34] [33].

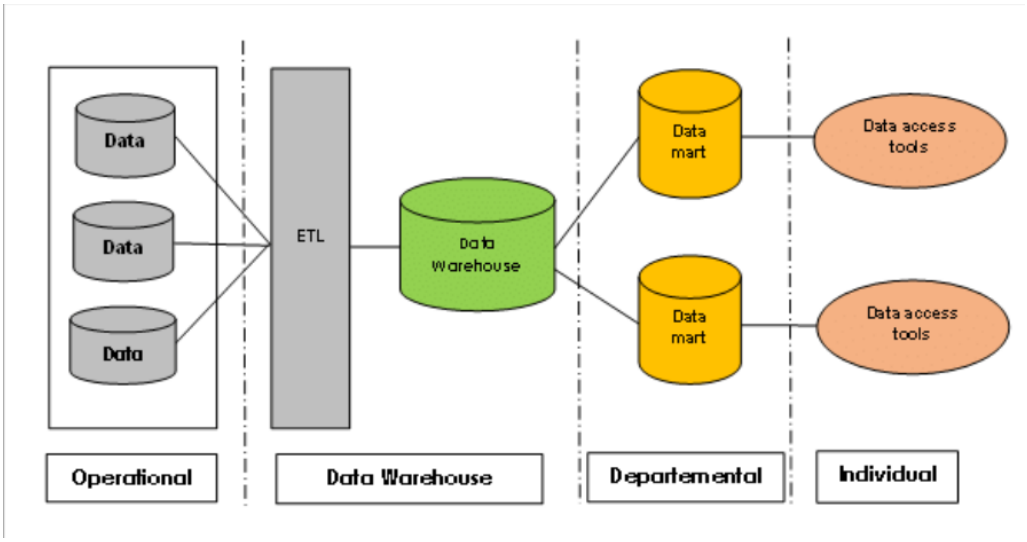

Figura 3: Arquitetura Data Warehouse Inmon [33]

A abordagem de Kimball, conhecida como "Bottom-Up", propõe começar pelo desenvolvimento de *data marts* departamentais ou focados em áreas temáticas específicas, cada um concebido segundo um modelo dimensional, como o esquema em estrela ou floco de neve. Estes *data marts* são

posteriormente integrados para formar um *data warehouse* completo. Esta metodologia privilegia a entrega rápida de valor ao negócio, ao permitir a implementação célere de soluções analíticas para áreas específicas. Entre as principais vantagens destacam-se o tempo reduzido para alcançar valor para as unidades de negócio, uma vez que os *data marts* podem ser implementados de forma independente e incremental, os custos iniciais mais baixos e a menor complexidade, bem como a flexibilidade para responder a necessidades e prioridades de negócio em constante evolução. Contudo, esta abordagem pode apresentar desafios, como o risco de criação de silos de dados ou inconsistências, caso os *data marts* não sejam bem integrados, e uma maior complexidade na análise a nível empresarial se os *data marts* não estiverem devidamente harmonizados.

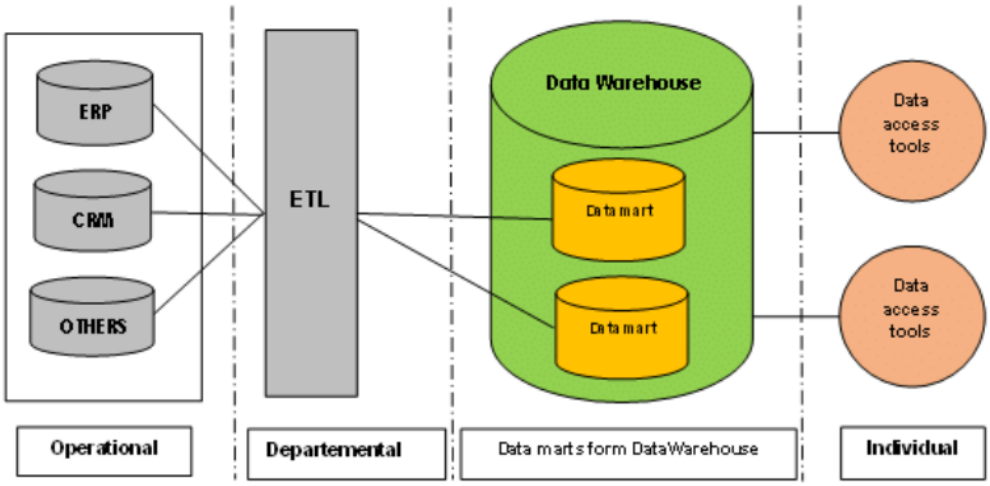

Figura 4: Arquitetura Data Warehouse Kimball [33]

2.3.2 Data Lake

Apesar da eficácia dos *data warehouses* na consolidação e análise de dados estruturados, a crescente diversidade, volume e velocidade dos dados gerados por fontes modernas como redes sociais, sensores IoT, logs de aplicações e dados multimédia evidenciou limitações nos modelos tradicionais. Neste contexto, emergiu o conceito de *data lake*, uma abordagem mais flexível e escalável, capaz de armazenar grandes volumes de dados em bruto, estruturados, semi-estruturados e não estruturados, sem a necessidade de transformação prévia.

Um *data lake* é um repositório centralizado concebido para armazenar grandes volumes de dados estruturados, semi-estruturados e não estruturados no seu formato bruto. Ao contrário dos

armazéns de dados tradicionais, que exigem esquemas predefinidos (*schema-on-write*), os *data lakes* utilizam uma abordagem de *schema-on-read*, permitindo maior flexibilidade e escalabilidade na gestão de diferentes tipos de dados e cargas de trabalho analíticas [35]. [36]

As principais características que distinguem um *data warehouse* de um *data lake* podem ser sintetizadas da seguinte forma [37] [38] [39]:

- **Ingestão e Armazenamento de Dados:** Os *data warehouses* exigem que os dados sejam previamente limpos, transformados e estruturados antes de serem carregados, o que representa um processo moroso e que limita a diversidade dos dados armazenáveis. Em contraste, os *data lakes* permitem a ingestão de dados em qualquer formato, facilitando o armazenamento de grandes volumes de dados brutos provenientes de múltiplas fontes.

- **Análise e Queries de Dados:** Os *data warehouses* são particularmente eficazes na execução de consultas analíticas complexas sobre dados estruturados, sendo amplamente utilizados em soluções tradicionais de Business Intelligence. Por outro lado, os *data lakes* são mais indicados para análises exploratórias, desenvolvimento de modelos de *machine learning* e contextos que envolvem dados com elevada variedade e volume.

- **Escalabilidade e Custo:** Em geral, os *data lakes* são mais escaláveis e apresentam custos inferiores, pois recorrem a *hardware* de baixo custo e armazenam dados em bruto. Já os *data warehouses*, embora robustos em termos de desempenho, podem tornar-se dispendiosos à medida que o volume de dados aumenta, devido às exigências de armazenamento e processamento especializados.

- **Governação e Gestão de Dados:** Os *data warehouses* beneficiam de ferramentas e processos de governação de dados mais consolidados, com ênfase na segurança, conformidade e controlo de qualidade. Nos *data lakes*, a eficácia depende fortemente da gestão adequada dos metadados, caso contrário, podem evoluir para *data swamps*, nos quais a informação se torna difícil de localizar e utilizar de forma eficiente.

- **Flexibilidade e Casos de Utilização:** Os *data lakes* oferecem uma maior flexibilidade e adaptabilidade a novos tipos de dados e requisitos analíticos, sendo particularmente valorizados em contextos de *big data* e inteligência artificial. Em contrapartida, os *data warehouses* continuam a ser preferidos em cenários onde a consistência, qualidade e conformidade dos dados são fatores críticos.

Uma análise mais detalhadas das diferenças entre *data warehouses* e *data lakes* é aprensentada na tabela 2

Parameters	Data Warehouse	Data Lake
Data	Data warehouse focuses only on business processes	Data lakes store everything
Processing	Highly processed data	Data are mainly unprocessed
Type of Data	They are mostly in the tabular form and structure	They can be unstructured, semi-structured, or structured
Task	Optimized for data retrieval	Share data stewardship
Agility	Less agile and has fixed configuration compared with data lakes	Highly agile and can configure and reconfigure as needed
Users	Widely used by business professionals and business analysts	Data lakes are used by data scientists, data developers, and business analysts
Storage	Expensive storage that gives fast response times is used	Data lakes are designed for low-cost storage
Security	Allows better control of the data	Offers less control
Schema	Schema on writing (predefined schemas)	Schema on reading (no predefined schemas)
Data Processing	Time-consuming to introduce new content	Helps with fast ingestion of new data
Data Granularity	Data at the summary or aggregated level of detail	Data at a low level of detail or granularity
Tools	Mostly commercial tools	Can use open-source tools such as Hadoop or Map Reduce

Tabela 2: Diferenças entre *data warehouses* e *data lakes*. [39]

Tal como o *data warehouse*, o *data lake* também tem a sua própria arquitetura e a compreensão desta é fundamental para perceber de que forma estes sistemas conseguem oferecer uma elevada escalabilidade, flexibilidade e capacidade de integração de dados heterogéneos. A estrutura arquitetónica de um *data lake* é tipicamente composta por várias camadas funcionais que abrangem a ingestão, armazenamento, gestão de metadados, segurança e acesso, bem como mecanismos de processamento e análise como é possível observar na Figura 5

As principais camadas e componentes da arquitetura de um *data lake* incluem [39]:

- **Raw data layer**: Tem como principal finalidade a ingestão de dados brutos da forma mais rápida e eficiente possível, sem qualquer tipo de transformação. Nesta fase, é crucial garantir a integridade dos dados originais, pelo que não é permitida a substituição (*overriding*) nem a duplicação de versões dos mesmos, sendo comum a utilização de mecanismos de arquivo (*archive*) que possibilitam a reversão para estados anteriores.

- **Standardized data layer:** Componente opcional na maioria das implementações de *data lake*, sendo particularmente recomendada em contextos onde se antecipa um crescimento acelerado da arquitetura. O seu principal objetivo consiste em otimizar o desempenho da transferência de dados da *Raw Data Layer* para a *Cleansed Layer*, através da conversão dos dados do seu

formato nativo para um formato padronizado que facilite os processos subsequentes de limpeza e transformação.

- **Cleansed layer or curated layer**: Constitui uma das componentes mais complexas da arquitetura de um *data lake*, sendo responsável por transformar os dados brutos em conjuntos de dados consumíveis, através de processos de limpeza, transformação, desnormalização e consolidação de diferentes fontes; os dados são então organizados por propósito, tipo e estrutura de ficheiro, sendo esta, geralmente, a camada à qual os utilizadores finais têm acesso.

- **Application layer**: Corresponde à versão refinada da Cleansed Layer, enriquecida com a lógica de negócio necessária, sendo a principal fonte de dados para aplicações analíticas e modelos de *machine learning*, mantendo a mesma estrutura de dados da camada anterior.

- **Sandbox data layer:**:Constitui uma componente opcional da arquitetura, destinada a analistas e cientistas de dados para a realização de experiências exploratórias e a identificação de padrões ou correlações, sendo também o espaço adequado para o enriquecimento dos dados com fontes externas.

- **Security**: Apesar de não serem geralmente expostos a um público alargado, os *data lakes* requerem mecanismos de segurança robustos desde as fases iniciais da sua concepção, uma vez que, ao contrário das bases de dados relacionais, não dispõem de sistemas de segurança nativos tão desenvolvidos.

- **Governance**: A monitorização e o registo de operações são essenciais para assegurar a rastreabilidade e a conformidade durante os processos analíticos.

- **Metadata**: Correspondem à informação descritiva sobre os dados armazenados, facilitando a sua catalogação, compreensão e utilização eficaz por parte dos utilizadores.

- **Stewardship**: Dependendo da escala do sistema, pode ser necessário atribuir funções específicas de gestão de dados ou recorrer a soluções baseadas em metadados para distribuir essa responsabilidade pelos utilizadores.

- **Master Data**: São fundamentais para garantir o fornecimento de dados prontos a utilizar, podendo estar armazenados no próprio *data lake* ou ser referenciados em processos ELT.

- **Archive**:Os *data lakes* podem incluir dados históricos oriundos de *data warehouses*, prevenindo problemas de desempenho e sobrecarga de armazenamento.

- **Offload**: Permite aliviar cargas intensivas de processamento (como ETL) dos sistemas relacionais, transferindo-as para o *data lake*.
- **Orchestration and ELT processes**: A movimentação de dados entre camadas requer ferramentas de orquestração ou recursos dedicados que garantam a automatização e eficiência do fluxo de dados.

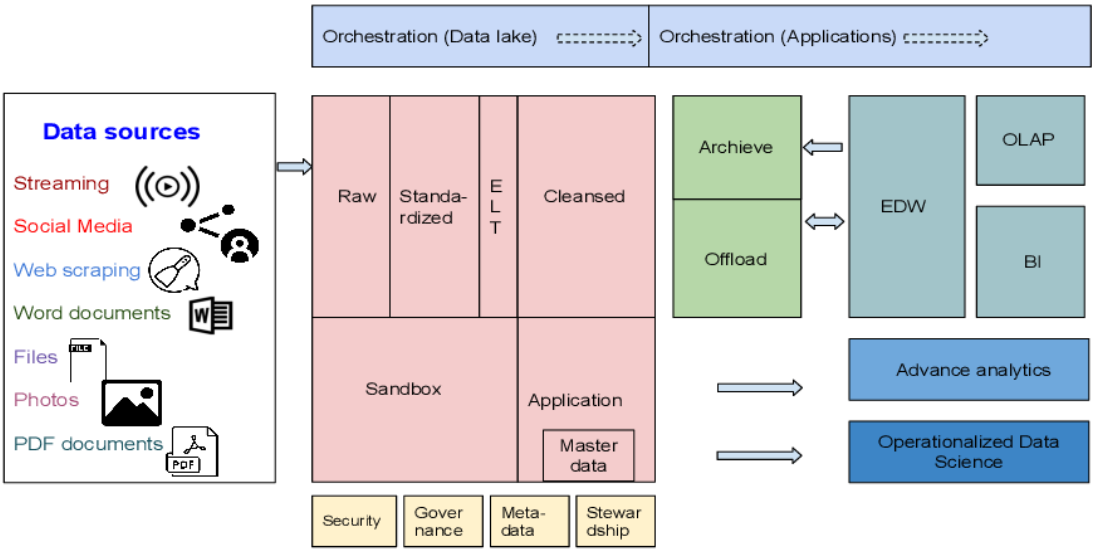

Figura 5: Arquitetura Data Lake [39]

Neste contexto, é fundamental compreender quais são os propósitos principais que tornam os *data lakes* uma solução cada vez mais relevante nas organizações modernas.

O artigo em [40] ilustra na Figura 6 os três principais propósitos dos *data lakes* em empresas, conforme revelado pelas entrevistas com especialistas em Business Intelligence (BI).

Um dos principais propósitos dos *data lakes* no contexto organizacional reside na sua utilização como áreas de *staging* ou como fontes primárias para *data warehouses*. A maioria dos estudos e testemunhos de especialistas enfatiza a importância dos *data lakes* enquanto componentes intermediários na arquitetura de dados, desempenhando um papel essencial nos processos de integração e transformação.

A principal mais-valia desta abordagem reside na capacidade dos *data lakes* em reter os dados no seu formato original, sem a aplicação imediata de transformações, possibilitando tanto a sua utilização para necessidades atuais bem definidas como para usos futuros ainda desconhecidos. Além disso, os *data lakes* podem contribuir para a redução das exigências de armazenamento nos

data warehouses, ao mesmo tempo que disponibilizam funcionalidades adicionais de exploração e manipulação de dados, indo além do mero armazenamento de grandes volumes de informação multiestruturada.

Um outro propósito dos *data lakes* prende-se com a sua função enquanto plataformas de experimentação para cientistas e analistas de dados. Estes utilizadores, considerados como "utilizadores avançados"dos sistemas de dados, beneficiam significativamente da liberdade oferecida pelos *data lakes* para realizar análises exploratórias e desenvolver modelos analíticos ou preditivos. A possibilidade de aceder a dados brutos, livres de transformações prévias, é particularmente vantajosa, uma vez que garante a integridade e a preservação de características dos dados que podem ser analiticamente relevantes, como valores em falta ou *outliers*. Este ambiente de experimentação designado como *sandbox* permite a realização de testes, exploração de padrões, desenvolvimento de algoritmos e, posteriormente, a transferência dos resultados mais promissores para os *data warehouses*, onde podem ser integrados em relatórios formais ou sistemas de apoio à decisão.

Por fim, um último propósito identificado para os *data lakes* consiste na sua utilização como fonte direta para soluções de *Business Intelligence de autoatendimento (self-service BI)*, em que os utilizadores podem construir novos relatórios diretamente sobre o *data lake*. Nesta configuração, as ferramentas de *self-service BI* são apoiadas por uma camada semântica que medeia o acesso entre o *data lake* e as ferramentas analíticas.

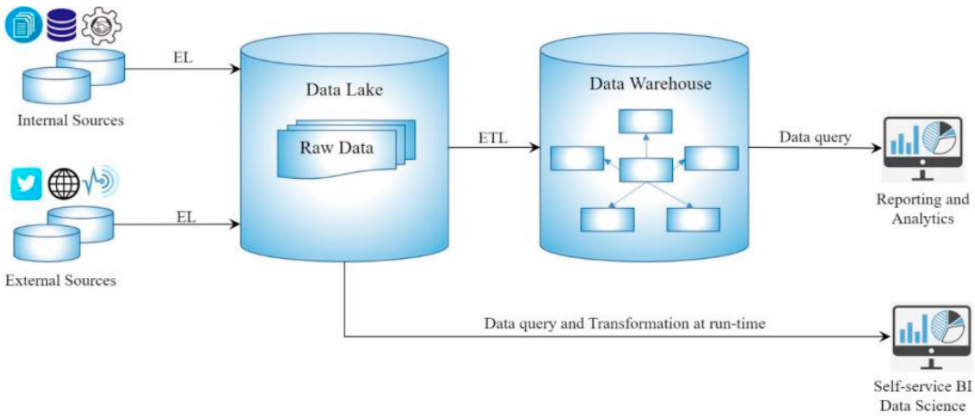

Figura 6: Proposta Data Lake [40]

2.4 Análise Comparativa de Soluções ETL

O *ETL* (Extract, Transform, Load), é um processo fundamental em projetos de integração de dados e análise de informação. Este conceito refere-se ao fluxo que consiste, primeiro, na extração de dados de diversas fontes, como bases de dados, ficheiros ou sistemas externos; em seguida, na transformação desses dados para um formato consistente e estruturado, de acordo com as necessidades do sistema de destino ou das análises a realizar; e, por fim, no carregamento desses dados processados para um ambiente de armazenamento, como um *data warehouse* ou um *data lake*.

A importância do *ETL* reside na sua capacidade de consolidar informações dispersas, garantir a qualidade e a consistência dos dados [41].

Este capítulo tem como objetivo realizar uma análise comparativa de algumas soluções *ETL* disponíveis no mercado, proporcionando uma visão crítica sobre as suas características, capacidades e limitações, com o objetivo de apoiar a seleção da ferramenta mais adequada para a reestruturação da plataforma SMARTDEST.

As soluções selecionadas para esta análise foram o Apache Airflow, o Apache NiFi e o AWS Glue. A escolha destas ferramentas fundamenta-se na sua relevância no domínio da orquestração e automação de fluxos de dados, conforme evidenciado em [42] [43].

Os critérios de avaliação utilizados para comparar as diferentes soluções basear-se-ão nos métodos descritos em [44]. Estes incluem a **eficácia**, que avalia a qualidade dos resultados de um sistema *ETL*, garantindo que os dados armazenados sejam completos, consistentes, resilientes a falhas e fáceis de manter, e a **eficiência**, que foca na otimização do desempenho, através de uma execução rápida, do uso do paralelismo para acelerar tarefas e da redução da sobrecarga de recursos nos sistemas envolvidos. Adicionalmente, será considerada a adequação das soluções para a aplicação a dados turísticos e no contexto da plataforma SMARTDEST.

2.4.1 Apache Airflow

O Apache Airflow é uma plataforma *open-source* concebida especificamente para a orquestração de fluxos de trabalho de dados complexos. Esta ferramenta permite aos utilizadores criar, agendar e monitorizar *workflows* de forma programática, o que a torna uma opção amplamente adotada

por equipas de engenharia de dados responsáveis pela gestão de *pipelines* automatizados em larga escala [45].

O Airflow é frequentemente utilizado na automatização de *pipelines* de ETL, envolvendo tarefas como a extração de dados a partir de APIs, a sua transformação através de linguagens como Python ou SQL, e o subsequente carregamento para sistemas de armazenamento na cloud ou *data warehouses*, tais como AWS S3, Redshift ou Snowflake [46].

A arquitetura do Apache Airflow baseia-se no conceito de Directed Acyclic Graphs (*Dags*), em que cada DAG representa um fluxo de trabalho composto por tarefas interdependentes, com uma ordem de execução bem definida. Esta arquitetura integra diversos componentes que interagem entre si, como é possivel visualizar na Figura 7 sendo estes [46]:

- Scheduler: Define a ordem e o momento de execução das tarefas dentro dos *Dags*.
- Executor: Responsável pela execução das tarefas, com suporte a vários sistemas de *backend* para garantir escalabilidade.
- Web Server: Fornece uma interface de utilizador para monitorizar, gerir e visualizar fluxos de trabalho.
- Metadata Database: Armazena informações sobre os *Dags*, os estados das tarefas e o histórico de execuções.

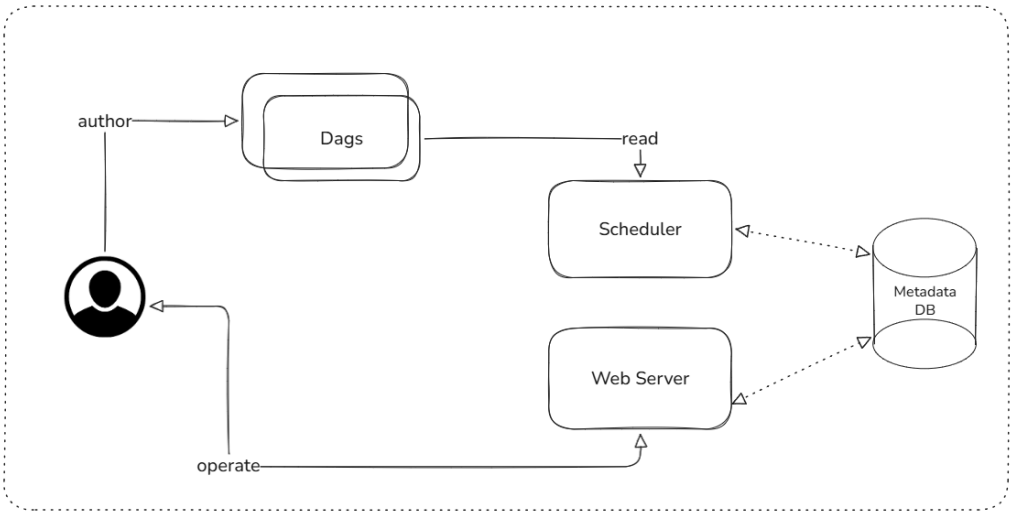

Figura 7: Principais componentes do Apache Airflow [47]

Os fluxos de trabalho no Airflow são definidos em Python, proporcionando aos utilizadores uma elevada flexibilidade na construção de *pipelines* dinâmicos e personalizados. Cada DAG é composto por uma sequência de tarefas, que podem englobar processos de extração, transformação e carregamento de dados. O componente *scheduler* ativa as tarefas com base em horários predefinidos ou em resposta a eventos externos, enquanto o *executor* trata da execução das mesmas, assegurando o cumprimento das dependências e a reexecução automática em caso de falhas [46].

O Apache Airflow apresenta diversas vantagens que consolidam a sua posição como uma ferramenta poderosa na orquestração de *workflows* de dados. Uma das principais é a sua flexibilidade, uma vez que os fluxos de trabalho são definidos através de código Python. Esta característica permite a criação de *pipelines* altamente personalizados e adaptáveis às necessidades específicas de cada projeto. Adicionalmente, o Airflow distingue-se pela sua escalabilidade, oferecendo suporte a execuções distribuídas e integração com ferramentas nativas de *cloud computing*, o que o torna especialmente adequado para ambientes de dados de grande dimensão.

Outra vantagem significativa reside na sua capacidade de monitorização e visibilidade que proporciona. Através de uma interface web intuitiva e abrangente, é possível acompanhar a execução dos *workflows*, gerir falhas e visualizar a estrutura de dependências entre tarefas, facilitando assim a manutenção e a operacionalização dos processos. Destaca-se ainda a extensibilidade da ferramenta, com um ecossistema robusto de operadores e *plugins* que possibilitam a integração com múltiplas fontes de dados, serviços em nuvem e ferramentas de *machine learning*, ampliando consideravelmente as possibilidades de aplicação.

Contudo, a adoção do Apache Airflow implica considerar algumas limitações. Uma das principais diz respeito à complexidade associada à sua configuração e manutenção, o que pode constituir um entrave para equipas com recursos técnicos limitados ou para contextos com fluxos de trabalho mais simples, dado exigir conhecimentos em Python e na gestão de infraestrutura. Adicionalmente, o Airflow foi concebido principalmente para o processamento em *batch*, apresentando, por isso, maiores latências e tornando-se menos adequado para cenários que requerem processamento em tempo real, a menos que sejam realizadas adaptações específicas. Por fim, a curva de aprendizagem é relativamente acentuada, pois a flexibilidade e a robustez da ferramenta requerem um nível de domínio técnico superior em comparação com soluções ETL mais simplificadas ou plataformas *no-code* [45] [48].

2.4.2 Apache Nifi

O Apache NiFi é uma ferramenta *open-source* de ETL (Extract, Transform, Load), concebida para automatizar e gerir o fluxo de dados entre sistemas heterogéneos. Desenvolvido inicialmente pela National Security Agency (NSA) dos Estados Unidos e subsequentemente disponibilizado à Apache Software Foundation, o NiFi foi projectado para responder a necessidades complexas de processamento de dados em tempo real, com especial ênfase na escalabilidade, flexibilidade e facilidade de utilização [49].

Esta plataforma é amplamente utilizada em contextos que requerem a movimentação e transformação automatizada de dados em tempo real entre diferentes sistemas. A sua arquitetura adaptável torna-a particularmente relevante em sectores como o da saúde, onde viabiliza a integração e análise de grandes volumes de dados provenientes da monitorização de pacientes e de laboratórios, promovendo práticas de medicina de precisão e suporte à decisão em tempo real [50].

O Apache NiFi adopta um paradigma de programação orientado a fluxos. A sua arquitectura, representada na Figura 8, assenta numa estrutura central composta pelos seguintes componentes fundamentais que possibilitam a criação, gestão e monitorização de *pipelines* de dados de forma eficiente e escalável [49]:

- FlowFile: Unidade básica de dados que circula no sistema, contendo tanto o conteúdo dos dados como os seus respetivos atributos.
- Processor: Blocos modulares responsáveis pela execução de tarefas como ingestão, transformação, encaminhamento de dados.
- Controller Services: Serviços partilhados que fornecem recursos aos Processors, como ligações a bases de dados ou caches distribuídas.
- Flow Controller: Responsável por orquestrar a movimentação dos FlowFiles entre os Processors, garantindo que os dados são processados conforme o fluxo definido.
- Provenance Repository: Regista a origem e o histórico de cada FlowFile, permitindo monitorização, auditoria e resolução de problemas.
- User Interface: Plataforma web baseada em arrastar e largar, que permite aos utilizadores desenhar, configurar e monitorizar visualmente os fluxos de dados.

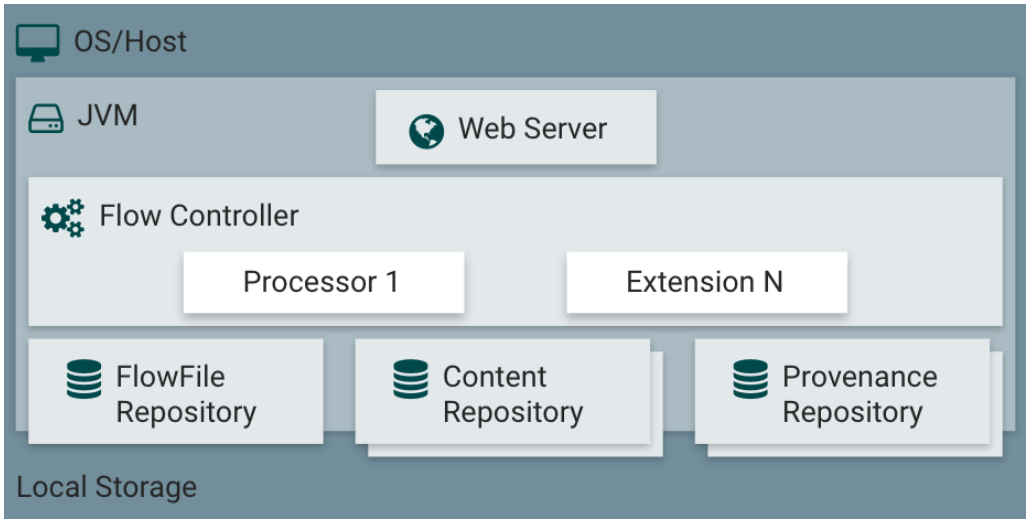

Figura 8: Principais Componentes Apache Nifi [51]

A configuração de um fluxo de trabalho no Apache NiFi é realizada através da ligação de Processors na User Interface. Cada Processor pode ser configurado para executar tarefas específicas, tais como a leitura de dados de diferentes fontes (por exemplo, bases de dados, sistemas de ficheiros, APIs), a transformação desses dados (incluindo filtragem, enriquecimento ou conversão de formatos) e a sua posterior escrita em destinos diversos (como outras bases de dados, sistemas de armazenamento em nuvem ou plataformas de análise). O Flow Controller gere o encaminhamento dos FlowFiles ao longo do fluxo, aplicando mecanismos de controlo de pressão (*backpressure*) e definição de prioridades, assegurando a estabilidade do sistema e a eficiência do processamento.

Entre as principais vantagens do NiFi destaca-se a sua interface gráfica intuitiva, baseada em operações de *drag-and-drop*, que facilita a criação, configuração e monitorização de fluxos de dados mesmo por utilizadores com conhecimentos técnicos limitados. Esta abordagem visual contribui significativamente para a agilidade no desenvolvimento de *pipelines* e para a redução do tempo de implementação. A plataforma oferece ainda suporte nativo ao processamento em tempo real, o que a torna particularmente útil em contextos de monitorização contínua e análises imediatas. Adicionalmente, o sistema de *data provenance* integrado permite retirar detalhadamente a origem, percurso e transformação dos dados, o que é essencial para auditoria e a rastreabilidade [52].

Contudo, o Apache NiFi apresenta algumas limitações que devem ser consideradas na sua adopção. Em cenários que exigem transformações complexas, cálculos intensivos ou processamento distribuído em larga escala, o seu desempenho pode ser inferior ao de outras plataformas mais especializadas, como o Apache Spark. Além disso, apesar da facilidade de utilização em casos

simples, a complexidade dos fluxos pode aumentar substancialmente à medida que se multiplicam os componentes, ligações e dependências, tornando a manutenção e a escalabilidade lógica mais desafiantes [53].

2.4.3 AWS Glue

O AWS Glue é um serviço *serverless* de integração de dados totalmente gerido pela Amazon Web Services (AWS), que simplifica o processo de ETL de dados para fins de análise, *machine learning* e criação de *data lakes*. Lançado inicialmente em 2017 como um serviço de ETL, o AWS Glue evoluiu para oferecer um conjunto mais abrangente de funcionalidades de integração de dados, servindo não apenas engenheiros de dados, mas também especialistas em ETL e cientistas de dados [54].

Este serviço é amplamente utilizado para automatizar de ETL e é particularmente valorizado por organizações que pretendem desenvolver *data lakes* e plataformas de análise de dados escaláveis na nuvem. Uma das suas principais características é a utilização de DynamicFrames, uma estrutura de dados concebida para lidar com informação desorganizada, sem esquema fixo e semi-estruturada , facilitando a manipulação e transformação de conjuntos de dados complexos.

O AWS Glue adota um paradigma de programação orientado a tarefas ETL *serverless*. A sua arquitetura, representada na Figura 9, baseia-se numa infraestrutura gerida que integra componentes fundamentais como :

- AWS Glue Data Catalog: Um repositório central de metadados compatível com o Hive Metastore, que armazena definições de tabelas, metadados de tarefas e informações de esquemas. Este catálogo facilita a descoberta, gestão e consulta de dados.

- Glue Crawlers: Processos automáticos que analisam as fontes de dados para inferir os esquemas e atualizar o catálogo de dados, garantindo que os metadados se mantêm sempre atualizados.

- ETL Jobs: *Scripts serverless* em Spark ou Python que realizam a extração, transformação e carregamento de dados. Estas tarefas podem ser criadas visualmente através do Glue Studio ou desenvolvidas em código.

- Glue Studio: Interface visual que permite desenhar, executar e monitorizar tarefas ETL, tornando o processo acessível a utilizadores com diferentes níveis de conhecimentos técnicos.

- Resource Manager: Responsável pela gestão dos recursos de computação utilizados pelas tarefas do Glue, assegurando tempos de arranque rápidos e capacidades de escalamento automático para responder eficientemente a diferentes volumes de trabalho.

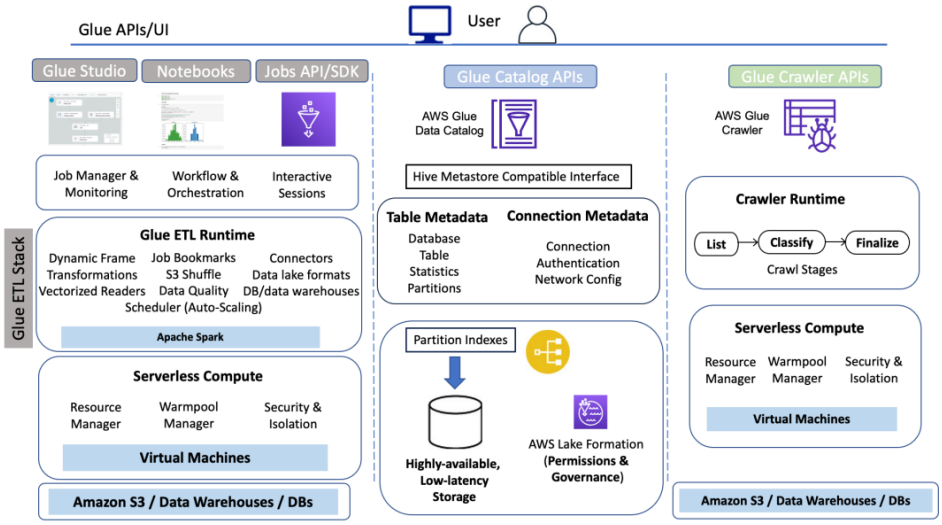

Figura 9: Principais Componentes AWS Glue [54]

A configuração de um fluxo de trabalho no AWS Glue é efetuada através do Glue Studio, uma interface gráfica que permite criar, configurar e orquestrar tarefas de ETL de forma visual. Cada tarefa, ou *job*, pode ser personalizada para executar operações específicas, como a leitura de dados a partir de diversas fontes (por exemplo, bases de dados relacionais, *data lakes* no Amazon S3 ou streams de dados), a transformação desses dados (incluindo limpeza, junção, enriquecimento ou conversão de formatos) e a sua gravação em destinos variados. O AWS Glue gere automaticamente a execução e o escalonamento destes *jobs* em ambientes *serverless*, assegurando a eficiência do processamento e a integração com outros serviços da AWS, como o Data Catalog, para gestão de metadados e descoberta de esquemas.

De acordo com [55] e [54], uma das principais vantagens do AWS Glue é a sua arquitetura *serverless*, que elimina a necessidade de gestão de infraestruturas, oferecendo escalabilidade automática e um modelo de pagamento baseado na utilização efetiva dos recursos. Esta abordagem permite reduzir significativamente os custos operacionais e simplificar a administração do sistema. Além disso, o serviço é altamente escalável, conseguindo processar grandes volumes de dados e múltiplas cargas de trabalho em simultâneo, o que o torna particularmente adequado para integrações de

dados à escala empresarial. Outra vantagem relevante é o suporte nativo a dados semiestruturados. A utilização de DynamicFrames e transformações integradas facilita o processamento de conjuntos de dados complexos e sem esquema fixo, agilizando o desenvolvimento de *pipelines* ETL.

No entanto, o AWS Glue também apresenta algumas limitações. Apesar de abranger a maioria dos cenários comuns de transformação de dados, operações mais complexas ou personalizadas podem exigir competências avançadas em Spark ou Python, o que pode aumentar o tempo de desenvolvimento e complexidade do projeto. Por fim, embora a *arquitetura serverless* possa ser mais económica em muitos casos, uma má optimização dos trabalhos ou a utilização frequente de crawlers pode originar custos adicionais inesperados.

2.4.4 Comparação

No contexto da avaliação de sistemas ETL, o Apache Airflow, o Apache NiFi e o AWS Glue apresentam abordagens distintas quanto à eficácia e à eficiência.

Em termos de eficácia, o Apache Airflow destaca-se pelo seu forte controlo de orquestração e pela capacidade de monitorizar fluxos de trabalho complexos configurados através dos *Dags*, assegurando a completude e consistência dos dados, embora requeira maior esforço na manutenção e gestão de falhas. O Apache NiFi, por outro lado, oferece uma interface visual intuitiva e capacidades nativas de tolerância a falhas, facilitando a manutenção e garantindo resiliência na movimentação de dados, sendo esta uma alternativa particularmente adequadas para equipas com recursos técnicos limitados, permitindo uma utilização mais simples e intuitiva sem comprometer a qualidade dos resultados [56]. O AWS Glue, sendo um serviço gerido na nuvem, proporciona elevada fiabilidade e integração nativa com outros serviços da AWS, garantindo consistência e menor necessidade de intervenção manual.

No que diz respeito à eficiência, o Airflow permite otimização através da execução paralela de tarefas e de uma arquitetura modular, embora possa sofrer de sobrecarga em ambientes com muitos *Dags* e dependências complexas [48].O Apache NiFi destaca-se pela sua elevada eficácia em cenários de processamento e manipulação de dados em tempo real, oferecendo mecanismos robustos de tolerância a falhas através da retenção do fluxo de dados e de tentativas automáticas de recuperação. A sua arquitetura orientada a fluxos de dados permite uma gestão eficiente do movimento e transformação da informação, suportando paralelismo e distribuição de tarefas de forma nativa [53]. Já o AWS Glue beneficia de uma elevada escalabilidade automática e de execu-

ção distribuída, permitindo um desempenho eficiente com menor gestão de infraestrutura, sendo particularmente vantajoso quando a redução da sobrecarga operacional é crítica.

No que concerne à adequação destas soluções ao contexto da plataforma SMARTDEST, com base no estudo teórico realizado no decorrer deste capítulo, o Apache Airflow revelou-se particularmente apropriado, destacando-se pela sua elevada flexibilidade e pela capacidade de integração com uma ampla variedade de operadores. Esta característica permite uma adaptação eficiente a diferentes requisitos e cenários, algo essencial numa plataforma que processa dados provenientes de múltiplas fontes.

O Apache NiFi também se mostrou adequado à plataforma SMARTDEST, oferecendo suporte tanto a processamento *batch* como a processamento em tempo real. Ao contrário do Airflow, que foi concebido principalmente para tarefas em *batch*, o NiFi proporciona esta maior versatilidade. Adicionalmente, disponibiliza uma interface gráfica intuitiva para a definição de fluxos de dados, facilitando significativamente a configuração e monitorização dos processos. No entanto, uma vez que o Apache Airflow segue uma abordagem mais programática, acaba por se revelar uma ferramenta mais flexível, permitindo uma personalização e integração mais extensiva de operadores e fluxos de trabalho.

Relativamente ao AWS Glue, embora se trate de uma solução escalável e totalmente gerida na nuvem, com suporte nativo para execuções em *batch*, a sua adoção não é considerada adequada para a arquitetura da SMARTDEST, uma vez que o projeto privilegia exclusivamente a utilização de ferramentas open source.

Concluindo, o setor do turismo tem vindo a assistir a uma crescente valorização do *big data*, que proporciona benefícios significativos, como a identificação de padrões e tendências que sustentam a tomada de decisões estratégicas. No entanto, a utilização de grandes volumes de dados neste contexto levanta diversos desafios. Entre os mais relevantes destacam-se questões relacionadas com a qualidade dos dados frequentemente afetada por imprecisões ou manipulações , os custos associados à aquisição e à manutenção de infraestruturas tecnológicas adequadas, e preocupações com a privacidade e segurança da informação, que dificultam a partilha entre os vários agentes do setor. Acrescem ainda desafios técnicos como a necessidade de sistemas escaláveis, tolerantes a falhas e capazes de lidar com a heterogeneidade dos dados, que surgem em múltiplos formatos, estruturas e fontes.

Para enfrentar estas exigências, torna-se essencial a adoção de arquiteturas de referência robustas e soluções de armazenamento adequadas, como os *data warehouses*, orientados a dados estruturados e analíticos, e os *data lakes*, mais adequados para dados brutos, semiestruturados ou não estruturados. Paralelamente, a escolha de uma ferramenta e de um processo de ETL (Extract, Transform, Load) apropriados assume um papel fundamental na consolidação, transformação e preparação dos dados. A análise realizada neste trabalho evidenciou que diferentes ferramentas de ETL apresentam características específicas em termos de desempenho, facilidade de uso, escalabilidade e adequação a diferentes tipos de dados e complexidades de carregamento, sendo a sua seleção dependente dos requisitos técnicos e operacionais de cada cenário.

3 Metodologia

Este capítulo descreve as estratégias, opções de design e procedimentos adotados durante o desenvolvimento da reestruturação arquitetural da plataforma SMARTDEST.

3.1 Análise da Plataforma SMARTDEST

Inicialmente foi realizada uma análise da arquitetura da plataforma SMARTDEST [9].

Esta arquitetura pode ser observada na Figura 10, composta por três principais componentes que interagem entre si: Recolha e Armazenamento de Dados, REST API, e *Web Application* para a Visualização e Extração de Dados [9].

O componente de Recolha e Armazenamento de dados é um componente composto por vários módulos, sendo que cada módulo está associado a uma fonte de dados específica e tem um horário programado que define quando deve correr. Estes módulos têm como função extrair dados de fontes externas, transformá-los para o formato exigido pela base de dados MySQL e, finalmente, armazená-los.

Inicialmente, os dados são armazenados numa base de dados de *Staging* que os guarda temporariamente ao longo do dia com o objetivo de realizar um pré processamento para prevenir duplicação e realizar uma a agregação de dados com granularidades temporais diferentes. No inicio de cada dia os dados já processados são transferidos para um *Data Warehouse* que é a base de dados principal e seguidamente é limpo os dados guardados dentro da base de dados de *Staging*.

Com os dados armazenados no *Data Warehouse*, a REST API, desenvolvida em Express (Node.js), possibilita o acesso a entidades externas, bem como à própria plataforma web. Esta plataforma, foi desenvolvida com tecnologias como Vue.js, Vue-pivottable e Plotly.

Dentro desta arquitetura, conseguimos identificar diversas vantagens. Uma delas é a sua modularidade, que facilita a adaptação e expansão do sistema para responder a novas necessidades. Além disso, em caso de falha de um componente, o sistema mantém-se funcional, evitando uma paragem total. Outra vantagem significativa é o processo de armazenamento em duas bases de dados (*Staging e Data Warehouse*), que contribui para a melhoria da qualidade dos dados e aumento da organização das informações.

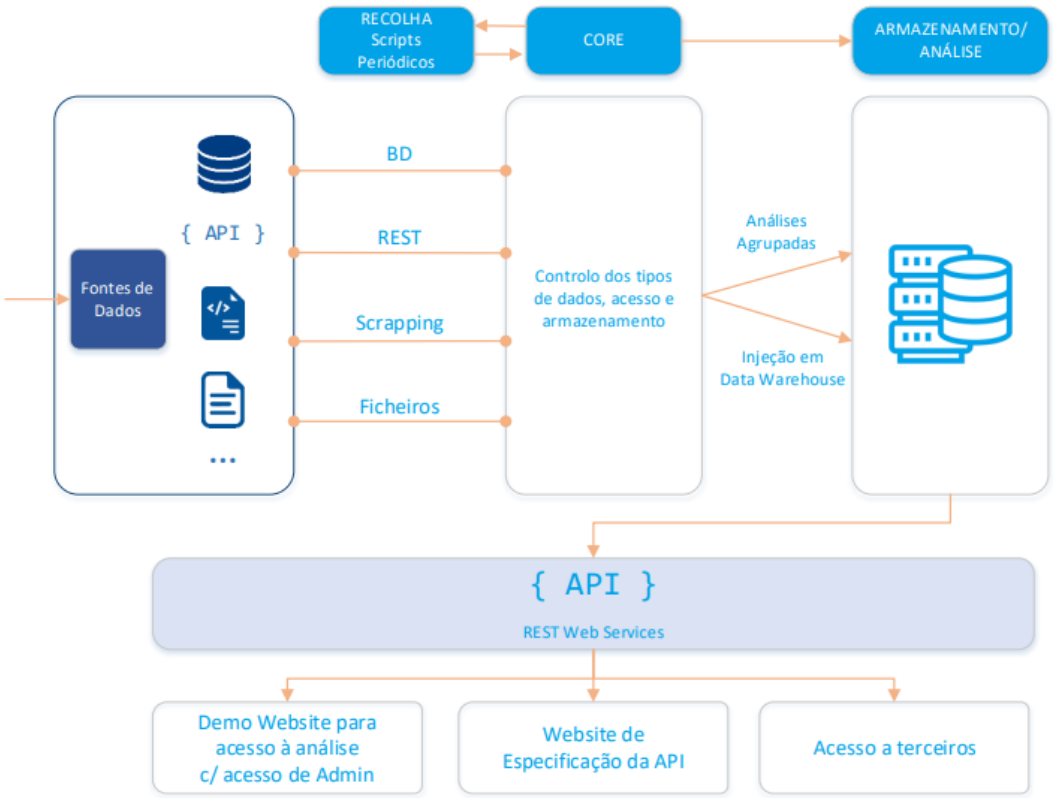

Figura 10: Arquitetura atual da plataforma SMARTDEST [9]

No entanto, esta arquitetura também contém algumas limitações, que incluem problemas fundamentais na infraestrutura, *performance*, e escalabilidade da plataforma.

Uma das limitações mais significativas identificadas na arquitetura atual da plataforma SMARTDEST reside na decisão de implementar um sistema de orquestração de dados completamente desenvolvido de raiz. Apesar de o código apresentar uma lógica funcional para agendamento e execução de tarefas de ingestão de dados, a sua natureza totalmente customizada, utilizando bibliotecas como *node-schedule*, representa uma limitação significativa para a robustez e escalabilidade da plataforma SMARTDEST. Esta abordagem requer que a equipa de desenvolvimento se encarregue de reinventar e manter complexidades inerentes a um sistema de agendamento, como a persistência do estado dos *jobs*, a gestão de falhas distribuídas, *retries*, *backfills*, e a monitorização centralizada. Consequentemente, isto não só aumenta drasticamente o custo e a complexidade de manutenção do sistema a longo prazo, como também restringe a sua capacidade de evoluir para suportar um número crescente de fontes de dados e fluxos de transformação mais complexos, sendo um fator de risco para a fiabilidade e *performance* da plataforma.

Uma limitação adicional, de particular relevância na arquitetura atual, reside na inexistência de estratégias eficazes para o processamento paralelo e distribuído. O processamento paralelo e distribuído assume um papel fundamental, ao possibilitar a divisão de tarefas complexas de transformação e agregação de dados em subtarefas que podem ser executadas em simultâneo por múltiplos núcleos ou máquinas. Esta capacidade é essencial para reduzir de forma significativa o tempo de processamento de grandes volumes de dados, tanto nas etapas de pré-processamento como na execução de *queries* analíticas ou na geração de *insights*, assegurando que a plataforma consegue escalar de forma eficiente e manter o desempenho à medida que a quantidade e a granularidade dos dados turísticos aumentam [57].

Uma outra limitação identificada foi a ausência explícita de uma estratégia para a gestão de dados brutos (*raw data*) e de uma base de dados preparada para dados não estruturados. O esquema atual foca-se na extração e transformação imediata dos dados para um formato relacional (MySQL), sem menção à persistência da informação na sua forma original. Esta abordagem levanta várias preocupações como a impossibilidade da realização de análises futuras que possam exigir um nível de detalhe ou um formato diferente do que foi inicialmente pré-processado e armazenado no Data Warehouse limitando a flexibilidade e a capacidade de adaptação da plataforma a novas exigências analíticas [40].

Uma outra preocupação é que a arquitetura atual assume que todos os dados turísticos podem ser convenientemente modelados e armazenados numa base de dados relacional, no entanto, a realidade do turismo é que uma grande parte dos dados possui formatos diversos e, frequentemente, não estruturados ou semi-estruturados como *reviews*, registos de sensores e documentos. Para uma plataforma de turismo inteligente que visa agregar múltiplas fontes, seria fundamental a inclusão de componentes como um *data lake* (baseado em armazenamento de objetos como S3 ou o Minio) para a ingestão e retenção de dados brutos, independentemente do seu formato.

Uma outra restrição notória na arquitetura da plataforma prende-se com o modelo de dados adotado, especificamente a prática de utilizar uma tabela relacional separada para cada fonte de dados. Embora esta abordagem possa ser funcional para um número limitado de fontes, torna-se uma limitação crítica à medida que o número de *data sources* se expande para dezenas ou centenas. Manter e gerir centenas de tabelas distintas num ambiente relacional, como o MySQL, introduz uma complexidade desnecessária na gestão da base de dados, na manutenção de esquemas e na execução

de *queries* analíticas que necessitem de agregar dados de múltiplas fontes. Esta fragmentação dificulta a otimização de *queries* e pode levar a problemas de *performance*, uma vez que cada agregação exigiria múltiplos JOINs entre tabelas com esquemas potencialmente divergentes. Além disso, a adição de novas fontes de dados implicaria a criação contínua de novas tabelas e a adaptação das *queries* existentes, tornando o sistema menos ágil e mais propenso a erros.

Por último, foi identificada como limitação a ausência de mecanismos eficazes de monitorização e de tolerância a falhas nos dados, fatores essenciais para assegurar a qualidade e a disponibilidade das operações. Atualmente, a visibilidade operacional da plataforma está restringida a um *dashboard* rudimentar, que apenas indica o estado de sucesso das fontes de dados, a sua ativação e o registo da última execução conforme ilustrado na Figura 11, nas colunas cinco, quatro e um, respetivamente. Faltam métricas relevantes como: indicadores de infraestrutura (utilização de CPU, memória, espaço em disco, I/O de rede e latência dos servidores), métricas de desempenho dos *pipelines* (tempo de execução de cada etapa, taxa de sucesso/falha e latência na disponibilização dos dados) e mecanismos de monitorização da qualidade dos dados (validação de schemas, contagem de registos, deteção de duplicados). Além disso, a plataforma revela uma capacidade limitada de tolerância a falhas. Não basta apenas detetar que ocorreu uma falha, é fundamental dispor de alertas proativos, de mecanismos de *retry* e de um sistema de registo histórico e de data *lineage*, que permita consultar o histórico de execuções de cada pipeline e rastrear a proveniência dos dados [58].

Em suma, embora a plataforma SMARTDEST represente um avanço significativo na centralização e disponibilização de dados turísticos na Madeira, a análise da sua arquitetura evidencia limitações estruturais que comprometem a escalabilidade, a flexibilidade e a robustez do sistema. A adoção de soluções mais maduras para orquestração de dados, a inclusão de estratégias para gestão de dados brutos e não estruturados, a reformulação do modelo de dados e a implementação de mecanismos abrangentes de monitorização e tolerância a falhas surgem como medidas imprescindíveis para garantir que a plataforma possa evoluir de forma sustentável e responder eficazmente às crescentes exigências do ecossistema turístico.

3.2 Requisitos do sistema

Após a análise da plataforma atual, realizou-se um levantamento de requisitos fundamentado nessa avaliação, com o objetivo de reestruturar a arquitetura do sistema. Este processo foi desenvol-

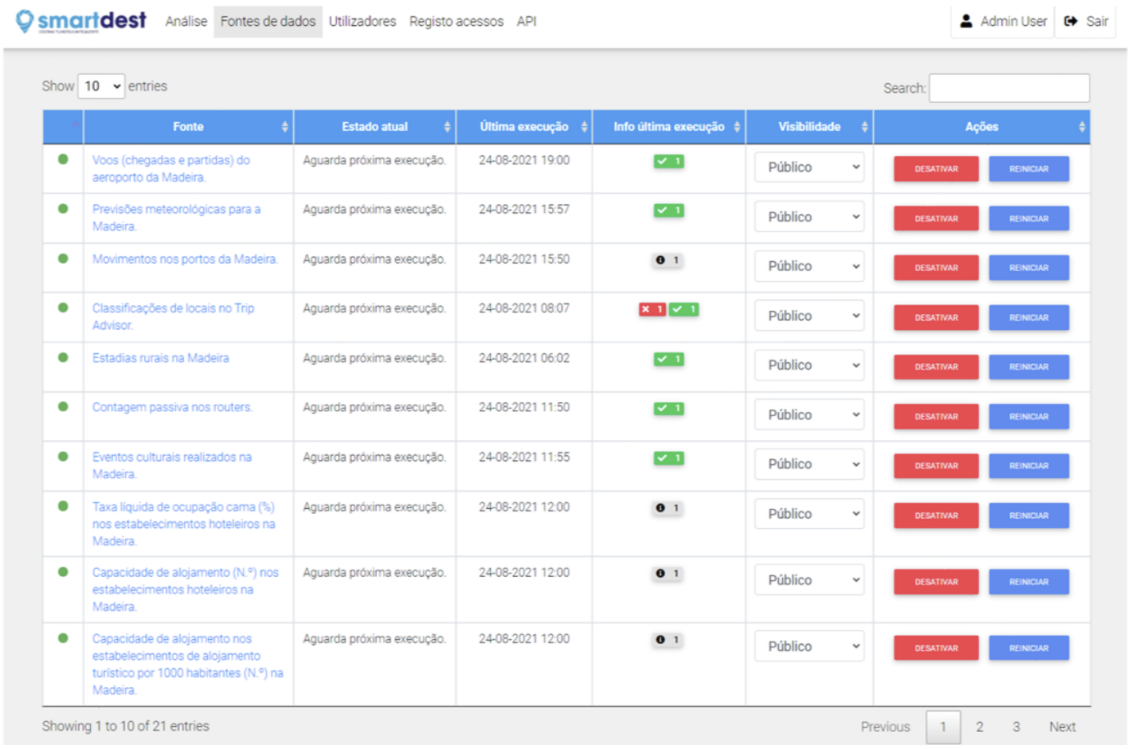

Figura 11: Administração web - Monitorização das fontes de dados [9]

vido através de reuniões com um especialista em desenvolvimento de software, visando identificar os requisitos essenciais e as expectativas operacionais do sistema.

Os requisitos foram categorizados em quatro áreas principais:

- Requisitos de Recolha de Dados
- Requisitos de Armazenamento de Dados
- Requisitos de Processamento de Dados
- Requisitos de Monitorização, Tolerância a Falhas e Qualidade de Dados

Todos os requisitos são classificados como funcionais (F), que descrevem comportamentos ou funções específicas do sistema, ou não funcionais (NF), que definem restrições de desempenho, usabilidade, fiabilidade e outros atributos de qualidade. Cada requisito é identificado pelo seu ID de categoria e o tipo (F ou NF).

Requisitos de Recolha de Dados:

Definem como a arquitetura recolhe os dados provenientes de diversas fontes turísticas, garantindo compatibilidade com diferentes formatos e frequências de atualização, de modo a assegurar a qualidade, consistência e atualidade da informação recolhida (Tabela : 3).

Tabela 3: Requisitos de Recolha de Dados

ID	Tipo	Descrição do Requisito
RID1	F	A plataforma deve ser capaz de testar a conexão a diferentes fontes de dados turísticas, incluindo APIs públicas, bases de dados , *Web Pages* e arquivos CSV/JSON antes da recolha de dados.
RID2	F	A plataforma deve conseguir realizar a recolha de dados de diversas fontes documentais abertas, como ficheiros em formato PDF ou Excel, para além de APIs e web scraping
RID3	F	O sistema deve ter a capacidade de realizar novas tentativas automáticas (*retries*) de recolha em caso de falha temporária de rede ou da fonte de dados, com um intervalo de tempo configurável.
RID4	F	A plataforma deve permitir agendamento de tarefas de recolha com diferentes frequências (ex: diária, horária).
RID5	F	O sistema deve armazenar os dados brutos (*raw data*) na sua forma original, sem processamento, permitindo análises futuras com diferentes níveis de detalhe ou formatos.
RID6	F	O sistema deve registar metadados sobre cada recolha de dados, incluindo a fonte, a data/hora da extração, o volume de dados e o estado da operação.
RID7	NF	O sistema deve manter um registo de auditoria de todas as operações de recolha de dados, permitindo rastrear a proveniência e as transformações dos dados.
RID8	NF	O processo de recolha deve ser facilmente configurável e mantido, com código bem documentado e modular para facilitar futuras adições de fontes de dados ou alterações nos formatos existentes.

Requisitos de Armazenamento de Dados

Definem o modo como os dados, quer em estado bruto quer já processados, são armazenados no sistema, abrangendo tanto a seleção da infraestrutura de armazenamento como a implementação de um modelo de dados flexível, capaz de acomodar diferentes tipos de informação e de se adaptar facilmente a novas exigências do sistema. (Tabela : 4).

Tabela 4: Requisitos de Armazenamento de Dados

ID	Tipo	Descrição do Requisito
AD1	F	A plataforma deve armazenar os dados brutos provenientes de múltiplas fontes em um sistema de armazenamento distribuído, garantindo alta disponibilidade e durabilidade.
AD2	F	O sistema deve armazenar os dados processados e transformados numa estrutura otimizada para análise e consumo por ferramentas de visualização.
AD3	F	O sistema deve garantir a integridade e consistência dos dados armazenados, aplicando validações durante o processo de ingestão e transformação.
AD4	F	O modelo de dados deve ser flexível o suficiente para acomodar diferentes tipos de informações turísticas e permitir a fácil evolução com a adição de novas fontes ou requisitos de dados.
AD5	F	Os dados processados devem ser indexados de forma apropriada para otimizar o desempenho das consultas mais frequentes.
AD6	F	O sistema deve ser capaz de armazenar diferentes tipos de dados, incluindo dados estruturados e dados não estruturados ou semiestruturados.
AD7	NF	O armazenamento de dados deve garantir alta escalabilidade para suportar crescimento no volume de dados ao longo do tempo.
AD8	NF	Os dados armazenados devem estar seguros contra acesso não autorizado.
AD9	NF	O modelo de dados deve ser flexível para permitir alterações e adições sem necessidade de *downtime* significativo ou perda de dados.
AD10	NF	O sistema deve ser resiliente a falhas, garantindo *backup* automático dos dados armazenados.

Requisitos de Processamento de Dados

Definem a forma como os fluxos de dados são orquestrados e transformados ao longo da arquitetura, incluindo a coordenação entre diferentes etapas de processamento, a capacidade de executar operações em paralelo e em ambientes distribuídos para garantir escalabilidade e eficiência, bem como a implementação de mecanismos robustos de deteção e recuperação de falhas, assegurando a continuidade e a fiabilidade do processamento dos dados mesmo em situações de erro (Tabela : 5.

Tabela 5: Requisitos de Processamento de Dados

ID	Tipo	Descrição do Requisito
PD1	F	A plataforma deve permitir a definição, agendamento e monitorização de *workflows* para ingestão, transformação e carregamento dos dados das múltiplas fontes de dados
PD2	F	A plataforma deve ser capaz de realizar transformações complexas nos dados
PD3	F	Deve ser possível reexecutar tarefas ou fluxos de trabalho específicos a partir de um ponto de falha ou manualmente, sem a necessidade de reprocessar todo o pipeline
PD4	F	O sistema deve gerir automaticamente as dependências entre as tarefas, garantindo que uma tarefa só comece após a conclusão bem-sucedida das suas dependências.
PD5	F	O sistema deve suportar a execução paralela de múltiplas tarefas dentro do pipeline, garantindo eficiência no processamento de grandes volumes de dados.
PD6	F	Deverá possibilitar a distribuição das tarefas de processamento entre múltiplos nós ou serviços para otimizar o uso de recursos computacionais.
PD7	F	A plataforma deverá identificar falhas em tarefas ou fluxos de trabalho, registar *logs* detalhados e permitir a reexecução automática ou manual das tarefas falhadas.
PD8	F	Deverá recolher métricas de desempenho e estado das tarefas de processamento para análise e visualização em *dashboards*.
PD9	NF	A solução deverá escalar horizontalmente para suportar aumento no volume de dados e na complexidade dos *workflows* sem perda significativa de desempenho.
PD10	NF	O sistema deverá garantir alta disponibilidade das operações de orquestração e processamento, minimizando perdas de dados e interrupções.
PD11	NF	Deverá haver mecanismos robustos para recuperação automática de falhas, com alertas para notificação de anomalias e falhas.
PD12	NF	O código e a configuração dos *workflows* deverão ser claros, modularizados e documentados para facilitar futuras alterações e extensões.
PD13	NF	Deverá garantir a recolha, armazenamento e visualização de métricas operacionais detalhadas para análise histórica e *troubleshooting*.
PD14	NF	O *pipeline* deverá processar os dados em tempo razoável, considerando o volume esperado, e executar tarefas paralelamente para otimizar o tempo total de processamento.

Requisitos de Monitorização, Tolerância a Falhas e Qualidade de Dados

Definem as capacidades necessárias para assegurar a fiabilidade e a consistência dos dados recolhidos, abrangendo a implementação de mecanismos de monitorização contínua que forneçam

visibilidade operacional sobre o estado e o desempenho da arquitetura, a deteção precoce de anomalias ou degradações no sistema, e a definição de estratégias de tolerância a falhas que permitam recuperar automaticamente de erros, minimizando o impacto no funcionamento global (Tabela 6).

Tabela 6: Requisitos de Monitorização e Qualidade

ID	**Tipo**	**Descrição do Requisito**
MTQ1	F	O sistema deve ser capaz de monitorizar o *status* de cada extração de dados de todas as fontes (por exemplo, APIs, bases de dados externas, ficheiros).
MTQ2	F	Deve ser possível visualizar métricas como o volume de dados extraídos, o tempo de extração e a taxa de sucesso/falha por fonte.
MTQ3	F	O sistema deve alertar automaticamente em caso de falha na extração de dados de qualquer fonte.
MTQ4	F	A plataforma deve fornecer visibilidade sobre o progresso e o *status* de cada worflow.
MTQ5	F	O sistema deve permitir a reexecução manual ou programada de tarefas falhadas.
MTQ6	F	O sistema deve gerar alertas configuráveis para eventos críticos, como falhas na extração, transformação ou carregamento de dados.
MTQ7	F	O sistema deve ter mecanismos automáticos de retentativa para operações que falham temporariamente, com um número configurável de tentativas e atrasos.
MTQ8	F	O sistema deve oferecer painéis de controlo configuráveis que apresentem métricas e visualizações chave sobre o estado operacional da arquitetura.
MTQ9	NF	O *pipeline* deve ser altamente disponível, minimizando o tempo de inatividade em caso de falha de componentes individuais.
MTQ10	NF	A arquitetura deve suportar a recuperação automática de falhas para os serviços críticos.
MTQ11	NF	O sistema deve garantir a entrega de dados "*at least once*"(pelo menos uma vez) para evitar perda de dados em caso de falha.
MTQ12	NF	As métricas de monitorização devem ser atualizadas em tempo real .
MTQ13	NF	Os dados históricos de monitorização devem ser retidos por um período mínimo de 6 meses para análise de tendências e resolução de problemas a longo prazo.
MTQ14	NF	A monitorização deve abranger todos os componentes críticos da arquitetura, desde a recolha de dados até ao armazenamento e processamento, incluindo o uso de recursos (CPU, memória, disco).

Estes requisitos servirão como base para o desenvolvimento da nova arquitetura e serão validados e avaliados através de cenários de validação práticos, detalhados no Capítulo 5.

3.3 Arquitetura Conceptual

Após uma análise da plataforma SMARTDEST e a identificação dos requisitos funcionais e não funcionais para a reestruturação da arquitetura da mesma, iniciou-se a fase de conceção de uma nova arquitetura conceptual. Esta etapa revelou-se essencial para definir a estrutura e o fluxo de dados do sistema.

A arquitetura conceptual proposta para o novo sistema de gestão de dados turísticos está representada na Figura 12 e teve como base a estrutura da arquitetura de referência descrita na Secção 2.2, nomeadamente a de [24], a qual foi desenvolvida com base nas melhores práticas e no contributo de especialistas na área de sistemas de *big data*.

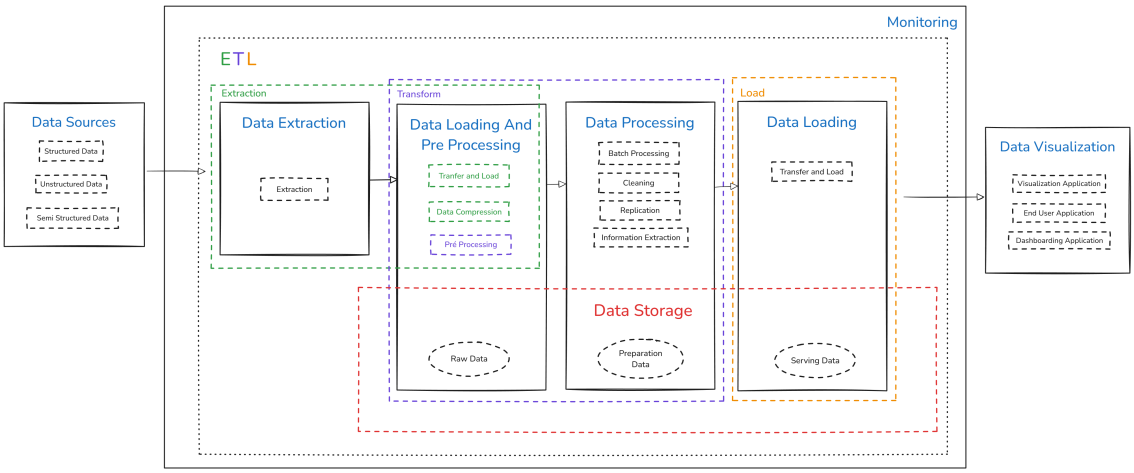

Figura 12: Arquitetura Conceptual

Esta abordagem metodológica permitiu delinear um *pipeline* robusto e flexível, que enfatiza uma clara separação de responsabilidades e a modularidade em todas as fases do ciclo de vida dos dados.

Embora se baseie na arquitetura de referência, a proposta desenvolvida introduz alterações significativas, destacando-se a simplificação de alguns módulos originalmente definidos, especialmente os associados à análise avançada de dados. Foi igualmente integrado um componente de monitorização abrangente, capaz de supervisionar todo o *pipeline*. Esta opção decorre da necessidade de cumprir os requisitos funcionais e não funcionais apresentados na secção anterior.

A arquitetura proposta foi concebida de forma a cobrir todas as fases do ciclo de vida dos dados, desde a sua recolha inicial até ao armazenamento e disponibilização final para consumo.

Esta estrutura é composta por um conjunto de módulos principais, cada um com responsabilidades bem definidas.

A arquitetura proposta tem início no **módulo Data Sources**, como é possível observar na Figura 12. Este módulo tem como objetivo especificar a origem e o formato de todos os dados que serão ingeridos no *pipeline*. As *data sources* podem abranger uma vasta gama de tipologias de dados, incluindo dados estruturados, tipicamente organizados em bases de dados relacionais, dados semiestruturados, como ficheiros JSON, XML, ou *logs* e dados não estruturados, como documentos de texto livre, imagens, áudios ou vídeos, que não se enquadram num modelo predefinido.

A capacidade de lidar com esta diversidade de fontes e formatos de dados é crucial para a robustez e versatilidade do sistema, assegurando que todas as informações relevantes possam ser recolhidas e processadas para posterior análise e armazenamento.

De seguida, o *pipeline* prossegue para o seu fluxo principal, o processo de **ETL (Extract, Transform, Load)**, composto por cinco módulos fundamentais: **Data Extraction**, **Data Loading and Pre-Processing**, **Data Processing**, **Data Loading** e **Data Storage**, conforme ilustrado na Figura 12.

O processo de ETL começa pelo módulo **Data Extraction**, representado na Figura 12, é responsável pela recolha dos dados no seu estado em bruto. A sua principal função reside na capacidade de executar a extração de dados através de diversas técnicas como web scraping, APIs (Application Programming Interfaces), ligações diretas a bases de dados, e leitura de ficheiros em formatos variados. Esta versatilidade assegura que o sistema é capaz de aceder eficientemente à diversidade de fontes definidas no módulo **Data Sources**.

Após a fase de extração, os dados são encaminhados para o módulo Data Loading and Pre-processing, representado na Figura 12. Este módulo assume um papel fundamental no *pipeline*, sendo responsável por receber a informação no seu estado bruto e prepará-la para o armazenamento inicial no *raw data storage*.

O principal objetivo desta etapa é assegurar que os dados sejam transferidos de forma eficiente e armazenados de maneira otimizada. As operações centrais realizadas neste módulo incluem o *transfer and load* isto é, a movimentação eficaz dos dados extraídos para o sistema de armazenamento bruto e a compressão de dados, que visa reduzir significativamente a dimensão dos ficheiros,

maximizando a eficiência do espaço de armazenamento e melhorando o desempenho nas operações subsequentes de leitura e escrita.

Importa salientar que, nesta fase, poderá ainda ser realizado um pré-processamento simples. Não se pretende executar transformações complexas nem aplicar procedimentos avançados de enriquecimento de dados. Pelo contrário, o foco recai na eliminação de anomalias evidentes e na aplicação de formatações mínimas, de modo a assegurar a integridade da informação e a sua preparação básica para utilização futura. Esta abordagem permite que os dados sejam preservados no formato mais próximo possível do original, facilitando a sua reutilização em etapas posteriores de análise que exijam informação na sua forma bruta.

Concluída a fase de pré-processamento, compressão e armazenamento dos dados no seu estado bruto, o fluxo prossegue para o módulo seguinte, designado **Data Processing**, representado na Figura 12. Nesta etapa, os dados anteriormente carregados e tratados de forma elementar são agora submetidos a transformações mais profundas e estruturantes.

O processo de *data cleaning* realizado neste módulo é consideravelmente mais rigoroso, abrangendo operações como a correção de inconsistências, a eliminação de registos duplicados e a padronização de formatos. Estas ações visam garantir a qualidade, integridade e fiabilidade dos dados, aspetos essenciais para suportar análises precisas e consistentes. Para além disso, poderá ser aplicada a replicação de dados, com o intuito de reforçar a sua disponibilidade e garantir redundância, contribuindo para a resiliência do sistema.

O objetivo central do módulo Data Processing é, assim, transformar os dados brutos numa versão mais estruturada, coerente e apta para consumo por sistemas analíticos ou operacionais.

Após as operações de processamento, os dados são temporariamente armazenados numa área de *staging* nomeada Preparation Data, que atua como um repositório intermédio. Este espaço transitório facilita a organização e validação final dos dados antes da sua transferência para o repositório de armazenamento definitivo.

Após a conclusão do processamento dos dados e o seu armazenamento intermédio na área de *staging*, procede-se à fase seguinte do *pipeline*, correspondente ao módulo Data Loading, representado na Figura 12. Este módulo tem como principal finalidade carregar os dados processados para a base de dados de *serving data*, onde se tornam disponíveis para consumo por aplicações, ferramentas de *reporting* e utilizadores finais.

O módulo Data Loading assegura que os dados respeitam o esquema previamente definido para a base de dados de destino, executando as validações finais necessárias com vista a garantir a sua integridade, conformidade e compatibilidade.

Para além das validações estruturais, este módulo é igualmente responsável pela gestão da lógica associada às operações de inserção, atualização e eliminação de registos. Desta forma, assegura-se que a base de dados de *serving* permanece atualizada, coerente e alinhada com os dados mais recentes disponíveis no sistema.

Uma vez concluído o carregamento dos dados na base de dados de *serving data*, o *pipeline* prossegue para o módulo de Data Visualization, representado na Figura 12. Esta fase desempenha um papel fundamental no processo global de gestão e exploração de dados, ao permitir a sua transformação em representações visuais claras, interativas e facilmente interpretáveis.

O principal propósito deste módulo consiste em proporcionar aos utilizadores uma forma intuitiva de explorar a informação disponível, facilitando a identificação de padrões, correlações, tendências e outras evidências relevantes que, no seu formato original e não estruturado, seriam de difícil perceção.

Recorrendo a técnicas e ferramentas de visualização, como gráficos interativos, *dashboards* e relatórios dinâmicos, o módulo Data Visualization assegura que os dados são apresentados de forma acessível, promovendo a tomada de decisões informadas e sustentadas em evidência.

Finalmente, para assegurar a robustez e a eficiência de todo o *pipeline*, criou-se o módulo de **Monitoring**. Este módulo tem como principal responsabilidade a supervisão contínua do desempenho e do estado operacional de todos os componentes e fases da arquitetura proposta.

A atividade de monitorização incide sobre as diversas etapas do processo ETL (Extração, Transformação e Carga) tendo como o seu principal objetivo a deteção proativa de anomalias, falhas, degradações de desempenho ou qualquer outro tipo de ocorrência que possa comprometer a integridade, disponibilidade ou fiabilidade dos dados.

3.4 Arquitetura Final e Tecnologias

Com a arquitetura conceptual devidamente definida, a fase seguinte consistiu na transição para a arquitetura final. Esta etapa focou-se na seleção e integração das tecnologias específicas que iriam suportar cada um dos módulos concebidos, transformando o design abstrato numa solução tangível

e funcional. A escolha destas tecnologias foi cuidadosamente ponderada para garantir a robustez, escalabilidade e eficiência do sistema, alinhando-se com os requisitos funcionais e não funcionais previamente identificados.

A arquitetura final, detalhada na Figura 13, espelha a estrutura conceptual previamente definida, mas agora materializada através da seleção e integração de tecnologias específicas.

Ao longo das próximas secções deste capítulo, serão abordados os componentes de ETL, Monitoring e Data Storage, explicando as tecnologias escolhidas. O componente de Visualization, embora crucial para o sistema, não será discutido, uma vez que a sua implementação não foi minha.

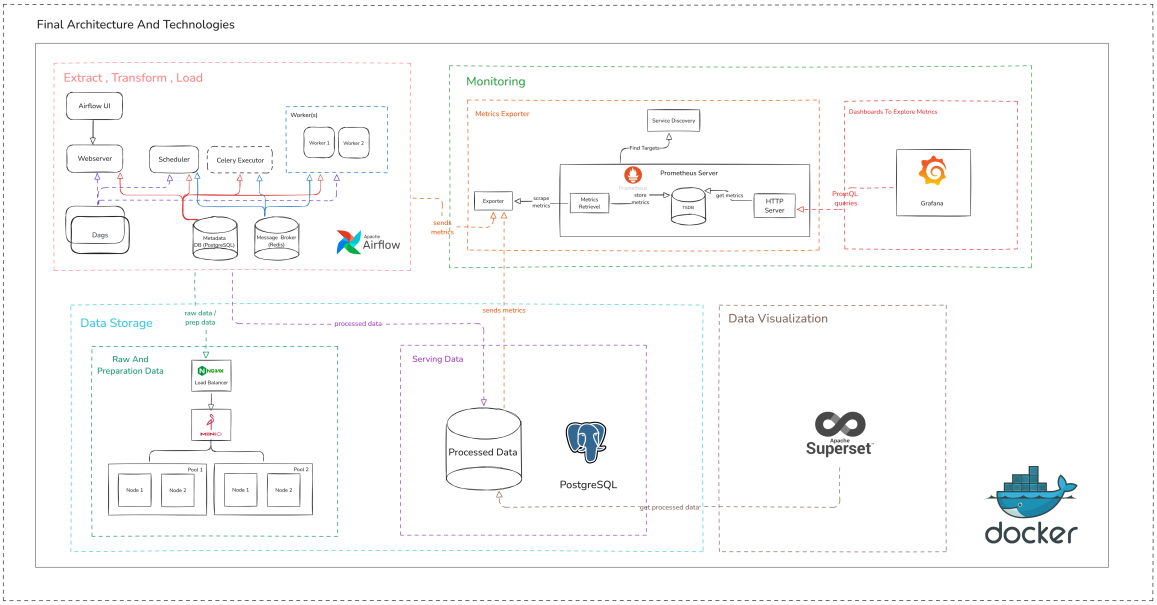

Figura 13: Arquitetura Final e Tecnologias

Para o componente **ETL (Extract, Transform, Load)**, a escolha recaiu sobre o **Apache Airflow** como é possível observar na Figura 13 no canto superior esquerdo. Esta escolha foi feita pois, o Airflow alinha-se com os requisitos de um ecossistema de dados em larga escala e também pela sua capacidade de endereçar eficazmente tanto requisitos funcionais como não funcionais, conforme identificados nas fases de levantamento e conceção. Uma análise mais aprofundada desta ferramenta pode ser consultada na Secção 2.4.1.

Será aqui onde será definido a recolha de cada *data source* e onde será realizado todo o processo de ETL que foi explicado no capitulo anterior.

A seleção do Apache Airflow fundamentou-se em diversas razões estratégicas:

- **Escalabilidade, Desempenho e Resiliência:** O Apache Airflow, devido à sua arquitetura distribuída, garante a robustez necessária para operações críticas de ETL, assegurando uma elevada escalabilidade e resiliência, características essenciais em ambientes de processamento de grandes volumes de dados. Para além disso, o Airflow disponibiliza mecanismos avançados de retentativa automática que asseguram a recuperação eficiente de Workflow em caso de falhas pontuais, minimizando assim a intervenção manual. Adicionalmente, o Airflow suporta a execução paralela de múltiplas tarefas dentro do mesmo *pipeline*, permitindo escalar horizontalmente o processamento e melhorar significativamente o desempenho, reduzindo os tempos de execução em *workflows* de maior complexidade.

- **Extensibilidade e Compatibilidade com Ecossistemas de Big Data:** O Airflow oferece uma vasta gama de operadores e sensores pré-construídos que permitem uma integração simples e direta com as tecnologias mais comuns no ecossistema de *big data*, tais como Hadoop, Spark, Hive, bases de dados NoSQL, serviços *cloud*, entre outros. Esta capacidade de integração facilita o desenvolvimento e acelera a implementação de *pipelines* complexos, reduzindo o esforço associado à construção de soluções de ETL personalizadas.

- **Flexibilidade Programática:** A orquestração de *pipelines* no Airflow é realizada através da definição de *Dags* (Directed Acyclic Graphs) em código Python. Esta abordagem programática permite a criação de *pipelines* altamente complexos, com a especificação detalhada de dependências entre tarefas, agendamento flexível e parametrização dos *workflows*, conferindo ao utilizador um elevado controlo sobre a sua execução.

- **Capacidades de Monitorização Abrangentes:** O Airflow disponibiliza uma interface web (Web UI) que permite monitorizar, de forma clara e intuitiva, o estado de execução de todos os *workflows*. Esta funcionalidade possibilita a rápida deteção de erros, bem como a análise detalhada dos *logs*, que são armazenados na base de dados de metadata do sistema. Esta visibilidade é fundamental para a manutenção da fiabilidade e integridade das operações de ETL.

No que diz respeito ao componente de **Data Storage**, este encontra-se segmentado em três tipos distintos de armazenamento, conforme definido previamente na arquitetura conceptual e ilustrado na Figura 12. Para as camadas de armazenamento designadas como *Raw Data* e *Preparation Data*, foi selecionado o **MinIO**, enquanto que para a camada de *serving data* a escolha recaiu sobre

a base de dados relacional **PostgreSQL** como é possível observar no canto inferior esquerdo da Figura 13.

O MinIO é uma solução de armazenamento de objetos compatível com a API S3 da Amazon, amplamente adotada no domínio das plataformas de *big data* devido à sua escalabilidade, desempenho e simplicidade de integração. Projetado para suportar cargas de trabalho intensivas de dados, o MinIO destaca-se pela sua capacidade de fornecer armazenamento distribuído altamente disponível e tolerante a falhas, características essenciais em arquiteturas modernas de dados, como *data lakes* e *pipelines* de *machine learning*. Este foi concebido para armazenar qualquer tipo de dados não estruturados, sendo particularmente eficaz no armazenamento de ficheiros de grandes dimensões e em grande volume, como imagens, vídeos, ficheiros de áudio, documentos, *backups* e ficheiros de *logs* [59].

Nesta arquitetura o MinIO vai ser responsável por armazenar a *Raw Data* que é enviada pelo o Airflow durante a fase inicial do processo de ETL e também responsável por guardar os dados temporários antes de serem armazenados na *serving data*.

A seleção do MinIO como ferramenta para o armazenamento da *Raw Data* e da *Preparation Data* fundamentou-se nas seguintes razões [59] :

- **Armazenamento de Objetos:** Sendo um armazenamento de objetos, o MinIO é ideal para lidar com a natureza diversificada e o volume crescente de dados que advêm de múltiplas fontes do setor turístico. O modelo de armazenamento de objetos é altamente escalável, não impõe uma estrutura rígida e é otimizado para grandes volumes de dados não estruturados ou semi-estruturados, o que é típico em cenários de *big data*.

- **Escalabilidade Horizontal:** O MinIO foi desenhado para escalar horizontalmente de forma elástica. Isto significa que, à medida que o volume de dados de turismo da Madeira cresce (o que é expectável), pode-se adicionar mais nós de armazenamento ao *cluster* MinIO para expandir a capacidade e o desempenho, sem interrupções. Esta flexibilidade é vital para acomodar picos de dados e o crescimento a longo prazo da plataforma.

- **Desempenho Otimizado:** O MinIO é conhecido pela sua elevada *performance* para operações de leitura e escrita, especialmente em cenários de dados de grande dimensão. A sua arquitetura

foi otimizada, o que é essencial para *pipelines* de ETL que precisam de processar grandes volumes de dados de forma eficiente e rápida.

- **Robustez e Resiliência:** O MinIO incorpora funcionalidades avançadas de proteção de dados, como a paridade e o *erasure coding*. Estas características garantem a resiliência dos dados contra falhas de *hardware* ou de nós, distribuindo os dados e a informação de paridade por múltiplos discos e servidores. Mesmo que alguns componentes falhem, os dados podem ser reconstruídos, assegurando a durabilidade e a disponibilidade contínua dos dados críticos para a análise do turismo.
- **Compatibilidade com o Ecossistema de Big Data :** Esta compatibilidade é crucial, pois permite a integração sem atritos com uma vasta gama de ferramentas e *frameworks* que já suportam a API S3, como Apache Spark, Apache Hive, Presto/Trino, e diversas bibliotecas de Python e Java. Esta interoperabilidade significa menos tempo a desenvolver conectores personalizados e mais tempo a focar-se na análise dos dados.

Ao contrário do MinIO, o PostgreSQL é um sistema de gestão de bases de dados relacionais (RDBMS) amplamente reconhecido pela sua robustez, fiabilidade e elevado desempenho. Esta tecnologia permite a gestão e o armazenamento eficientes de grandes volumes de dados de forma estruturada, através de tabelas compostas por linhas e colunas, oferecendo suporte a relações complexas entre essas mesmas entidades [60].

No presente contexto, o PostgreSQL será responsável por receber os dados já processados e validados, no final do processo de ETL executado pelo Airflow.

Deste modo, para além da sua reconhecida fiabilidade, o PostgreSQL apresenta-se como uma solução particularmente adequada para o armazenamento de dados processados e estruturados, requisito fundamental na camada de *serving data* [60]. Esta característica garante que a informação se encontra otimizada para consultas rápidas e eficientes, fator determinante para aplicações que exigem elevado desempenho na disponibilização e visualização de dados, como é o caso da plataforma SMARTDEST.

Adicionalmente, a *performance* do PostgreSQL em cenários de consultas complexas, aliada à vitalidade da sua comunidade que assegura suporte contínuo e promove inovação tecnológica reforça a sua seleção como a opção mais estratégica para a camada de *serving data*. Desta forma, assegura-

se que os dados do setor do turismo permanecem permanentemente disponíveis, consistentes e devidamente otimizados para consumo por sistemas e utilizadores finais [60].

No que respeita ao componente de Monitoring, este encontra-se dividido em dois módulos distintos: um dedicado à extração das métricas e outro responsável pela receção dessas métricas e respetiva apresentação através de *dashboards*. Para a recolha e extração das métricas foi selecionada a ferramenta **Prometheus**, enquanto que, para a exploração, visualização e monitorização das mesmas, foi escolhida a ferramenta **Grafana** como é possível observar no canto superior direito da Figura 13. Estas ferramentas foram selecionadas por permitirem cumprir todos os requisitos previamente definidos, como por exemplo monitorização em real time, a visualização flexível e a configuração de alertas [61].

O Prometheus é uma ferramenta de monitorização e alerta open-source que se destaca pela sua abordagem baseada em *pull* para recolha de métricas. Através de "*exporters*", o Prometheus consegue extrair métricas de diversas fontes, incluindo o Apache Airflow e a base de dados PostgreSQL como é caso nesta arquitetura, que são componentes vitais [62].

Complementarmente ao Prometheus, o Grafana é utilizado para a visualização e análise das métricas recolhidas, transformando os dados brutos em *dashboards* interativos e intuitivos. Com o Grafana, é possível criar painéis personalizados que apresentam o estado dos *workflows* do Airflow e o desempenho do PostgreSQL. Esta sinergia entre o Prometheus e o Grafana oferece uma visibilidade abrangente sobre a maior parte da plataforma, permitindo identificar rapidamente anomalias, otimizar recursos e responder proativamente a potenciais problemas, assegurando a fiabilidade e eficiência da plataforma [62].

A seleção de Prometheus e Grafana para o módulo de monitorização da plataforma foi escolhida devido à sua sinergia e por ser uma solução de monitorização robusta e com um ecossistema amplamente adotado na indústria [62].

A transição para a arquitetura final foi um passo decisivo, transformando o design conceptual numa solução tangível e funcional através da seleção criteriosa de tecnologias que garantem a robustez, escalabilidade e eficiência do sistema. O Apache Airflow destaca-se no ETL pela sua resiliência e flexibilidade na orquestração de *pipelines* de dados, enquanto o MinIO e o PostgreSQL foram escolhidos para o Data Storage, oferecendo armazenamento escalável de objetos e gestão robusta de dados relacionais, respetivamente. Para o Monitoring, a sinergia entre o Prometheus e

o Grafana assegura uma visibilidade abrangente e em tempo real sobre a saúde da plataforma. Em suma, esta arquitetura tecnologicamente fundamentada não só concretiza os requisitos funcionais e não funcionais, mas também posiciona a plataforma para um desempenho otimizado e contínuo para o setor do turismo.

4 Desenvolvimento

Neste capítulo é descrito o processo de desenvolvimento da arquitetura previamente apresentada, bem como os processos que foram definidos para a sua implementação.

Para o desenvolvimento deste projeto, foram-me disponibilizadas duas máquinas com sistema operativo Debian. A primeira foi dedicada ao armazenamento dos dados já processados e estruturados, recorrendo para esse efeito à base de dados PostgreSQL. A segunda máquina foi destinada à implementação (num ambiente Docker) das restantes ferramentas que compõem a arquitetura final, nomeadamente Apache Airflow, MinIO, Prometheus e Grafana.

4.1 Configuração dos Componentes

O trabalho teve início com a instalação e configuração do PostgreSQL na máquina destinada ao armazenamento da *serving data*, isto é, dos dados que já se encontram devidamente tratados e preparados para consumo por sistemas ou utilizadores finais.

Concluída esta etapa, procedeu-se à instalação do Docker numa segunda máquina, com o objectivo de implementar as ferramentas necessárias de forma eficiente. A escolha do Docker revelou-se estratégica, permitindo simular um ambiente distribuído, garantir a portabilidade, simplificar a gestão de dependências e acelerar o processo de desenvolvimento.

Foram então criados dois ficheiros docker-compose.yaml, cuidadosamente configurados para orquestrar adequadamente os diversos serviços que compõem a arquitetura. Estes ficheiros permitiram definir e interligar, de forma isolada e modular, todos os componentes essenciais. É possível observar os detalhes das seguintes configurações nos anexos 7.1 7.2 7.3 7.4 7.5 7.6.

Configuração Apache Airflow

O **Apache Airflow**, elemento central para a orquestração dos processos ETL, foi criado recorrendo a uma imagem personalizada, `extending_airflow:latest_v8`, baseada na imagem oficial (`apache/airflow:2.10.5`), conforme definido no ficheiro `docker-compose.yml` no anexo 7.1. Esta imagem personalizada foi criada com o objetivo de adicionar novas *packages* à imagem oficial do Apache Airflow estando estas presentes num ficheiro `requirements.txt` como é possível observar no anexo 7.6.

A sua configuração exigiu o mapeamento de volumes para persistência dos *Dags* (`/opt/airflow/dags`), logs de execução (`/opt/airflow/logs`), configurações (`/opt/airflow/config`) e *plugins* (`/opt/airflow/plugins`).

De seguida, foi necessário definir o tipo de *Executor* a utilizar pelo Apache Airflow, responsável por gerir a execução das tarefas descritas nos *Dags*. O Airflow suporta diversos tipos de executores, nomeadamente:

- SequentialExecutor: executa uma tarefa de cada vez, ideal para testes e desenvolvimento.
- LocalExecutor: permite a execução paralela de tarefas em múltiplos processos numa única máquina.
- CeleryExecutor: permite execução distribuída através de múltiplos workers, usando um *message broker* (ex.: RabbitMQ ou Redis).
- DaskExecutor e KubernetesExecutor: destinados a ambientes mais exigentes em termos de escalabilidade, frequentemente usados em *clusters* e infraestruturas cloud.

Neste projeto, foram testados tanto o *CeleryExecutor* (anexo 7.7) como o *LocalExecutor* (anexo 7.1), com o intuito de determinar qual se adequaria melhor aos recursos e exigências do sistema. Apesar da escalabilidade proporcionada pelo *CeleryExecutor*, optou-se pelo *LocalExecutor* (`AIRFLOW__CORE__EXECUTOR: LocalExecutor`), uma vez que a complexidade adicional inerente à gestão de uma infraestrutura distribuída não se justificava face aos recursos disponíveis. O *LocalExecutor* revelou-se suficiente para garantir uma execução eficiente dos *workflows*, simplificando simultaneamente a configuração e manutenção do sistema.

Posteriormente, foram definidos os serviços necessários à execução do Apache Airflow no ambiente de desenvolvimento. Todos os serviços foram implementados em *containers* Docker, garantindo portabilidade, isolamento e escalabilidade do ambiente.

A configuração base incluiu os seguintes serviços:

- PostgreSQL: utilizado como base de dados relacional para persistência do estado e metadados do Airflow. Foi utilizada a imagem `postgres:13`, com os parâmetros essenciais de ambiente definidos, nomeadamente o nome da base de dados (airflow), o utilizador (airflow) e a respetiva palavra-passe. Foi ainda configurado um volume persistente para garantir que os dados não se percam entre reinicializações do *container*.

- Airflow Webserver: responsável pela interface web do Airflow, acessível através da porta 8080, permitindo aos utilizadores visualizar os *Dags*, monitorizar a execução das tarefas e interagir com o sistema.
- Airflow Scheduler: responsável por verificar periodicamente os *Dags* e agenda as tarefas a executar com base nas definições de cada *workflow*.
- Airflow Triggerer: responsável pela execução de tarefas baseadas em eventos assíncronos, permitindo uma utilização mais eficiente de recursos.

Todos os componentes do Apache Airflow em funcionamento podem ser observados nas seguintes Figuras 14 e 15 , onde se verifica o arranque bem-sucedido de todos os serviços (Figura 14). Adicionalmente, na Figura 15 seguinte é apresentada a interface web (Web UI) do Airflow, acessível e operacional.

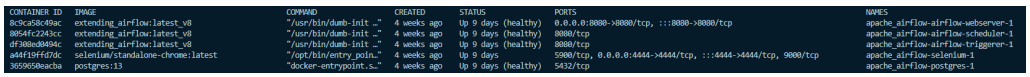

Figura 14: Apache Airflow Containers

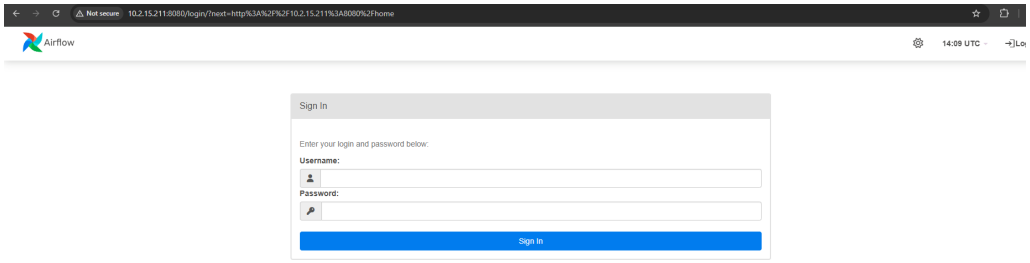

Figura 15: Apache Airflow Web UI

Configuração MinIO

O **MinIO**, selecionado para as camadas de *Raw Data* e *Preparation Data*, foi implementado, de forma distribuída e com balanceamento de carga, garantindo elevada disponibilidade e escalabilidade.

A configuração do MinIO, detalhada no docker-compose.yml (Anexo 7.2), utilizou dois nós (minio1 e minio2) em modo distribuído. Foram montados múltiplos volumes de dados localmente para cada nó, o que garante redundância, tolerância a falhas e escalabilidade horizontal ao sistema.

Cada instância MinIO foi criada a partir da imagem oficial minio/minio, utilizando a versão estável mais recente à data do projeto.

Para facilitar a gestão e o acesso, os comandos de arranque dos *containers* foram parametrizados para expor duas portas essenciais: a porta 9000 para a interface principal do MinIO e a porta 9001 para a consola de administração, acessível através de um painel de controlo web.

Para além dos nós MinIO, incorporou-se um serviço NGINX, cuja configuração se encontra detalhada no Anexo 7.3. Este foi configurado como *reverse proxy* e balanceador de carga, garantindo um acesso unificado e distribuído aos dois nós MinIO. O acesso foi estabelecido através das portas 9000 (para o serviço de armazenamento de objetos) e 9001 (para a interface de administração).

A configuração do NGINX, definida no ficheiro nginx.conf, especificou dois blocos *upstream*. Um para o serviço principal de armazenamento e outro para o acesso à consola. Estes blocos incluíram regras de distribuição e encaminhamento de pedidos, assegurando que o tráfego seria devidamente direcionado. Adicionalmente, ativaram-se otimizações cruciais, como client_max_body_size 0 e proxy_buffering off. Estas definições foram essenciais para suportar o envio de ficheiros de grandes dimensões e garantir uma transmissão de dados eficiente e em tempo real.

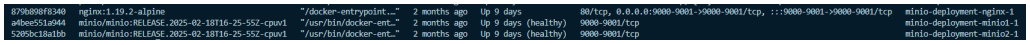

Figura 16: Minio Containers

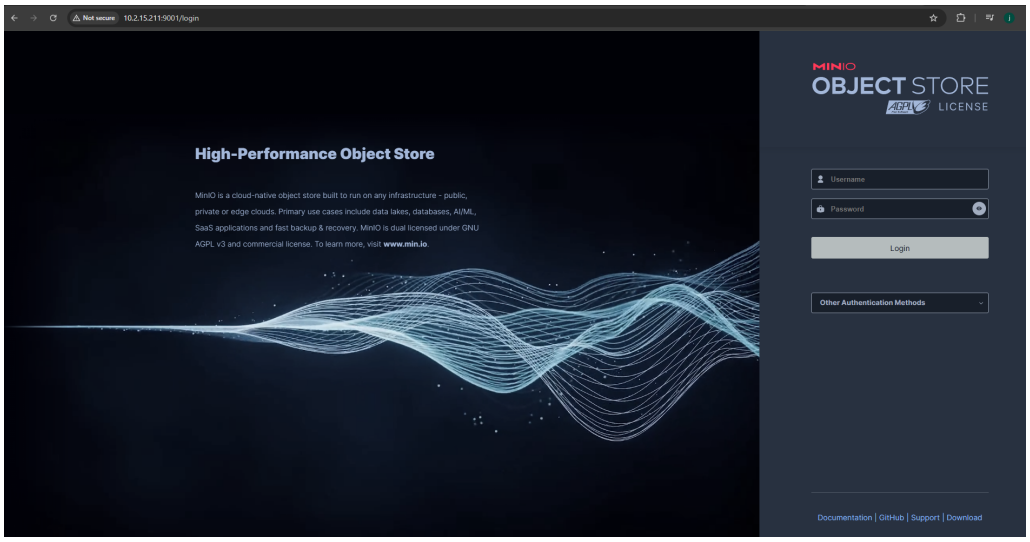

Figura 17: Minio Web UI

Todos os componentes do MinIO em funcionamento podem ser observados nas seguintes Figuras 16 e 17 , onde se verifica o arranque bem-sucedido de todos os serviços (Figura 16). Adicionalmente, na Figura 17 é apresentada a interface web do MinIO, acessível e operacional.

Configuração Prometheus e Grafana

Por fim, para a monitorização global da *arquitetura*, foram integrados os serviços Prometheus e Grafana, conforme especificado no docker-compose.yml no anexo 7.1.

O Prometheus foi configurado como sistema central de recolha de métricas, recorrendo à imagem oficial prom/prometheus. O serviço foi exposto na porta 29090 e configurado com base num ficheiro externo prometheus-config.yml (anexo 7.4), onde foram definidos os *targets* de monitorização, nomeadamente o Airflow e o PostgreSQL.

As métricas do Airflow foram recolhidas através do serviço statsd-exporter, que converte os eventos StatsD emitidos pelo Airflow num formato compatível (anexo 7.5) e expõe na porta 9102 com o Prometheus.

As métricas do PostgreSQL (localizado na primeira máquina) foram recolhidas através do serviço PostgreSQL Exporter, configurado para se ligar remotamente à instância de PostgreSQL e expor as métricas na porta 9187.

Em complemento, foi integrado o Grafana, com base na imagem oficial grafana/grafana:12.0.0. Este serviço foi exposto na porta 23000 e configurado com volumes persistentes para salvaguarda de *dashboards* e definições, assegurando a continuidade das visualizações e análises.

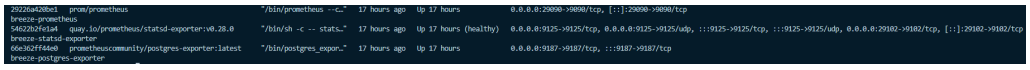

Figura 18: Prometheus Container

Todos os componentes do sistema de monitorização, nomeadamente o Prometheus e o Grafana, encontram-se em execução conforme ilustrado nas Figuras 18 e 20, onde é possível verificar o arranque correto dos respetivos serviços. Nas imagens subsequente, é apresentada a interface web do Grafana (Figura 21) e do Prometheus (Figura 19), demonstrando o seu correto funcionamento e acessibilidade.

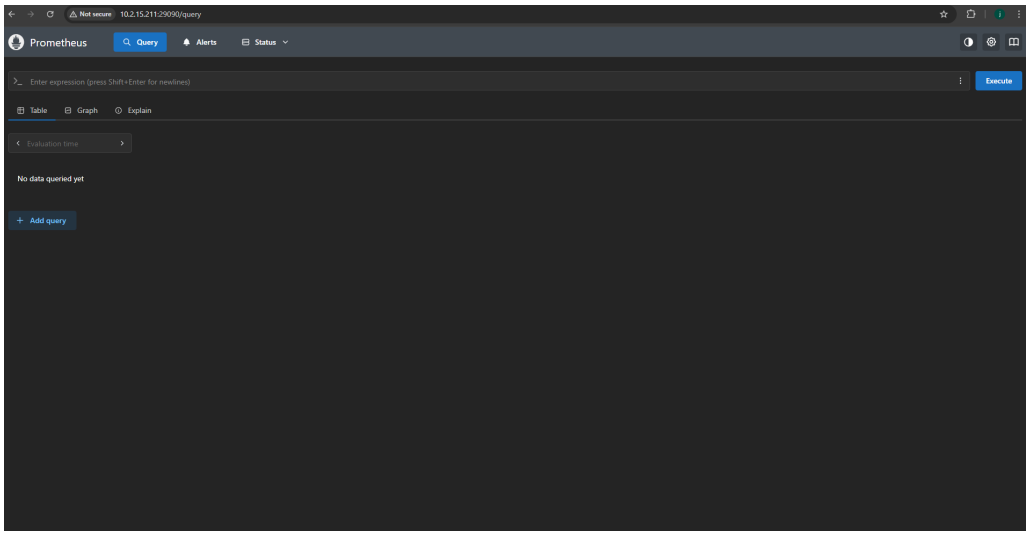

Figura 19: Prometheus Web UI

Figura 20: Grafana Container

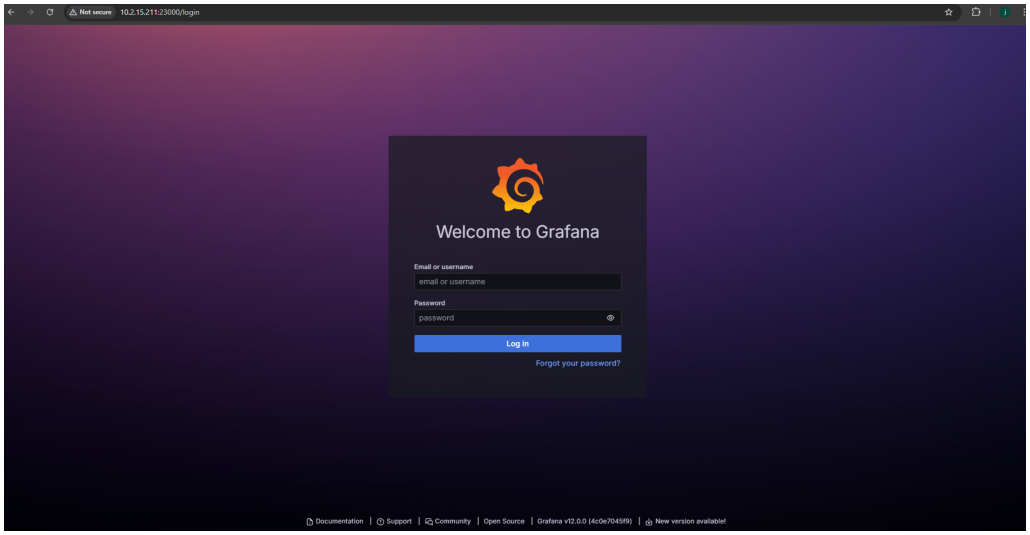

Figura 21: Grafana Web UI

4.2 Implementação do Pipeline

Com a infraestrutura devidamente implementada e os serviços essenciais operacionais, avançou-se para a fase subsequente deste trabalho, o desenvolvimento do fluxo de processamento de dados.

4.2.1 Análise das Fontes de Dados do SMARTDEST

O primeiro passo consistiu numa análise aprofundada das fontes de dados do antigo sistema SMARTDEST. O objetivo foi identificar quais fontes se mantinham ativas, válidas e funcionais. Esta verificação visou assegurar a continuidade da utilização de dados relevantes e fiáveis, evitando redundâncias. Como resultado desta análise, foi criada a Tabela 7.

A Tabela 7 oferece uma visão geral das dez principais fontes de dados utilizadas no sistema SMARTDEST, abrangendo diversas tipologias de informação. Estas incluem desde dados de chegadas e partidas de voos e movimentos de navios, até dados sobre atividades turísticas, como avaliações de restaurantes, eventos e alojamentos rurais. Cada fonte é caracterizada pelo tipo de dados que disponibiliza, o método de acesso ou origem, a frequência de atualização e o seu estado funcional atual.

Da análise efetuada, verificou-se que apenas duas fontes permanecem ativas e funcionais: as previsões meteorológicas (OOM) e as estatísticas oficiais de turismo (INE), ambas com dados regularmente atualizados. As restantes encontram-se atualmente indisponíveis ou descontinuadas no caso de recolha via API. No que respeita às recolhas por *web scraping*, as atualizações no HTML e CSS dos sites, ao longo do tempo, provocaram erros nos processos de extração tornando-as inutilizáveis.

Esta análise foi crucial para determinar quais fontes poderiam ser reutilizadas, quais exigiriam substituição e onde poderiam surgir oportunidades para integrar novas fontes de dados mais estáveis e atualizadas.

Tabela 7: Visão geral das fontes de dados do SMARTDEST

Fonte	**Tipo de dados**	**Origem**	**Frequência de atualização**	**Funcional**
Aeroporto Madeira (ANA)	Chegadas/partidas de voos	API (JSON)	De hora em hora	Não
OOM	Previsões meteorológicas	API (XML)	Diário	Sim
APRAM	Movimentos de navios	API (JSON)	Diário	Não
Beanstalk Wi-Fi	Contagens de dispositivos Wi-Fi	API (JSON)	Diário	Não
Travel Portal (TripAdvisor)	Avaliações e análises dos 85 principais sites	Web Scraping	Diário	Não
Social Media	Posts no Instagram, Flickr, Twitter	API (JSON) + Web Scraping	Diário	Não
INE	Estatísticas oficiais de turismo	API (JSON/XML)	Mensal	Sim
Madeira Rural	Ofertas de alojamento rural	API (JSON)	Diário	Não
Restaurant Portal (TheFork)	Avaliações, classificações e preços	Web Scraping	Diário	Não
Event Agenda (DRT + ViralAgenda)	Listagens de eventos	API + Web Scraping	Diário	Não

Com base nesta análise, foi elaborada uma nova lista de fontes de dados a integrar no sistema redesenhado. Esta seleção teve como critérios principais a atualidade, estabilidade, relevância e a viabilidade técnica da extração dos dados.

Das fontes anteriormente utilizadas, apenas a OOM foi mantida, dada a sua fiabilidade e utilidade para previsões meteorológicas de curto prazo.

Além disso, foram integradas novas fontes que não faziam parte do sistema SMARTDEST original, como por exemplo os dados meteorológicos do IPMA e informações sobre eventos culturais fornecidas pelo Visit Madeira.

A Tabela 8 resume as novas fontes de dados planeadas, evidenciando a diversidade de fontes, formatos e frequências de atualização.

Tabela 8: Novas Fontes de Dados Definidas para a plataforma SMARTDEST

Fonte	**Tipo de dados**	**Origem**	**Frequência de atualização**	**Funcional**
OOM	Previsão do tempo (3 dias)	API (XML)	Diário	Sim
IPMA	Avisos meteorológicos	API (JSON)	A cada 13h e 23h	Sim
IPMA	Dados de previsão das condições do mar	API (JSON)	A cada 13h e 23h	Sim
DREM	Estatísticas de Turismo (Mensal)	CSV	Uma Vez	Sim
DREM	Estatísticas das Receitas Turísticas	CSV	Uma Vez	Sim
DREM	Estatísticas de turismo por país	CSV	Uma Vez	Sim
DREM	Turismo Hóspedes Por tipo de acomodação Estatísticas	CSV	Uma Vez	Sim
DREM	Estatísticas de tipo de acomodação turística	CSV	Uma Vez	Sim
DREM	Receita por Tipo de Alojamento	CSV	Uma Vez	Sim
DREM	Receita por tipo de acomodação por mês	CSV	Uma Vez	Sim
DREM	Custos por tipo de acomodação por mês	CSV	Uma Vez	Sim
DREM	Passageiros em trânsito no porto	CSV	Uma Vez	Sim
DREM	Passageiros em trânsito no porto por nacionalidade	CSV	Uma Vez	Sim
DREM	Publicações de Turismo	Web Scrapping	Semanalmente	Sim
Visit Madeira	Eventos Culturais	Web Scrapping	Diário	Sim
APRAM	Tráfego Marítimo	Web Scrapping	Diário	Sim
APRAM	Agendamento de doca	Web Scrapping	Diário	Sim
Aeroporto Madeira	Chegadas e partidas	Web Scrapping	Diário	Sim
eDreams	Ofertas de voos	Web Scrapping	Diário	Sim

Esta análise das fontes de dados do sistema SMARTDEST foi um passo fundamental para o desenvolvimento do novo fluxo de processamento de dados, revelando que apenas uma minoria das fontes originais se mantinha funcional. Esta verificação detalhada permitiu não só identificar as fontes reutilizáveis, como a OOM, mas também substituir as desatualizadas.

4.2.2 Definição do pipeline de dados

Após a análise das fontes de dados, procedeu-se à definição da abordagem metodológica a adotar para a construção do *pipeline* de dados, tendo em conta a arquitetura anteriormente definida. Esta definição teve como objetivo garantir um processo estruturado, fiável e passível de adaptação às particularidades de cada fonte de dados.

Foi, então, concebido um *pipeline* genérico que descreve as etapas comuns a todas as fontes de dados, independentemente do seu formato ou do método de acesso. A Figura 22 ilustra este processo que está repartido em seis partes :

- *Set metada for the data Source:* consiste na definição e registo de um conjunto de metadados essenciais para a caracterização de cada fonte de dados integrada no *pipeline*. Para cada uma, são registados o nome, a origem , uma breve descrição, o método de acesso, a sua relevância, o intervalo de atualização, o domínio temático e o estado atual. Esta informação permite uma gestão eficiente das fontes e facilita futuras atualizações ou substituições no *pipeline*.
- *Configuration validation and endpoint integrity check:* Nesta fase, são testados os acessos aos endpoints para garantir que estão operacionais e que respondem conforme esperado. Adicionalmente, valida-se a configuração específica de cada fonte, assegurando que todos os parâmetros necessários estão corretamente definidos e que quaisquer requisitos extras, específicos de determinada fonte de dados, são cumpridos. Esta validação prévia é fundamental para evitar falhas durante a recolha dos dados e garantir a fiabilidade do *pipeline*.
- *Data Fetching and Upload to Raw Storage:* Os dados são obtidos exatamente como definido na fase inicial de metadados, seja através de uma API, *web scraping* ou outro método de acesso. Após a recolha, é realizado um ligeiro pré-processamento, que consiste em pequenas operações para organizar e limpar minimamente os dados, garantindo que estão em condições de serem armazenados. Finalmente, estes dados brutos (*raw data*), na sua forma quase original, são carregados e guardados no Raw Storage.
- *Data Processing and Temporary Storage:* Nesta etapa, os dados brutos são submetidos a um processamento mais aprofundado, que pode incluir a limpeza, transformação, agregação e enriquecimento dos dados, de forma a prepará-los para a análise e utilização final. Este processamento visa padronizar formatos, resolver inconsistências, lidar com valores omissos e extrair informações relevantes, convertendo os dados brutos em um formato mais estruturado e utili-

zável. Uma vez processados, os dados são então armazenados temporariamente servindo como um ponto intermédio antes da validação final e do carregamento para o armazenamento de dados processados.

- *Data Quality And Validation:* Nesta etapa, são implementados mecanismos para avaliar a qualidade dos dados, verificando a sua consistência, completude, precisão e unicidade. São aplicadas regras de validação predefinidas, que podem incluir a verificação de tipos de dados, intervalos de valores, formatos específicos e a identificação de registos duplicados ou em falta. O objetivo é detetar e assinalar quaisquer anomalias ou inconsistências nos dados, garantindo que apenas informação de alta qualidade progride para o armazenamento de dados processados.
- *Upload to Processed Data Storage:* Nesta etapa final, os dados que foram submetidos a todas as transformações necessárias e que passaram com sucesso pelo controlo de qualidade e validação são carregados para um armazenamento de dados processados.

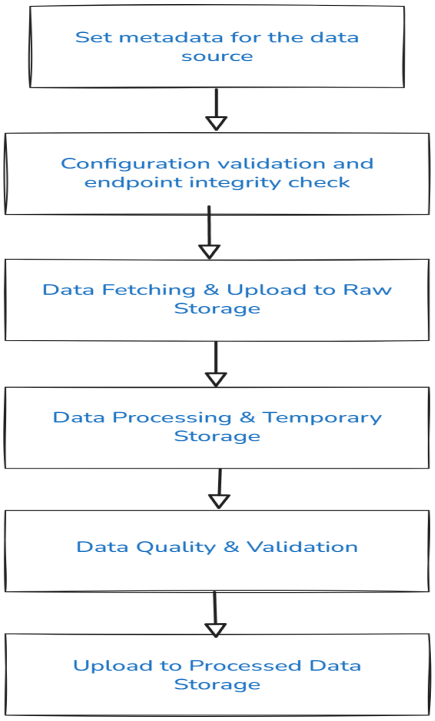

Figura 22: Definição do Pipeline de Dados

4.2.3 Definição do Modelo de Armazenamento de dados

Após a conceção do *pipeline* genérico, um passo fundamental para a sua operacionalização e para a posterior utilização consistiu na definição do modelo de armazenamento de dados. Esta fase

é crucial, pois determina como os dados serão estruturados, organizados e armazenados ao longo de todo o seu ciclo de vida dentro do sistema.

Como demonstrado na arquitetura descrita nos capítulos anteriores, esta solução baseia-se em três tipos principais de armazenamento. O primeiro utiliza o MinIO, responsável pela conservação dos dados brutos (*raw data*), em segundo, dos dados em fase de preparação (preparation data) e o terceiro recorre ao PostgreSQL, destinado ao armazenamento dos dados já processados.

Estrutura de Armazenamento no MinIO para Dados Brutos

A gestão eficiente do volume crescente de dados brutos, na sua forma original e sem alterações, é um pilar essencial para a fiabilidade e rastreabilidade do *pipeline*. Para tal, foi desenhada uma estrutura de armazenamento lógica e hierárquica dentro do ambiente MinIO. A Figura 23 ilustra esta organização que visa facilitar a descoberta e recuperação dos dados.

Esta organização hierárquica é delineada da seguinte forma:

- Bucket : No nível mais alto, todos os dados brutos são centralizados num único *bucket* denominado "Raw Data". Este *bucket* funciona como o contentor principal para toda a informação na sua forma original, antes de qualquer transformação significativa.

- Folder data_source_name : Imediatamente abaixo do *bucket*, é criada uma pasta para cada fonte de dados. Isto é crucial para a governança de dados e para facilitar a identificação e o acesso a dados específicos de uma determinada origem.

- Folder year e month : Dentro da pasta de cada fonte de dados, a organização prossegue com pastas organizadas por ano e mês. Esta estrutura baseada no tempo (data_source_name/YYYY/MM/) é uma prática comum e extremamente eficaz para dados que são recolhidos de forma contínua ou periódica.

- Ficheiro data_source_name_year_month_day : Finalmente, no nível mais granular, os dados são armazenados cujo nome incorpora o nome da fonte de dados, o ano, o mês e o dia (nome_da_fonte_AAAA_MM_DD). Esta convenção de nomenclatura facilita a identificação unívoca e a pesquisa de dados específicos por *data*.

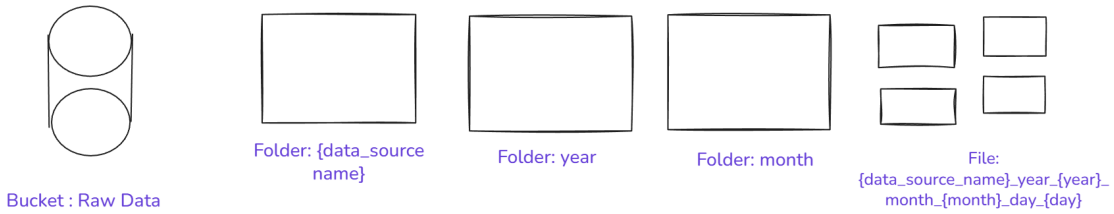

Figura 23: Modelo de Armazenamento de dados - Minio

Armazenamento de Dados em Preparação

Para além dos dados brutos, foi também criado um *bucket* específico, denominado *tempfiles*, cuja função é acolher todos os ficheiros intermédios necessários ao processamento.

Cada ficheiro armazenado neste *bucket* recebe como nome o identificador da fonte de dados que lhe deu origem, assegurando assim uma correspondência direta e inequívoca entre os dados temporários e a respetiva origem.

Estrutura de Armazenamento no PostgreSQL para Dados Processados

A estratégia adotada para o armazenamento de dados processados, um elemento essencial da camada de *serving data* da arquitetura. Com o objetivo de suportar uma plataforma de visualização robusta, foi desenvolvido um modelo de dados orientado à eficiência na consulta, à flexibilidade e à governança de um ecossistema de dados heterogéneo. A Figura 24 apresenta a organização proposta, baseada em duas tabelas principais data_source e data complementadas por uma camada de *data marts* [40].

A tabela data_source funciona como um catálogo central de metadados, reunindo informação crítica sobre cada fonte de dados que alimenta o *pipeline*. Esta estrutura é fundamental para a governação dos dados, permitindo identificar a proveniência, frequência de atualização, domínio, descrição e o esquema esperado de cada conjunto de dados turísticos processados.

Por sua vez, a tabela data serve como repositório central dos dados turísticos já submetidos ao processo de ETL e prontos para consumo. Cada registo representa um conjunto específico de dados processados.

A camada de *data marts* é construída a partir da tabela data e agrega subconjuntos de dados otimizados para visualização e análise. Estes *data marts* são concebidos para responder a necessidades específicas de negócio, através da pré-agregação, sumarização e estruturação dos dados, o

que permite maximizar o desempenho das ferramentas de visualização e reduzir significativamente os tempos de resposta [40].

Para os utilizadores finais e analistas, os *data marts* oferecem uma visão temática e simplificada dos dados, abstraindo a complexidade dos formatos brutos ou semi-estruturados e disponibilizando estruturas alinhadas com as suas necessidades analíticas.

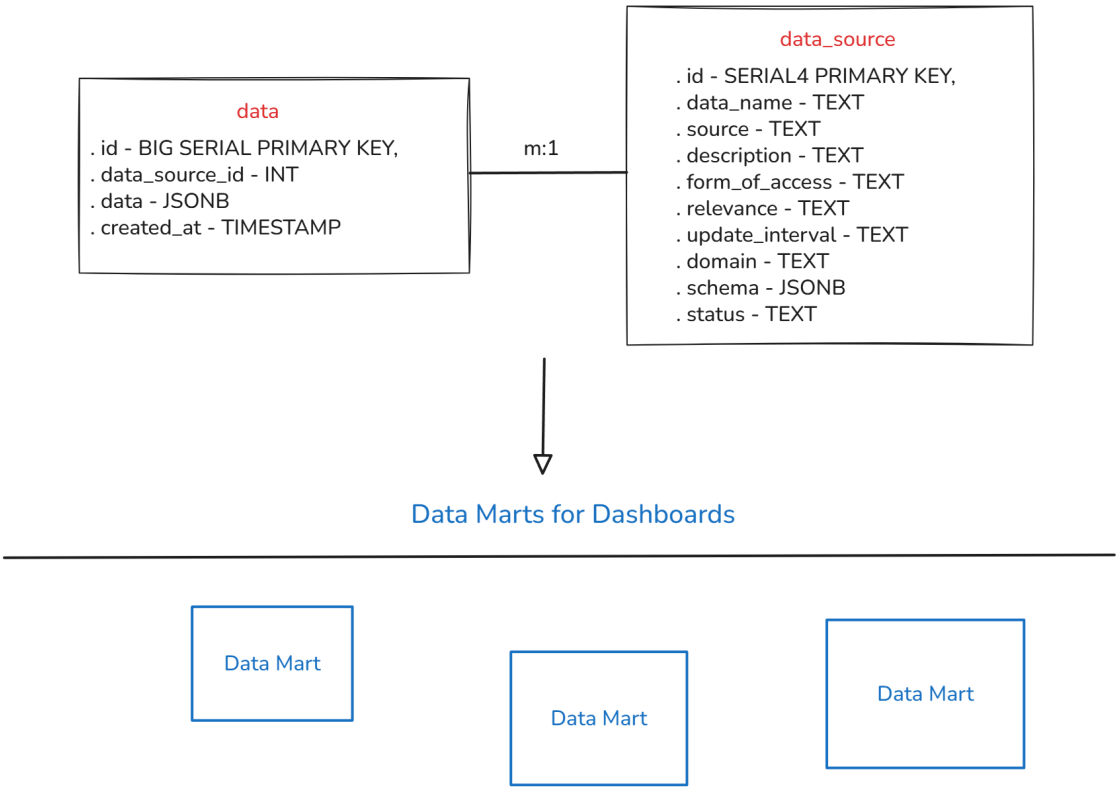

Figura 24: Modelo de Armazenamento de dados - PostgreSQL

Neste capítulo, foi descrito em detalhe o processo de desenvolvimento e implementação da arquitetura proposta, desde a configuração das infraestruturas e ferramentas até à definição dos *pipelines* de processamento e modelos de armazenamento de dados.

Na próxima secção, serão apresentados quatro cenários de validação que ilustram, de forma prática, o funcionamento da arquitetura desenvolvida. Cada caso aborda um desafio específico, desde os diferentes tipos de recolha de dados, passando por situações de falha ou perda de informação, até à integração realizada por utilizadores menos experientes e à análise de observabilidade em comparação com a solução SMARTDEST. Estes exemplos não só evidenciam os resultados

obtidos com a integração das novas fontes de dados e a implementação do *pipeline*, como também permitem avaliar a eficácia da solução proposta e a sua aplicabilidade em cenários reais.

5 Validação da Arquitetura

Neste capítulo, procede-se à análise da eficácia da arquitetura proposta através de diferentes cenários de validação que refletem cenários reais de integração e processamento de dados no setor do turismo. O primeiro caso centra-se na comparação entre metodologias de recolha de informação, incluindo APIs, *web scraping* e fontes documentais abertas. O segundo aborda o impacto de falhas na ingestão e perda de dados, avaliando a resiliência da arquitetura. Por fim, o terceiro caso analisa a evolução dos mecanismos de observabilidade, comparando a solução atual com a utilizada no projeto SMARTDEST. Com estes casos, pretende-se evidenciar os benefícios e limitações da abordagem seguida, bem como identificar oportunidades de melhoria para reforçar a aplicabilidade prática da metodologia.

5.1 Cenário de Validação 1 : Diferentes tipos de recolha de dados

A diversidade das fontes de dados utilizadas no contexto turístico obriga a conter uma arquitetura capaz de lidar com múltiplos métodos de recolha, cada um com características, vantagens e desafios específicos. Este cenário de validação visa demonstrar a flexibilidade e adaptabilidade da arquitetura implementada, através da comparação entre quatro abordagens distintas de aquisição de dados: acesso via API e ligação direta a bases de dados, técnicas de *web scraping* e consumo de feeds de dados abertos em formatos como CSV e PDF.

Cada uma destas abordagens impõe diferentes exigências ao *pipeline*, tanto ao nível da complexidade de implementação como da sua manutenção e fiabilidade a longo prazo. Ao longo deste cenário de validação, serão analisadas fontes de dados representativas de cada uma destas abordagens, comparando-se tempos de resposta, frequência de falhas, robustez das integrações, e o esforço necessário para garantir a sua manutenção contínua no *pipeline*.

Recolha via API: Weather Warnings (IPMA)

Neste cenário de validação foi integrada a fonte Weather Warning, que utiliza a API pública do IPMA para a recolha periódica de alertas meteorológicos.

Conforme o *pipeline* generalizado apresentado na Secção 4.2.2, a primeira etapa envolveu o registo da *metadata* da fonte. Esta foi armazenada na tabela `data_source` da base de dados PostgreSQL, em conformidade com o modelo descrito na Secção 4.2.3.

Recorrendo ao Apache Airflow, as etapas subsequentes do *pipeline* foram orquestradas, conforme ilustrado na Figura 25 e detalhado no Anexo 7.8.

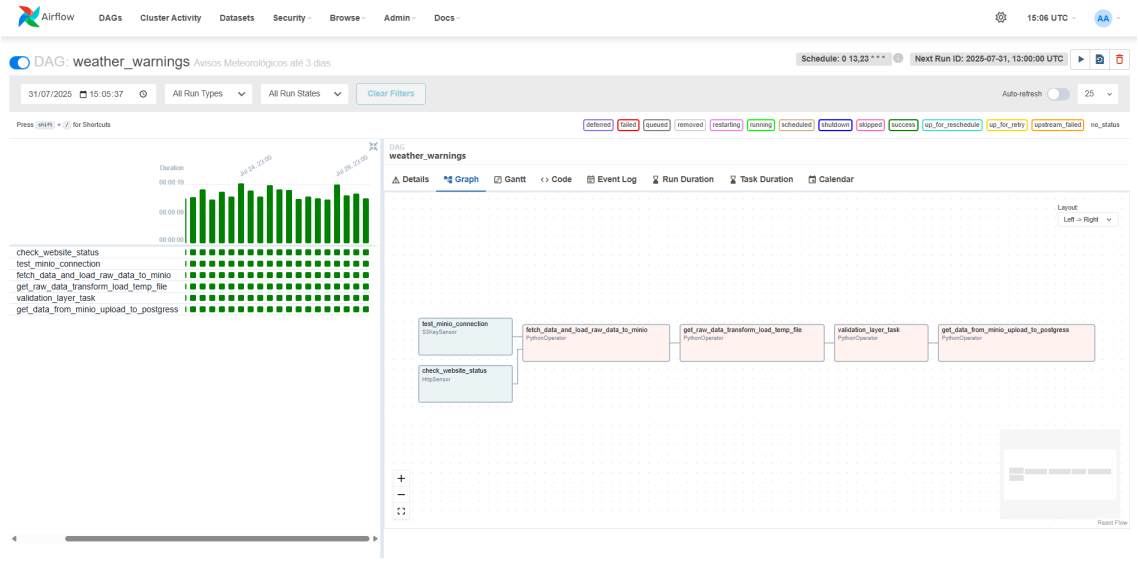

Figura 25: DAG no Airflow para a fonte OOM (IPMA)

O DAG foi configurado para executar automaticamente duas vezes por dia (13:00h e 23:00h) e integra os seguintes passos:

- Verificação da disponibilidade da API e do serviço MinIO, utilizando `HttpSensor` e `S3KeySensor`, respetivamente com execução em paralelo;
- Extração dos dados em formato JSON, no seu estado bruto, através do `HttpHook` e armazenamento inicial no `Raw Data Storage` no MinIO (Figura 26);
- Limpeza e transformação dos dados, removendo inconsistências ou campos vazios e posterior armazenamento dos dados processados no *Temporary Data Storage* no MinIO;
- Validação dos dados transformados com base em esquemas definidos através da biblioteca `pandera`, garantindo a integridade e coerência dos dados como por exemplo ordenação temporal entre datas de início e fim dos alertas;
- Armazenamento final dos dados validados na base de dado PostgreSQL (Figura 27)

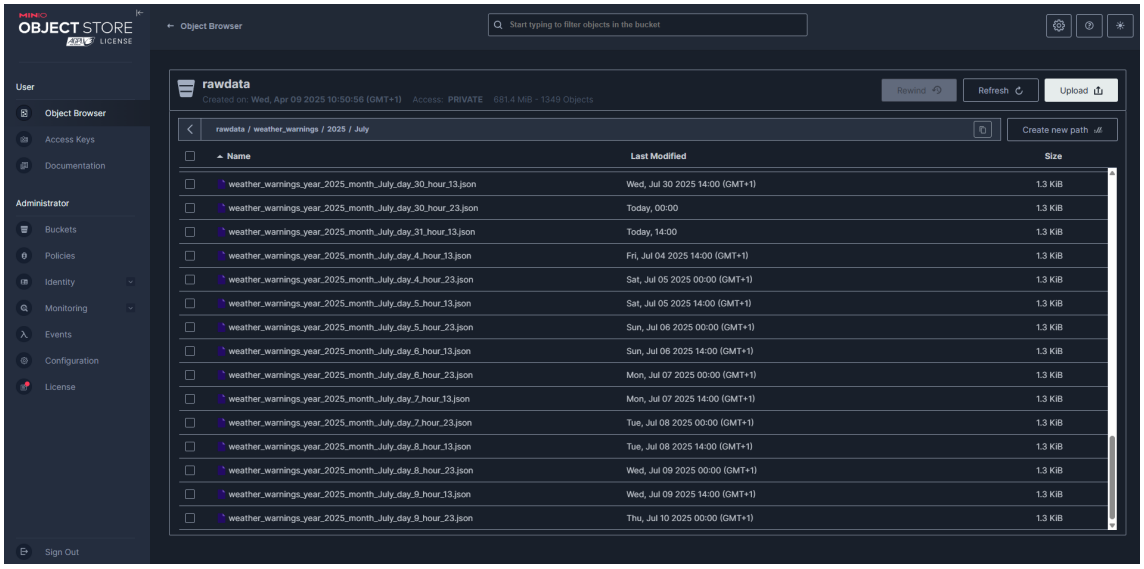

Figura 26: Armazenamento Intermédio no MinIO – Fonte OOM (IPMA)

	id	data_source_id	data	created_at
1	12,565	2	{"endTime": "2025-05-06T20:00:00", "startTime": "2025-05-03T20:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Nevoeiro"}	2025-05-03 23:00:16.3
2	12,566	2	{"endTime": "2025-05-06T20:00:00", "startTime": "2025-05-03T20:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Quente"}	2025-05-03 23:00:16.3
3	12,567	2	{"endTime": "2025-05-06T20:00:00", "startTime": "2025-05-03T20:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Frio"}	2025-05-03 23:00:16.3
4	12,569	2	{"endTime": "2025-05-06T20:00:00", "startTime": "2025-05-03T20:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Neve"}	2025-05-03 23:00:16.3
5	12,570	2	{"endTime": "2025-05-06T20:00:00", "startTime": "2025-05-03T20:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Trovoada"}	2025-05-03 23:00:16.3
6	12,571	2	{"endTime": "2025-05-06T20:00:00", "startTime": "2025-05-03T20:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-05-03 23:00:16.3
7	28,918	2	{"endTime": "2025-06-08T18:00:00", "startTime": "2025-06-05T18:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Trovoada"}	2025-06-05 23:00:16.6
8	28,919	2	{"endTime": "2025-06-08T18:00:00", "startTime": "2025-06-05T18:32:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-06-05 23:00:16.6
9	29,719	2	{"endTime": "2025-06-11T06:00:00", "startTime": "2025-06-08T06:07:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Trovoada"}	2025-06-08 13:00:17.1
10	29,720	2	{"endTime": "2025-06-11T06:00:00", "startTime": "2025-06-08T06:07:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-06-08 13:00:17.1
11	30,351	2	{"endTime": "2025-06-13T06:00:00", "startTime": "2025-06-10T06:21:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Neve"}	2025-06-10 13:00:17.4
12	30,352	2	{"endTime": "2025-06-13T06:00:00", "startTime": "2025-06-10T06:21:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Trovoada"}	2025-06-10 13:00:17.4
13	30,353	2	{"endTime": "2025-06-13T06:00:00", "startTime": "2025-06-10T06:21:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-06-10 13:00:17.4
14	30,811	2	{"endTime": "2025-06-14T11:00:00", "startTime": "2025-06-11T11:01:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-06-11 23:00:16.6
15	31,915	2	{"endTime": "2025-06-18T11:00:00", "startTime": "2025-06-15T11:40:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-06-15 13:00:15.7
16	12,036	2	{"endTime": "2025-05-05T10:00:00", "startTime": "2025-05-02T10:27:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Nevoeiro"}	2025-05-02 13:00:14.4
17	32,551	2	{"endTime": "2025-06-20T06:00:00", "startTime": "2025-06-17T06:36:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-06-17 13:00:17.2
18	12,037	2	{"endTime": "2025-05-05T10:00:00", "startTime": "2025-05-02T10:27:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Quente"}	2025-05-02 13:00:14.4
19	12,038	2	{"endTime": "2025-05-05T10:00:00", "startTime": "2025-05-02T10:27:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Frio"}	2025-05-02 13:00:14.4
20	22,017	2	{"endTime": "2025-05-14T10:00:00", "startTime": "2025-05-11T10:54:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Neve"}	2025-05-11 13:00:16.2
21	22,018	2	{"endTime": "2025-05-14T10:00:00", "startTime": "2025-05-11T10:54:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Trovoada"}	2025-05-11 13:00:16.2
22	12,040	2	{"endTime": "2025-05-05T10:00:00", "startTime": "2025-05-02T10:27:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Neve"}	2025-05-02 13:00:14.4
23	20,930	2	{"endTime": "2025-05-11T12:00:00", "startTime": "2025-05-08T12:25:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Quente"}	2025-05-08 23:00:14.5
24	12,041	2	{"endTime": "2025-05-05T10:00:00", "startTime": "2025-05-02T10:27:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Trovoada"}	2025-05-02 13:00:14.4
25	20,931	2	{"endTime": "2025-05-11T12:00:00", "startTime": "2025-05-08T12:25:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Frio"}	2025-05-08 23:00:14.5
26	12,042	2	{"endTime": "2025-05-05T10:00:00", "startTime": "2025-05-02T10:27:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-05-02 13:00:14.4
27	27,111	2	{"endTime": "2025-05-26T00:00:00", "startTime": "2025-05-23T00:04:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Nevoeiro"}	2025-05-23 13:00:16.0
28	8,658	2	{"endTime": "2025-04-27T12:00:00", "startTime": "2025-04-24T12:14:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Nevoeiro"}	2025-04-24 17:13:38.5
29	8,659	2	{"endTime": "2025-04-27T12:00:00", "startTime": "2025-04-24T12:14:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Quente"}	2025-04-24 17:13:38.5
30	27,112	2	{"endTime": "2025-05-26T00:00:00", "startTime": "2025-05-23T00:04:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Quente"}	2025-05-23 13:00:16.0
31	8,660	2	{"endTime": "2025-04-27T12:00:00", "startTime": "2025-04-24T12:14:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Frio"}	2025-04-24 17:13:38.5
32	8,661	2	{"endTime": "2025-04-27T12:00:00", "startTime": "2025-04-24T12:14:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Precipitação"}	2025-04-24 17:13:38.5
33	8,662	2	{"endTime": "2025-04-27T12:00:00", "startTime": "2025-04-24T12:14:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Neve"}	2025-04-24 17:13:38.5
34	10,587	2	{"endTime": "2025-05-01T14:00:00", "startTime": "2025-04-28T14:45:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Nevoeiro"}	2025-04-28 23:00:13.2
35	8,663	2	{"endTime": "2025-04-27T12:00:00", "startTime": "2025-04-24T12:14:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Trovoada"}	2025-04-24 17:13:38.5
36	10,588	2	{"endTime": "2025-05-01T14:00:00", "startTime": "2025-04-28T14:45:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Quente"}	2025-04-28 23:00:13.2
37	8,664	2	{"endTime": "2025-04-27T12:00:00", "startTime": "2025-04-24T12:14:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Vento"}	2025-04-24 17:13:38.5
38	10,589	2	{"endTime": "2025-05-01T14:00:00", "startTime": "2025-04-28T14:45:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Tempo Frio"}	2025-04-28 23:00:13.2
39	10,591	2	{"endTime": "2025-05-01T14:00:00", "startTime": "2025-04-28T14:45:00", "idAreaAviso": "MCS", "awarenessLevelID": "green", "awarenessTypeName": "Neve"}	2025-04-28 23:00:13.2

Figura 27: Dados Persistidos no PostgreSQL – Fonte OOM (IPMA)

Esta abordagem evidencia um conjunto de vantagens específicas da recolha de dados via API, sobretudo pela previsibilidade e pela consistência estrutural da informação disponibilizada. A integração com o Apache Airflow reforça estes benefícios, uma vez que os seus *Sensors* e *Hooks* nativos

reduzem de forma significativa o esforço de implementação e manutenção. Adicionalmente, a resiliência do *pipeline* é assegurada através da configuração explícita de parâmetros como `retries` e `retry_delay`, que permitem lidar eficazmente com falhas temporárias, interrupções de rede ou dados malformados, evitando que erros ocasionais comprometam o fluxo global.

Apesar destas vantagens, a integração via API não está isenta de limitações. A principal prende-se com a dependência da disponibilidade contínua do serviço externo do IPMA, bem como com a possibilidade de alterações futuras na versão da API ou na estrutura dos dados. Ainda assim, os mecanismos de tolerância a falhas previamente descritos e a validação da estrutura dos dados antes do armazenamento final contribuem para garantir a robustez e a estabilidade do *pipeline* perante situações inesperadas.

No que respeita à avaliação prática, esta fonte de dados tem apresentado uma execução regular e estável desde abril de 2025, comprovando a fiabilidade da integração. A validação sistemática assegura a coerência e a qualidade da informação recolhida, enquanto os mecanismos de recuperação de falhas reforçam a resiliência do sistema. No seu conjunto, este caso demonstra que a integração de dados via API constitui uma solução estável e de baixa manutenção no âmbito da arquitetura implementada.

Recolha via Web Scrapping : Flight Offers (eDreams)

A recolha de dados através de técnicas de *web scraping* revela-se particularmente pertinente em cenários nos quais não existem APIs públicas devidamente estruturadas. Neste estudo de caso, recorreu-se à fonte Flight Offers, através do *website* da eDreams, com o objetivo de recolher diariamente os itinerários de voos que ligam Funchal a Lisboa, Porto, Porto Santo, Londres e vice-versa.

A orquestração do *pipeline* seguiu a lógica previamente descrita no caso anterior, distinguindo-se, contudo, pelo método de extração: neste caso, via *Web Scraping* com *Selenium* em modo remoto. Para tal, foi configurado um serviço em Docker (Anexo 7.1), permitindo a criação de sessões diretamente no *website* a partir do Airflow. O DAG foi agendado para execução diária às 10:00h, coordenando em paralelo a recolha entre os diferentes destinos.

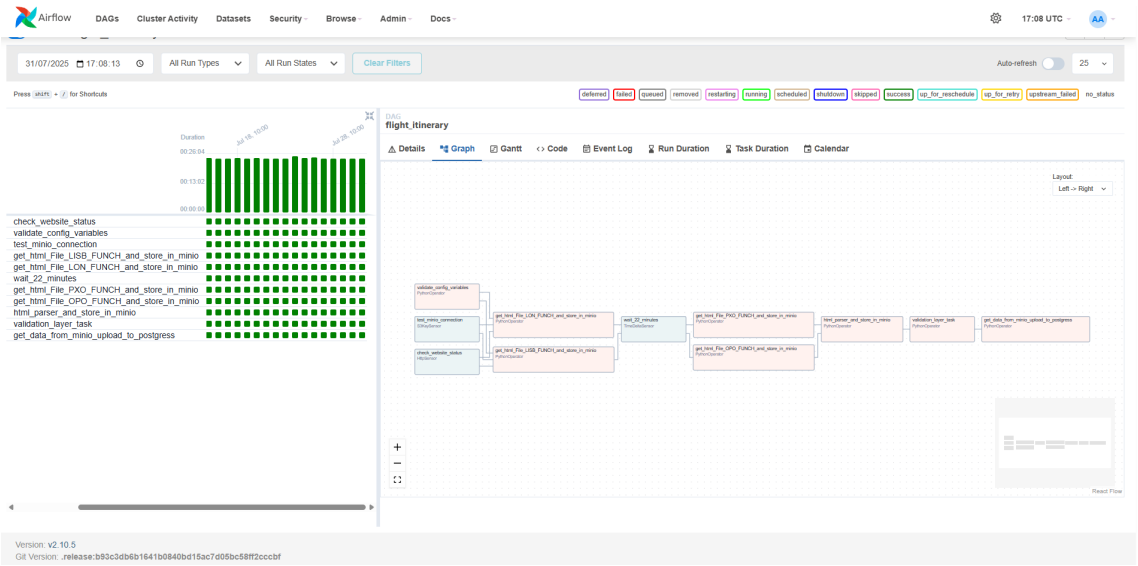

Figura 28: Airflow – Fonte Flight Offers (eDreams)

Ao contrário da integração por API, onde os dados são disponibilizados de forma estruturada e estável, a recolha via *web scraping* implica uma maior complexidade, quer no momento da extração, quer na fase de processamento. Apesar disso, as etapas de validação de parâmetros, verificação da integridade dos dados e armazenamento nas bases de dados seguem o mesmo modelo das integrações por API, sendo o mecanismo de recolha a principal diferença.

O processo inicia-se com a utilização do Selenium WebDriver em modo remoto, que possibilita a interação com páginas dinâmicas dependentes de JavaScript, incluindo operações como *scrolling* e aceitação de *cookies*. Os ficheiros HTML resultantes são armazenados no *bucket* rawdata do sistema MinIO, organizados por data e por pares origem-destino (Figura 29).

Posteriormente, procede-se ao *parsing* dos ficheiros HTML com recurso à biblioteca BeautifulSoup, extraindo os itinerários e respetivos atributos, como preços, horários, duração, escalas, franquias de bagagem e companhias aéreas. A partir desta fase, o *pipeline* segue as mesmas etapas definidas para as integrações por API.

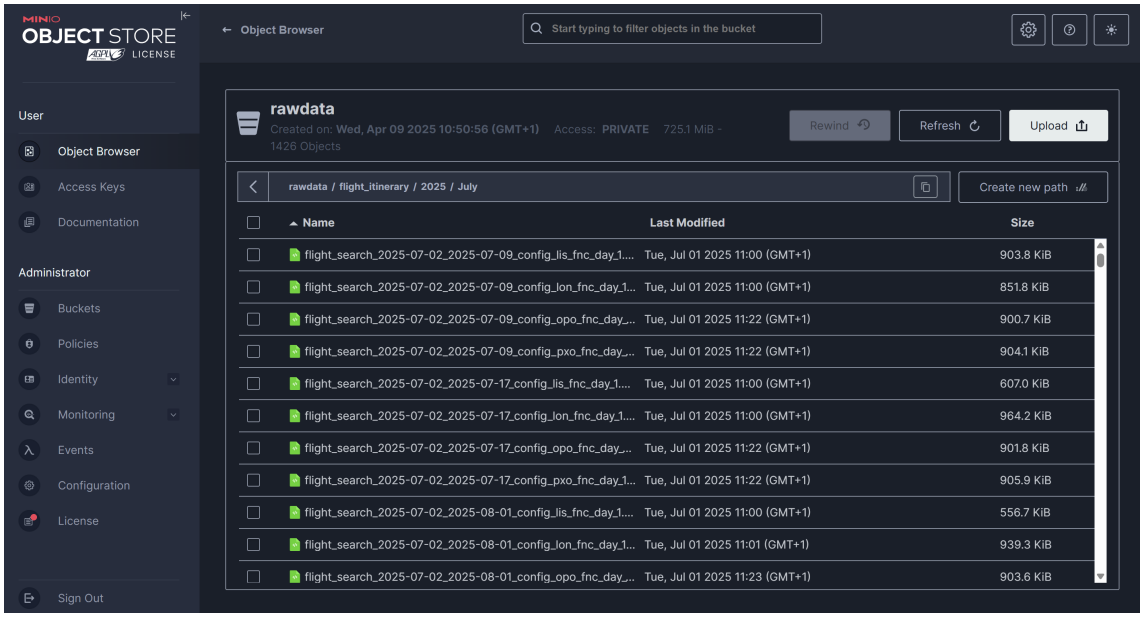

Figura 29: Armazenamento Intermédio no MinIO – Fonte Flight Offers (eDreams)

Entre as principais vantagens deste método destaca-se o acesso a informação que, de outra forma, não estaria disponível de forma estruturada ou programática, permitindo enriquecer a arquitetura de dados com novas fontes de elevado valor analítico. A integração entre Airflow e Selenium garante ainda flexibilidade e capacidade de adaptação a diferentes cenários, assegurando a continuidade da recolha mesmo perante páginas dinâmicas que exigem renderização de JavaScript e interações complexas. Estas características contribuem significativamente para a robustez e escalabilidade da solução.

Contudo, a adoção do *web scraping* acarreta limitações relevantes. A dependência de elementos visuais e estruturais torna o sistema vulnerável a alterações súbitas na interface do *website*, implicando custos adicionais de manutenção.

No que respeita à avaliação prática esta fonte encontra-se em operação desde 6 de maio de 2025 e, até à data de redação deste documento, registaram-se dois incidentes significativos que interromperam temporariamente a recolha automática. O primeiro esteve relacionado com a apresentação recorrente de *captchas* pelo *website* da eDreams, inviabilizando a progressão do processo automatizado. Para mitigar este problema, foi introduzido um intervalo fixo de 22 minutos entre execuções paralelas, reduzindo assim a frequência de acessos simultâneos (Figura 28). O segundo incidente decorreu de alterações na estrutura do HTML e das classes de CSS utilizadas pela eDre-

ams, exigindo uma atualização dos seletores no código de extração para manter a compatibilidade com a nova versão da página.

Em síntese, este caso evidencia que a fiabilidade na recolha de dados por *web scraping* não é imediata, mas sim alcançável. Apesar da instabilidade inerente causada por alterações na interface e mecanismos como *captchas*, é possível assegurar a continuidade do processo através de uma arquitetura bem estruturada, modular e monitorizável. Todavia, importa ressalvar que esta fiabilidade operacional tem um custo, exige uma manutenção e monitorização contínua substancialmente superiores às de uma integração via API, dado que a robustez do sistema depende da sua constante adaptação a elementos visuais voláteis.

Recolha de dados através de Fontes Documentais Abertas : DREM

A recolha de dados a partir de Fontes Documentais Abertas constitui uma abordagem centrada na extração de documentos públicos disponibilizados de forma manual ou semi-estruturada, geralmente em formatos como PDF ou Excel, habitualmente acessíveis em *websites* institucionais ou plataformas de acesso aberto.

Este tipo de integração revela-se particularmente útil em contextos em que a fonte de dados não disponibiliza uma API, nem permite uma extração automatizada e contínua. Nestes casos, a informação relevante é publicada de forma pontual sob a forma de documentos, exigindo um processo específico de recolha e integração.

Ao contrário dos métodos baseados em APIs ou em técnicas de *web scraping*, esta abordagem caracteriza-se pela sua natureza estática e esporádica. A recolha de dados ocorre apenas uma vez ou sempre que novos ficheiros são disponibilizados pela entidade responsável.

O cenário de validação aqui analisado incide sobre a recolha de dados a partir de relatórios públicos fornecidos pela Direção Regional de Estatística da Madeira (DREM), disponibilizados em formatos Excel e PDF. Estes documentos contêm informação estatística relevante sobre o setor do turismo na Região Autónoma da Madeira, mas apresentam desafios adicionais, uma vez que a estrutura interna varia entre ficheiros, exigindo processos de extração, limpeza e transformação personalizados.

Tal como nas abordagens anteriormente descritas, esta fonte seguiu o *pipeline* genérico definido na Secção 4.2.2, com a exceção de que, neste cenário, a utilização do Apache Airflow é opcional ou mesmo dispensável. O processo de ETL pode ser concretizado de duas formas distintas:

- **Execução via Apache Airflow**: É definido um DAG configurado para permanecer inativo por defeito, sendo apenas acionado manualmente quando necessário.
- **Execução através de Script Autónomo**: Consiste na utilização de um script independente, desenvolvido em Python, que realiza pontualmente as etapas de extração, transformação e carregamento, sem qualquer integração com o Apache Airflow.

Independentemente da abordagem escolhida, é essencial assegurar o cumprimento do *pipeline* genérico descrito na Secção 4.2.2.

Para ilustrar estas duas estratégias, foram integradas duas fontes distintas:

- **Transit Passengers at Port by Nationality**: processada com recurso ao Apache Airflow. Os ficheiros CSV, referentes ao período de 2018 a 2023, foram transferidos e colocados no diretório `/apache_airflow/downloads`, previamente mapeado como volume partilhado no ficheiro `docker-compose.yaml`, permitindo o acesso dos *Dags* do Airflow. A partir deste ponto, o processo de ETL seguiu a mesma lógica aplicada à extração por API, com a diferença de que os dados foram lidos localmente a partir de ficheiros em vez de obtidos através de um *endpoint*.
- **Transit Passengers at Port**: tratada através de um script autónomo em Python, que igualmente recolheu ficheiros CSV referentes ao mesmo período (2018–2023). Este script estabeleceu ligação às instâncias da base de dados PostgreSQL e ao sistema de armazenamento MinIO, executando o *pipeline* de ETL até ao armazenamento final dos dados estruturados na camada de *serving data*.

Entre as principais vantagens desta abordagem destaca-se a sua simplicidade técnica, uma vez que os processos podem ser executados de forma autónoma através de scripts ou, em alternativa, integrados no Airflow, garantindo um baixo custo computacional e reduzida complexidade de orquestração.

Contudo, apresentam-se também limitações significativas. A heterogeneidade dos formatos frequentemente disponibilizados em PDF, Excel ou CSV com estruturas inconsistentes implica mai-

ores esforços de pré-processamento, incluindo tarefas de limpeza, normalização e transformação adaptadas a cada documento.

Na prática, esta abordagem, embora pouco exigente em termos de infraestruturas e de recursos computacionais, enfrenta desafios relacionados com a diversidade e variabilidade dos ficheiros. Ainda assim, uma vez concluído o processo de integração, a manutenção necessária é bastante reduzida, não exigindo monitorização contínua. Assim, adapta-se de forma eficiente no âmbito da arquitetura implementada.

Conclusão

Este cenário de validação permitiu evidenciar a importância de uma arquitetura de dados flexível e modular, capaz de integrar diferentes métodos de recolha de dados com origens e características distintas. Através da análise comparativa entre as abordagens por API, *web scraping* e Fontes Documentais Abertas, demonstrou-se que cada método impõe requisitos técnicos próprios, implicando níveis variados de esforço de implementação, manutenção e robustez.

A integração via API revelou-se a mais eficiente e fiável, com elevada automatização e baixos custos de manutenção, beneficiando de dados bem estruturados e acessos estáveis. Por outro lado, o *web scraping*, embora essencial em contextos sem interfaces programáticas, demonstrou ser mais vulnerável a alterações externas e exigente do ponto de vista de monitorização e adaptação contínua. Já a abordagem baseada em fontes documentais abertas, apesar de menos automatizável, mostrou-se valiosa para a integração de dados estatísticos e institucionais, proporcionando uma solução viável para contextos de atualização menos frequente.

Em conjunto, estas três estratégias ilustram a versatilidade da arquitetura proposta, validando a sua capacidade de responder de forma eficaz aos diferentes desafios impostos pela diversidade das fontes de dados no setor do turismo. Esta flexibilidade é essencial para assegurar um *pipeline* resiliente, escalável e adaptado à realidade heterogénea das fontes de informação disponíveis.

5.2 Cenário de Validação 2 : Falha na recolha ou perda de dados

A recolha e a gestão de dados constituem a base de qualquer arquitetura de *big data*, assumindo particular relevância em setores onde a informação é gerada de forma contínua e em grande escala, como sucede no turismo. No entanto, a fiabilidade e a integridade destes dados encontram-se

permanentemente expostas a riscos, sendo a falha na recolha ou a perda de informação um dos problemas mais críticos, com impacto direto na eficácia e robustez dos sistemas.

No contexto da plataforma desenvolvida no âmbito do projeto SMARTDEST, dedicada ao tratamento e análise de dados turísticos, foram identificadas limitações significativas relacionadas com a falha na recolha e a perda de informação. A arquitetura anteriormente utilizada apresentava fragilidades estruturais, na qual qualquer falha ocorrida em etapas posteriores ao processo de recolha resultava na perda completa dos dados, ainda que a fase inicial de ingestão tivesse sido bem-sucedida. Esta baixa resiliência do processo de ETL originava perdas recorrentes de dados e comprometia a rastreabilidade das operações.

Adicionalmente, foram observados problemas associados a falhas temporárias de rede ou da infraestrutura tecnológica. Nessas circunstâncias, também se verificava a perda de dados, uma vez que não estavam implementados mecanismos de re-tentativa automática, de validação da integridade das fontes ou de verificação do estado da infraestrutura antes do início da recolha. A inexistência destes mecanismos aumentava substancialmente a vulnerabilidade do sistema a falhas transitórias.

O objetivo deste cenário de validação é, portanto, analisar de que forma a arquitetura reestruturada procura superar estas limitações, aumentando a resiliência global do sistema e garantindo maior fiabilidade e continuidade na recolha e gestão dos dados.

Para mitigar os problemas identificados, foram introduzidas duas medidas centrais. A primeira consistiu na implementação de uma política de re-tentativas automáticas, definindo-se pelo menos duas tentativas adicionais para cada *task* do *pipeline*, com um intervalo de 10 minutos entre cada execução. Este mecanismo foi concebido para lidar com falhas temporárias, como erros de conexão, permitindo que a operação fosse reexecutada com sucesso após um curto período de espera. Por exemplo, conforme ilustrado nas Figuras 30 e 31, no dia 18 de junho de 2025 ocorreu um *connection timeout* no *Dag* `docks_booking` durante a inserção de dados na base de dados *serving data*. Graças ao mecanismo de *retry*, a tarefa foi reexecutada 10 minutos mais tarde e os dados foram corretamente inseridos. Importa ainda salientar que, caso esta *task* falhasse em todas as tentativas de reexecução, as tarefas subsequentes não seriam executadas e a execução do *Dag* seria considerada como falhada.

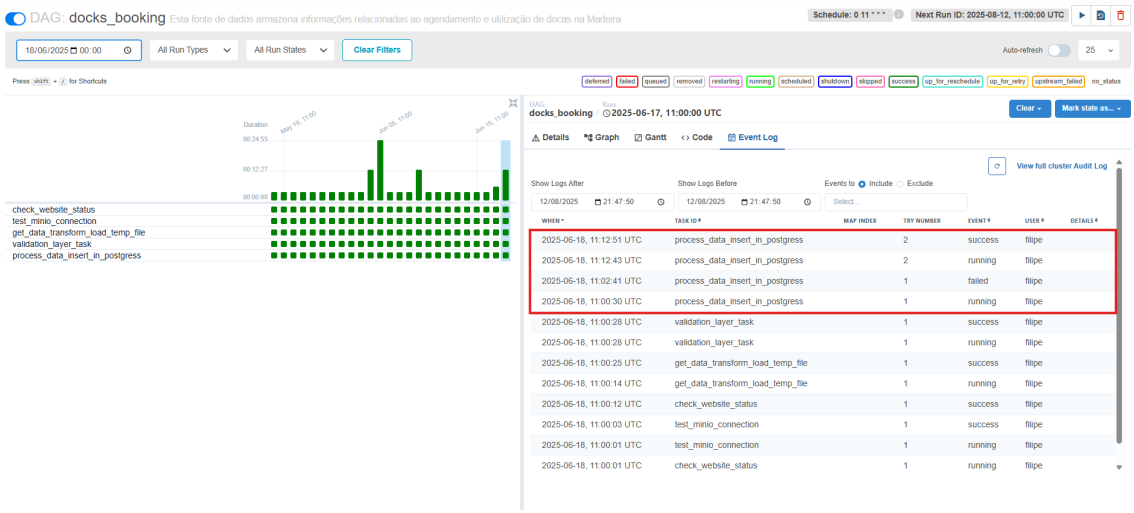

Figura 30: Falha na inserção da fontes de dados Dock Scheduling por timeout - Logs

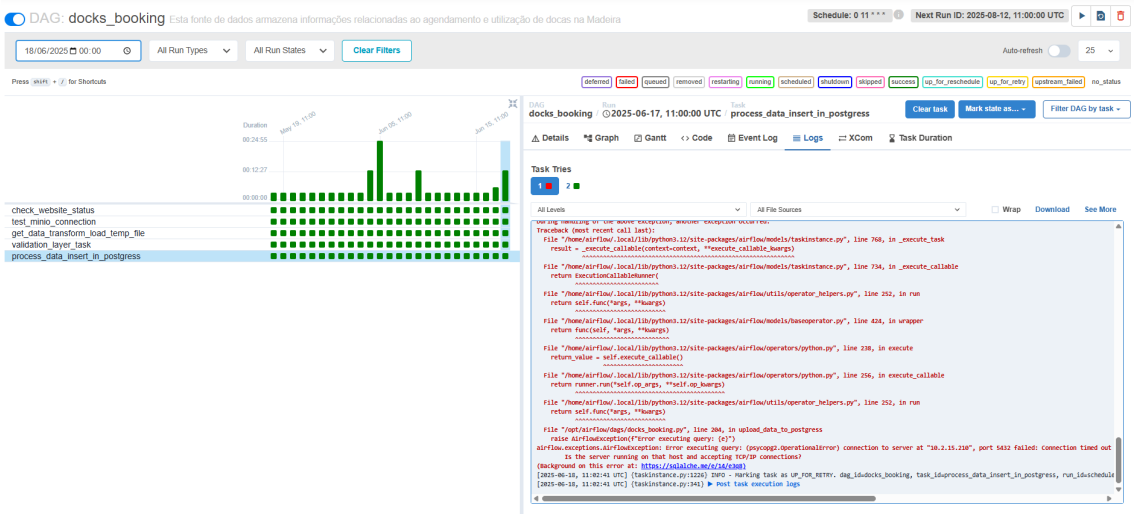

Figura 31: Sucesso na inserção da fontes de dados Dock Scheduling através do segundo retry

A segunda medida consistiu na introdução de uma camada de armazenamento de *raw data*. Este mecanismo permite preservar os dados no seu estado bruto, garantindo que a informação não se perde em caso de falha durante o processamento ou na fase de inserção no repositório de *Serving Data*. Como exemplo, na Figura 32 observa-se que, a 3 de junho de 2025, ocorreu um erro na camada de validação devido a alterações na página web do Edreams, o que resultou numa falha no processamento e, subsequentemente, na validação dos dados. No entanto, dado que os dados brutos já haviam sido armazenados em fases anteriores, não se verificou qualquer perda de informação. Numa situação futura, bastará reprocessar os dados armazenados para recuperar a continuidade do sistema.

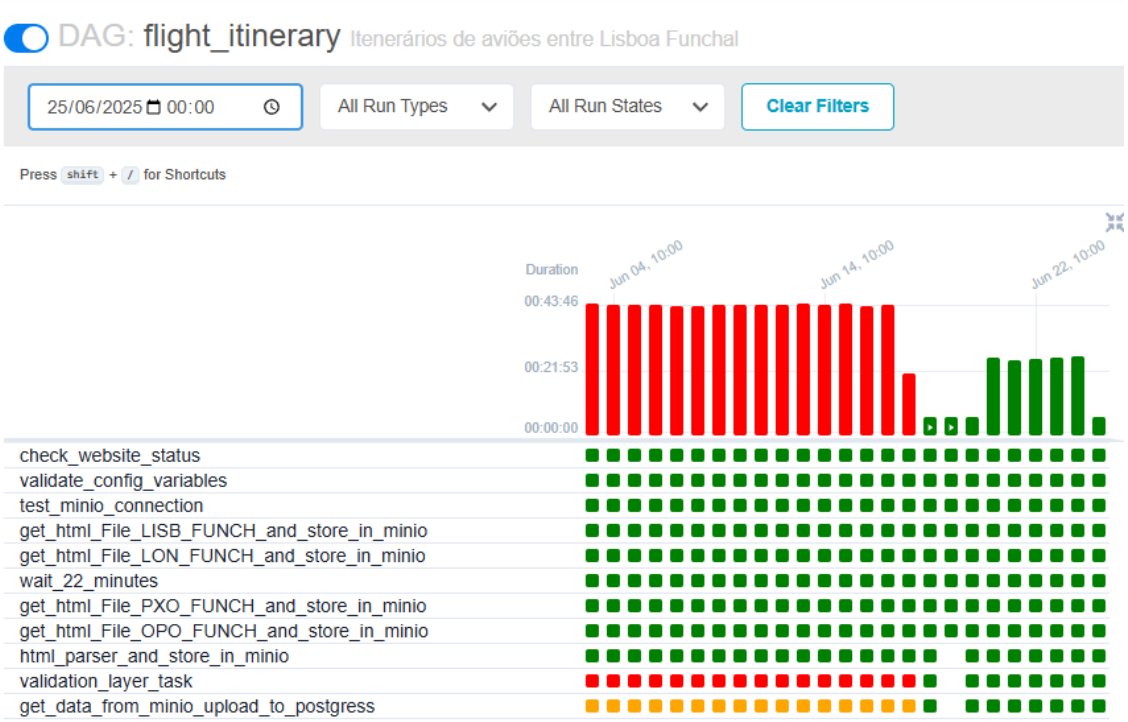

Figura 32: Erro no processamento de dados mitigado pelo armazenamento de raw data

Em síntese, este cenário de validação demonstra que a arquitetura reestruturada apresenta uma melhoria substancial face à anterior no tratamento de falhas relacionadas com a recolha e a perda de dados. A introdução de mecanismos de re-tentativa automática e de armazenamento de dados brutos garantiu que falhas momentâneas de rede, interrupções de serviços externos ou erros no processamento deixassem de resultar em perdas irreversíveis de informação. Estas alterações tornaram o sistema mais resiliente, assegurando a continuidade da recolha de dados e preservando a sua integridade mesmo perante falhas transitórias ou problemas inesperados. A nova arquitetura revelou-se, assim, mais robusta e fiável, conseguindo mitigar os principais riscos previamente identificados.

5.3 Cenário de Validação 3 : Observabilidade e comparação com SMARTDEST

A monitorização eficaz de uma arquitetura de dados constitui um elemento essencial para garantir a fiabilidade, a qualidade e a disponibilidade contínua dos serviços. O objetivo deste cenário de validação é, precisamente, demonstrar de que forma os problemas de monitorização

identificados no capítulo anterior são abordados e resolvidos através da nova arquitetura proposta, em comparação com a solução anteriormente implementada na plataforma SMARTDEST.

Na sua versão inicial, a plataforma SMARTDEST dispunha de capacidades de monitorização bastante limitadas. A visibilidade operacional restringia-se a um *dashboard* básico, que apenas permitia identificar o sucesso ou insucesso da execução das fontes de dados, a sua classificação como pública ou privada, a possibilidade de desativar ou reiniciar cada fonte, bem como a data da última atualização e o respetivo estado de ativação. Esta abordagem fornecia, portanto, apenas um conjunto reduzido de métricas elementares (Figura 11).

Contudo, uma supervisão eficaz de sistemas de *big data* exige a definição e acompanhamento de um leque mais abrangente de indicadores, que incluam não apenas o estado da infraestrutura, mas também métricas de desempenho e qualidade ao longo dos diferentes processos de ingestão, transformação e disponibilização da informação.

Em contraste, a nova arquitetura introduz um modelo de observabilidade mais moderno e robusto, concebido para proporcionar uma visibilidade detalhada sobre toda a infraestrutura e sobre os processos críticos, permitindo uma monitorização mais completa, proativa e orientada à deteção e resolução de falhas.

Este cenário de validação será dividido em duas secções principais, que abordarão a monitorização do *pipeline* e a monitorização da infraestrutura, sendo que esta última era inexistente na plataforma antiga.

Monitorização do Pipeline

Um dos problemas identificados na antiga plataforma SMARTDEST dizia respeito à ausência de informação detalhada no *dashboard* geral, o que limitava a capacidade de supervisão. Em particular, não era possível consultar aspetos relevantes como a periodicidade de execução de cada fonte de dados ou o momento previsto para a sua próxima execução.

Na nova arquitetura, estas limitações foram superadas através da adoção do Apache Airflow, cuja interface de monitorização oferece uma visão muito mais completa sobre o funcionamento da plataforma. Na Figura 33 podemos observar o *dashboard* principal da arquitetura atual, no qual se encontram listadas todas as fontes de dados definidas e orquestradas pelo sistema. Para além da informação já disponível na versão anterior do SMARTDEST, nomeadamente o estado das últimas

execuções (sucesso ou falha), a ativação ou desativação de *pipelines* e a data da última execução, o sistema atual acrescenta um conjunto de funcionalidades avançadas. Entre estas, destaca-se a possibilidade de visualizar, de forma clara, a periodicidade de execução de cada fonte, bem como o dia e a hora previstos para a próxima execução. Adicionalmente, são disponibilizadas estatísticas agregadas, como o número total de tarefas falhadas, o que permite uma avaliação rápida do desempenho operacional e do estado global do sistema.

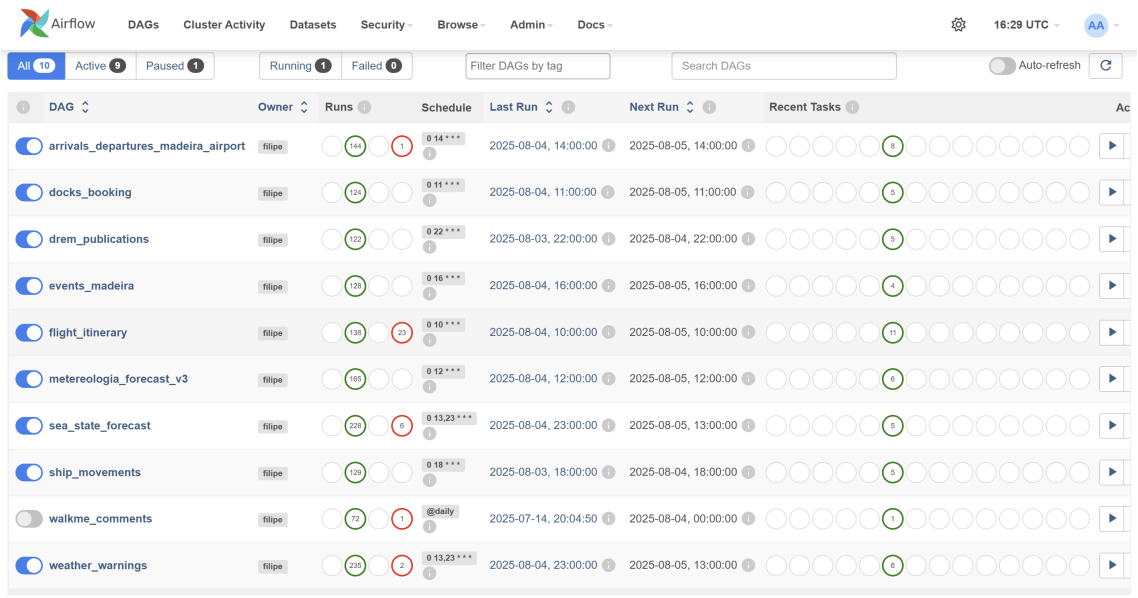

Figura 33: Airflow Dashboard Principal

Um outro problema identificado na arquitetura anterior, no que respeita à monitorização, residia na sua falta de granularidade. A supervisão estava limitada a métricas globais e superficiais, o que dificultava a análise detalhada do comportamento dos diferentes componentes do sistema. Esta limitação comprometia não só a capacidade de diagnosticar a origem de falhas específicas, mas também a possibilidade de avaliar o desempenho de forma rigorosa e identificar potenciais pontos de melhoria.

Na nova arquitetura, esta fragilidade é ultrapassada pela granularidade disponibilizada pelo Apache Airflow, que introduz um nível de observabilidade muito mais aprofundado. Para cada fonte de dados, é agora possível aceder ao histórico completo de execuções, analisar a duração individual de cada tarefa e encontrar falhas com elevada precisão. Esta visibilidade detalhada

representa uma mais-valia significativa, pois permite compreender com clareza o comportamento interno do sistema e fundamentar decisões de otimização com base em dados objetivos.

Esta capacidade é particularmente evidente ao examinar a fonte de dados Flight Offers já analisada anteriormente nos cenários de validação 1 e 2. Podemos observar na Figura 34 que o Airflow permite visualizar uma série de detalhes operacionais e metadados associados à DAG responsável pelo tratamento desta fonte. A secção DAG Runs Summary revela, por exemplo, que foram executadas 25 instâncias da DAG entre 13 de julho e 6 de agosto de 2025, todas concluídas com sucesso. Esta taxa de sucesso de 100 % é indicativa de uma *pipeline* estável e bem configurada. Além disso, o Airflow fornece estatísticas granulares sobre a duração das execuções. Podemos observar na Figura 34 que a execução mais longa registou 24 minutos e 29 segundos, enquanto a mais curta durou apenas 2 minutos e 31 segundos, com uma duração média de aproximadamente 22 minutos e 45 segundos. Esta informação é essencial para avaliar o desempenho e a previsibilidade do processo, permitindo identificar padrões e otimizar eventuais gargalos.

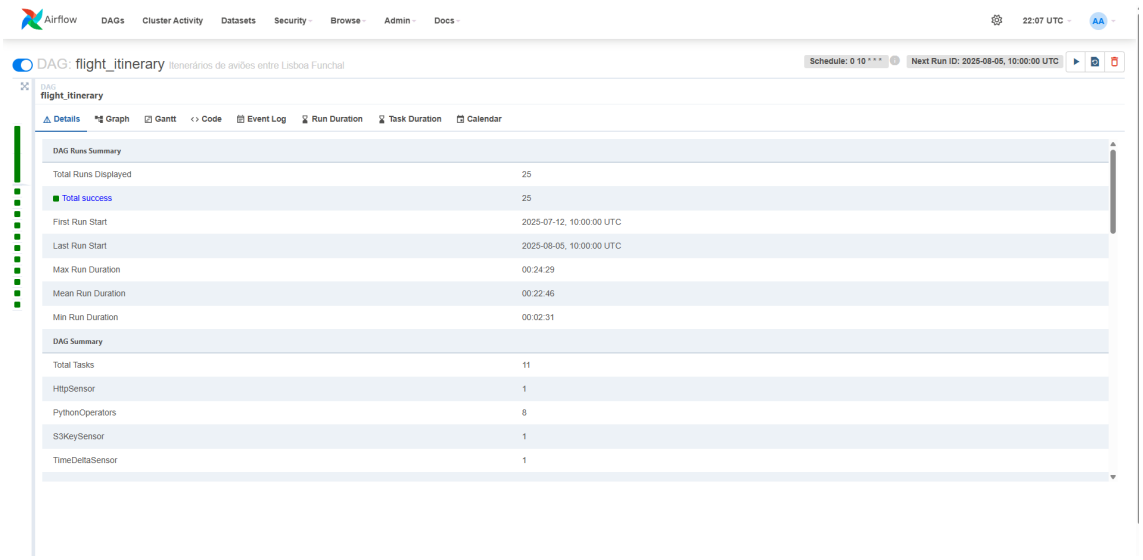

Figura 34: Airflow Detail Metrics

A arquitetura anterior não permitia visualizar a duração das diferentes execuções da DAG, o que dificultava a análise de padrões de desempenho e a deteção de anomalias. Na solução inicialmente adotada, era possível apenas confirmar o sucesso ou a falha de cada execução, sem qualquer detalhe adicional sobre o tempo despendido, o que impedia a avaliação sistemática da estabilidade temporal das *pipelines*.

Na nova arquitetura, este problema foi superado pela funcionalidade de representação gráfica da duração de cada execução, conforme ilustrado na Figura 35. O Airflow permite não só acompanhar as últimas 25 execuções, mas também selecionar intervalos temporais específicos. Esta funcionalidade oferece uma visão imediata da consistência temporal e facilita a identificação de comportamentos fora do padrão. No caso em análise, a maioria das execuções apresenta durações entre 20 e 25 minutos, com uma única exceção pontual devido a um tempo de *landing* invulgarmente elevado. Esta funcionalidade é particularmente relevante para monitorizar visualmente a estabilidade e o desempenho temporal da *pipeline*, permitindo a identificação de desvios que podem sinalizar potenciais anomalias e possibilitando ações de diagnóstico proativas.

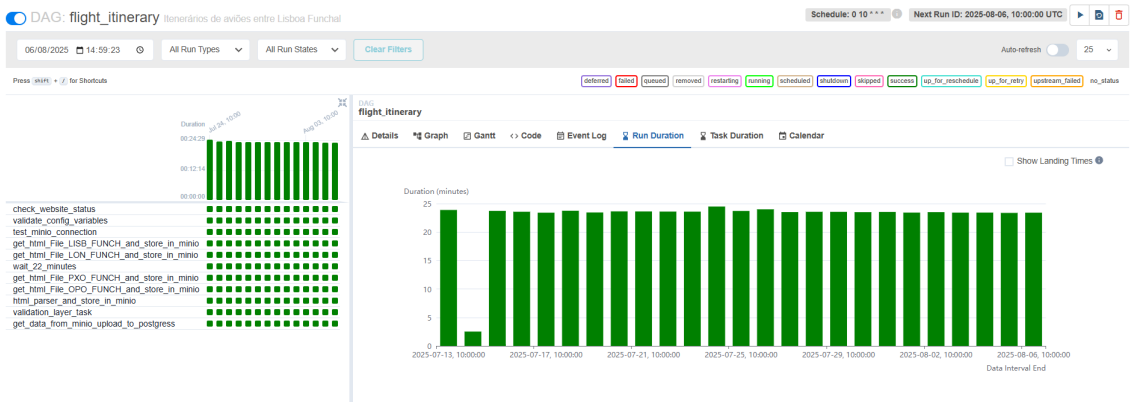

Figura 35: Airflow Run Duration

Outro problema relevante na arquitetura anterior era a ausência de visibilidade sobre o tempo de execução de cada tarefa no processo de ETL. Esta falta de detalhe comprometia o diagnóstico, pois não era possível identificar com precisão em que etapa do fluxo ocorrera um problema ou qual era a tarefa mais crítica ou demorada.

Na nova arquitetura, esta limitação é superada pelas funcionalidades de monitorização do Airflow. A Figura 36 apresenta uma visualização detalhada da duração de cada tarefa de uma DAG, sob a forma de um gráfico de linhas que permite acompanhar a evolução ao longo das últimas 25 execuções ou de um intervalo personalizado.

Para uma análise mais clara, a tarefa com tempo fixo de 22 minutos foi removida do gráfico, o que permite observar com maior nitidez as restantes durações. Verifica-se que as tarefas mais demoradas são as de recolha de dados do *website* da eDreams e de armazenamento de ficheiros

HTML no MinIO, com tempos médios entre 40 segundos e 1 minuto e 30 segundos. Em contraste, as restantes tarefas variam entre 1 segundo e 20 segundos. Este tipo de visualização é fundamental para identificar gargalos, permitindo priorizar otimizações e garantir maior previsibilidade na execução da DAG. Em contextos de produção, esta informação é essencial para assegurar a eficiência e a escalabilidade dos processos de ingestão e processamento de dados.

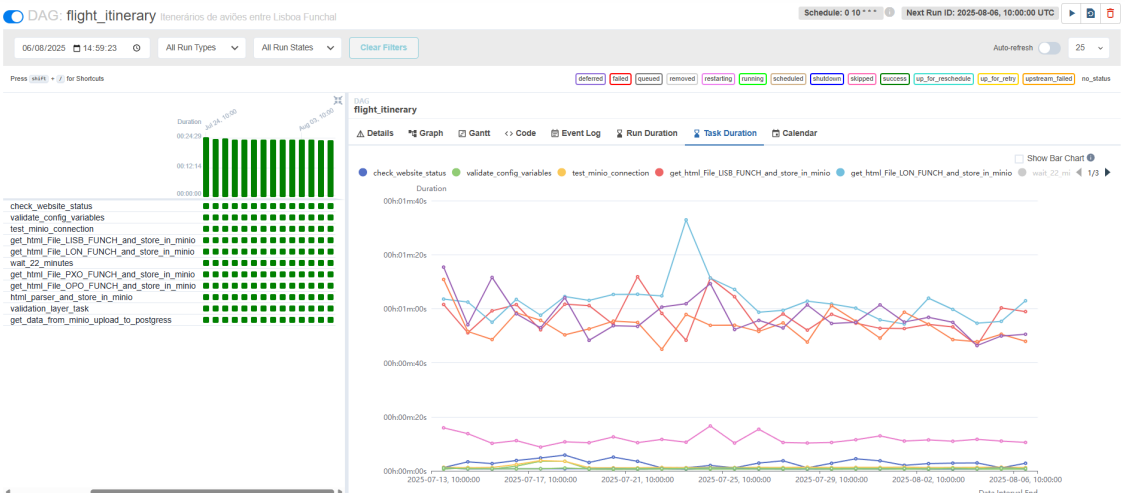

Figura 36: Airflow Task Duration

Uma fragilidade adicional da arquitetura anterior era a impossibilidade de consultar execuções antigas. Na nova arquitetura, esta limitação foi ultrapassada através do calendário de histórico de execuções do Airflow. A Figura 37 ilustra uma representação gráfica do histórico da DAG sob a forma de um calendário anual, que permite identificar, de forma intuitiva, o estado de cada execução diária (bem-sucedida, falhada ou em curso).

Na imagem, é possível verificar que entre os dias 26 de maio e 17 de junho, esta fonte de dados apresentou um estado de erro contínuo, como detalhado no Cenário de Validação 1. Fora desse período, o comportamento da DAG manteve-se estável. Ao oferecer uma visão agregada num formato temporal contínuo, esta funcionalidade facilita a deteção de padrões recorrentes de falhas e a avaliação da fiabilidade e consistência da DAG ao longo de períodos prolongados. É uma ferramenta essencial para validar a robustez do processo e apoiar decisões relacionadas com a manutenção, escalabilidade ou ajustes de agendamento.

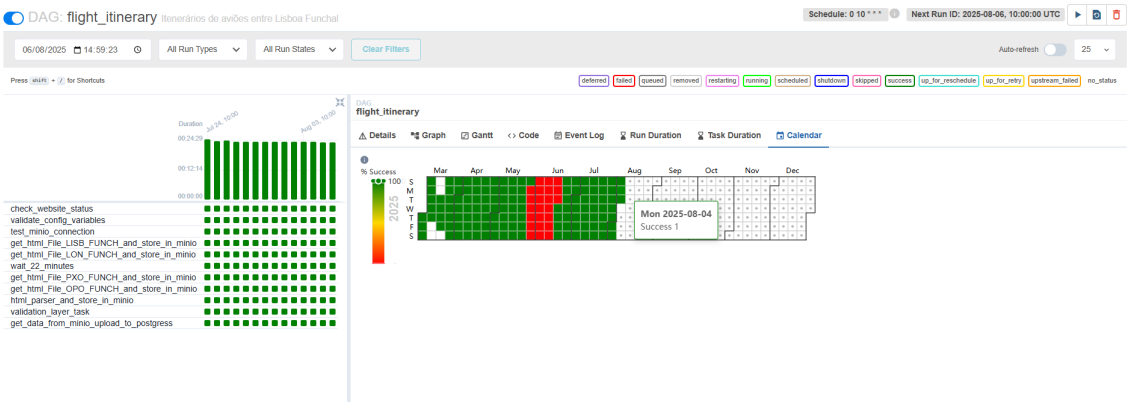

Figura 37: Airflow Calendar

Monitorização da infraestrutura

A monitorização da infraestrutura, uma capacidade inexistente na arquitetura anterior, foi um foco central na nova arquitetura. A arquitetura proposta baseia-se na integração e visualização de métricas de desempenho de cada componente da arquitetura sendo estes o Airflow, MinIO e PostgreSQL, o que permite uma visão holística e granular do sistema.

A monitorização da infraestrutura do Airflow é composta por um conjunto de métricas extraídas pelo Prometehus. Estas métricas fornecem visibilidade abrangente sobre o funcionamento interno do orquestrador, incluindo o número de *Dags* em execução, a duração de tarefas, as falhas por tarefa e o tempo de agendamento.

Dentro dessas métricas foram apenas escolhidas algumas e foram construidos os dashbaords presentes nas Figuras 38 e 39 tendo estes sido divididos nas seguintes categorias :

- **Executor queued tasks:** Na Figura 38, observa-se no canto superior esquerdo o número de tarefas atualmente em fila no *executor*, o qual se encontra a zero, indicando ausência de sobrecarga. Este valor poderá aumentar em situações de elevado número de *tasks* simultâneas, limitações de recursos do *executor* ou atrasos na execução de tarefas dependentes.

- **Executor open slots:** Na Figura 38, no canto inferior esquerdo mostra a disponibilidade de *slots* para execução de tarefas. Os valores constantes perto de 32 demonstram que a infraestrutura está subutilizada e que existe capacidade de resposta imediata a novas execuções.

- **DAG Processing Total Parse Time:** Na Figura 38, no canto superior direto, este gráfico mostra a variação no tempo necessário para processar e interpretar os ficheiros de *Dags*. O

valor médio oscila abaixo de 1 segundo, demonstrando eficiência e ausência de sobrecarga no motor de *parsing*.

- **Operators Successes:** Na Figura 38, no canto inferior direito é apresentado um somatório de execuções bem-sucedidas por tipo de operador, como `PythonOperator`, `S3KeySensor` e `TimeDeltaSensor`, sendo útil para identificar padrões de sucesso e o volume de operações por tipo.

- **Jobs started per minute:** Na Figura 39 no canto superior esquerdo, é apresentado o número de *jobs* iniciados por minuto, permitindo avaliar a frequência de execução das *Dags*. A sua oscilação revela a natureza programada e intermitente dos processos, refletindo o comportamento esperado.

- **Total Failed Tasks:** Na Figura 39, no lado esquerdo é indicado que apenas uma tarefa falhou no período analisado, evidenciando a fiabilidade da solução implementada.

- **DAG Processing Import Errors:** Na Figura 39, no canto inferior esquerdo o painel regista zero erros na importação de *Dags*, significando que todos os ficheiros de definição foram corretamente interpretados e carregados.

- **Scheduler Heartbeat:** Na Figura 39, no canto superior direito mostra a métrica `scheduler_heartbeat`, que indica a frequência de comunicação do *scheduler*. Um valor crescente e elevado (neste caso, 241441) demonstra que o *scheduler* está ativo e a funcionar corretamente, sem interrupções.

- **Total Successful Tasks:** Na Figura 39, no lado direito é apresentado um total de 929 tarefas concluídas com sucesso, reforçando a estabilidade e a eficiência operacional da infraestrutura.

- **DAG Bag Size:** Na Figura 39, no canto inferior direito mostra-se o número de *Dags* ativas carregadas na memória do *scheduler* neste caso, 10 , um valor indicativo de uma carga perfeitamente gerível. O *DAG Bag Size*, não possui um limite fixo no Airflow, sendo condicionado pelos recursos do sistema, nomeadamente a memória RAM disponível, o número de CPUs e o espaço em disco. Para determinar se o sistema se encontra a operar dentro de limites seguros, é possível monitorizar a latência do *scheduler*, o consumo de CPU e memória, bem como o número de *tasks* em fila. Valores elevados de *DAG Bag Size* podem aumentar o tempo de carregamento dos DAGs e a probabilidade de sobrecarga.

Esta abordagem de monitorização granular baseada em métricas no Airflow permite não só a deteção precoce de anomalias e falhas, mas também o acompanhamento do desempenho ao longo do tempo. Em contraste com a abordagem limitada da plataforma SMARTDEST, que apenas indicava o estado geral das fontes de dados, a solução atual oferece uma visibilidade aprofundada sobre o funcionamento interno do orquestrador, essencial para uma resposta rápida a problemas, para a escalabilidade informada do sistema e para garantir a continuidade do serviço com elevados níveis de confiança.

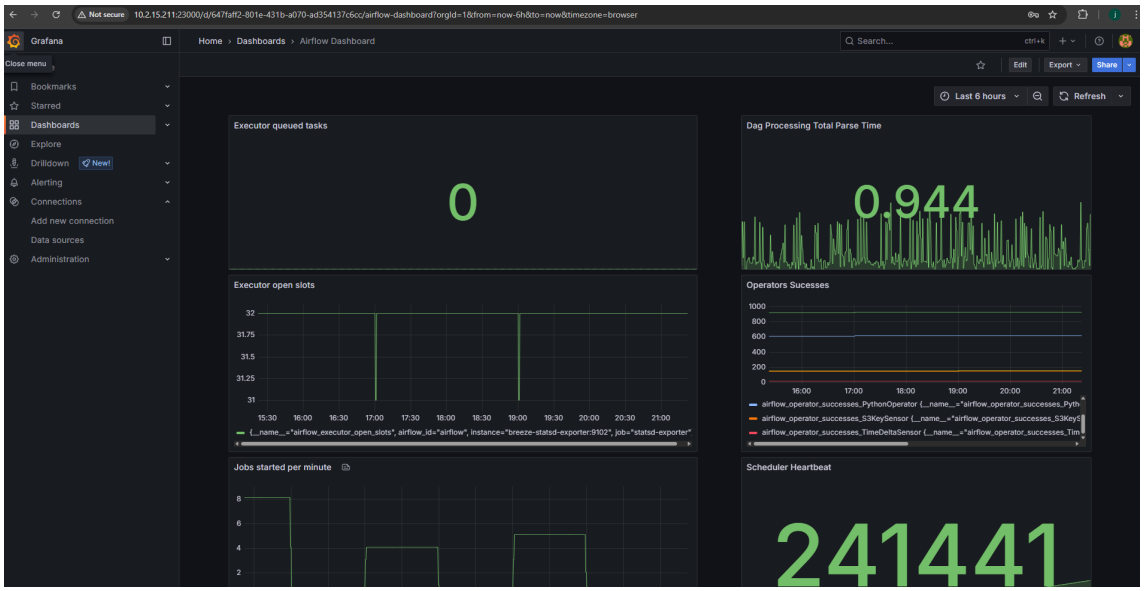

Figura 38: Monitorização do Airflow - Parte 1

A monitorização da infraestrutura do MinIO é realizada através do seu painel de monitorização nativo, que permite acompanhar em tempo real o estado do armazenamento de objetos (Figura 40). O painel exibe indicadores essenciais para a gestão e fiabilidade do sistema, como o número de *buckets* existentes (2), a quantidade total de objetos armazenados (1.5K) e o espaço ocupado (747 MiB). O sistema também fornece informação sobre a disponibilidade dos servidores e discos (dois servidores e quatro discos operacionais), garantindo redundância e resiliência. Embora esta capacidade de monitorização possa ser estendida com o Grafana e o Prometheus, a solução nativa já oferece visibilidade suficiente para ambientes controlados, facilitando a deteção de anomalias e garantindo a continuidade do serviço do MinIO.

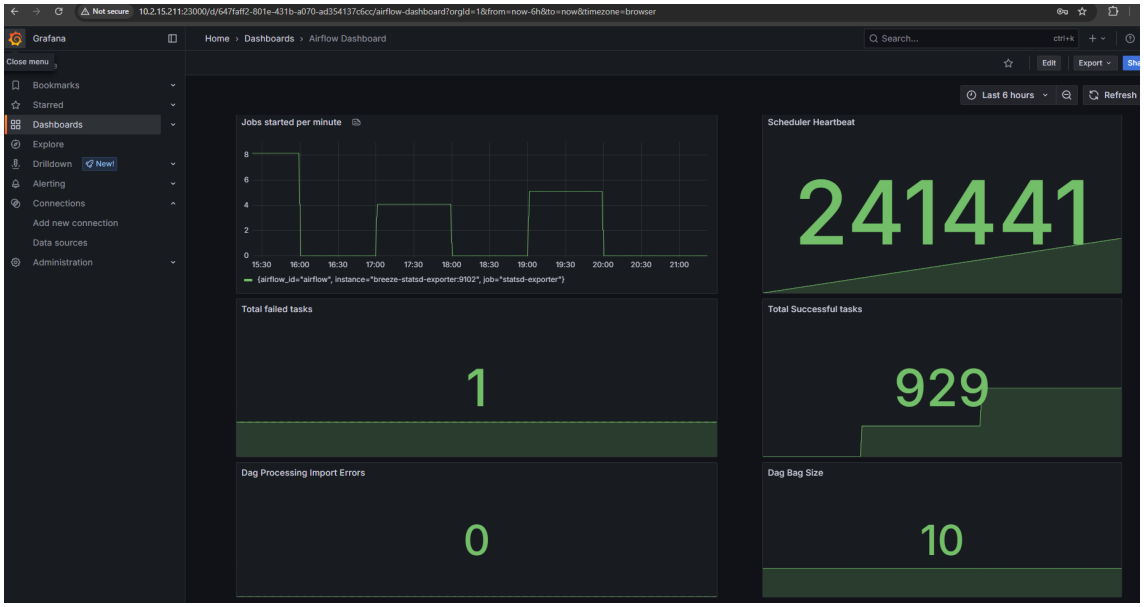

Figura 39: Monitorização do Airflow - Parte 2

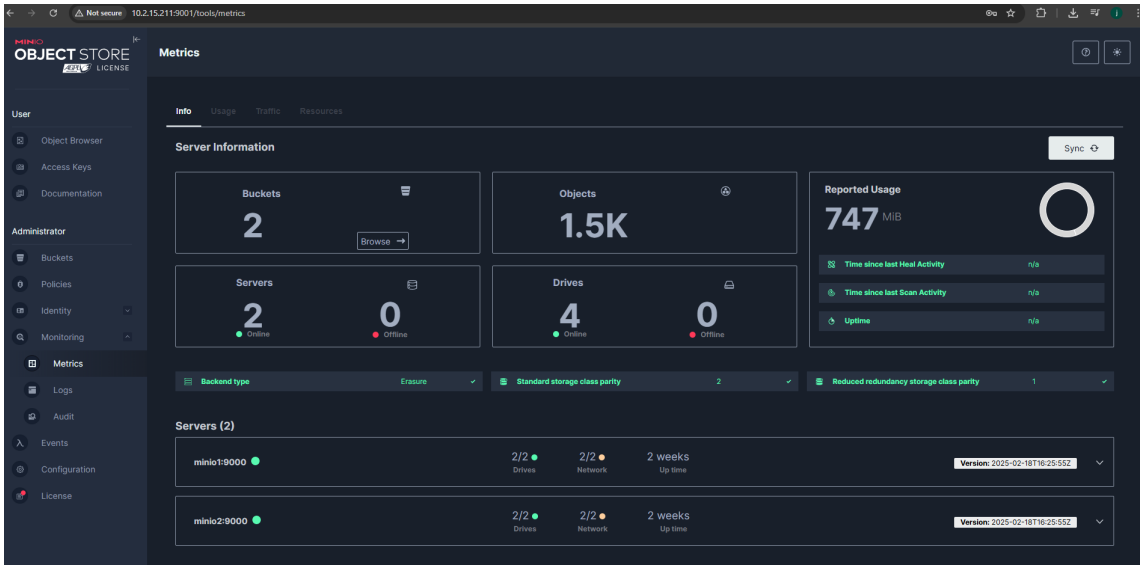

Figura 40: Monitorização do Minio

A monitorização da infraestrutura da base de dados *serving data*, um elemento central no fornecimento de dados, é realizada através da sua integração contínua com o sistema Prometheus. Esta integração permite a recolha automática e sistemática de métricas de desempenho, como o tempo de resposta, o número de conexões ativas, o consumo de CPU e a utilização de memória.

A adoção desta abordagem permite a deteção proativa de degradações de desempenho e garante a escalabilidade e fiabilidade da infraestrutura. Em contraste com a ausência de visibilidade que

caracterizava a arquitetura anterior (SMARTDEST), a introdução desta camada de observabilidade representa uma melhoria significativa na capacidade de diagnóstico e gestão operacional da base de dados.

A Figura 41 ilustra um primeiro conjunto de *dashboards* que monitorizam indicadores gerais da infraestrutura e a utilização de recursos do sistema. Este painel permite observar, em tempo real, métricas essenciais como o consumo médio de CPU e memória, o número de descritores de ficheiros abertos e o volume de dados processados. São também visíveis parâmetros críticos de configuração do PostgreSQL, como shared_buffers, o que é fundamental para o ajuste fino (tuning) e para a identificação de potenciais gargalos.

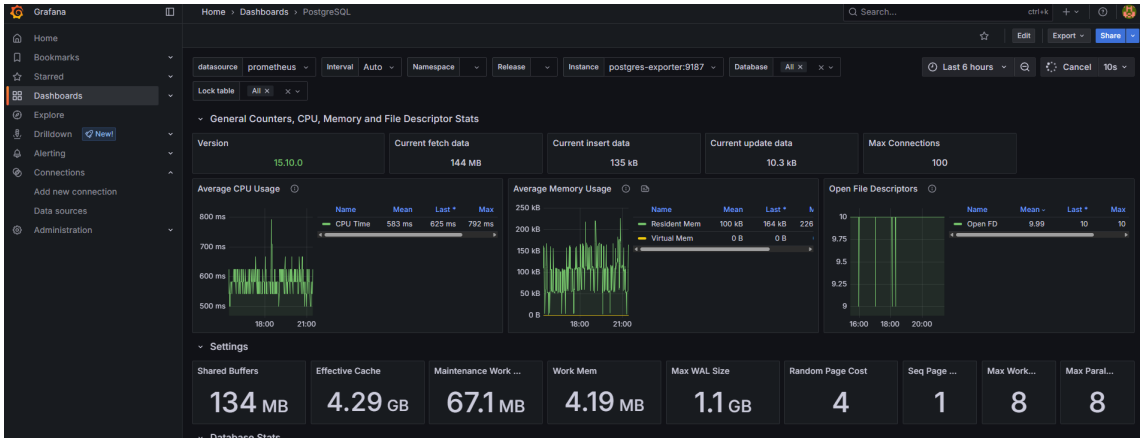

Figura 41: Monitorização do PostgreSQL - Parte 1

A Figura 42 apresenta um segundo conjunto de gráficos com foco no comportamento transacional e na gestão de concorrência. Neste painel são monitorizadas as sessões ativas, a taxa de transações e a ocorrência de bloqueios em tabelas. Adicionalmente, a monitorização inclui a identificação de conflitos entre diferentes bases de dados, informação crucial para diagnosticar situações de contenção de recursos.

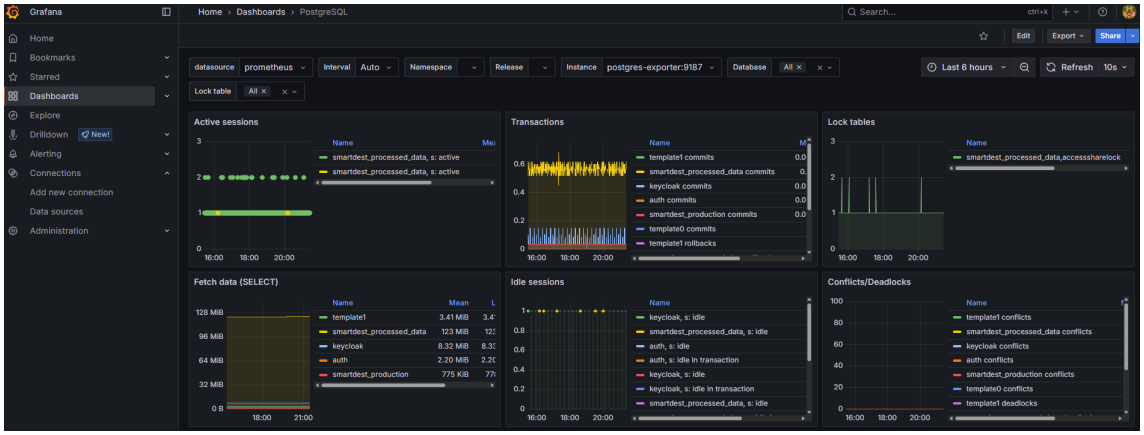

Figura 42: Monitorização do PostgreSQL - Parte 2

Na Figura 43, um terceiro painel focado nas métricas de interação com os dados permite observar o volume de dados retornados, inseridos, atualizados e eliminados. Destaca-se o indicador de Cache Hit Rate, que se mantém consistentemente acima dos 99,9 %, revelando uma utilização extremamente eficiente da memória para *caching* e, consequentemente, melhorando significativamente a latência nas consultas.

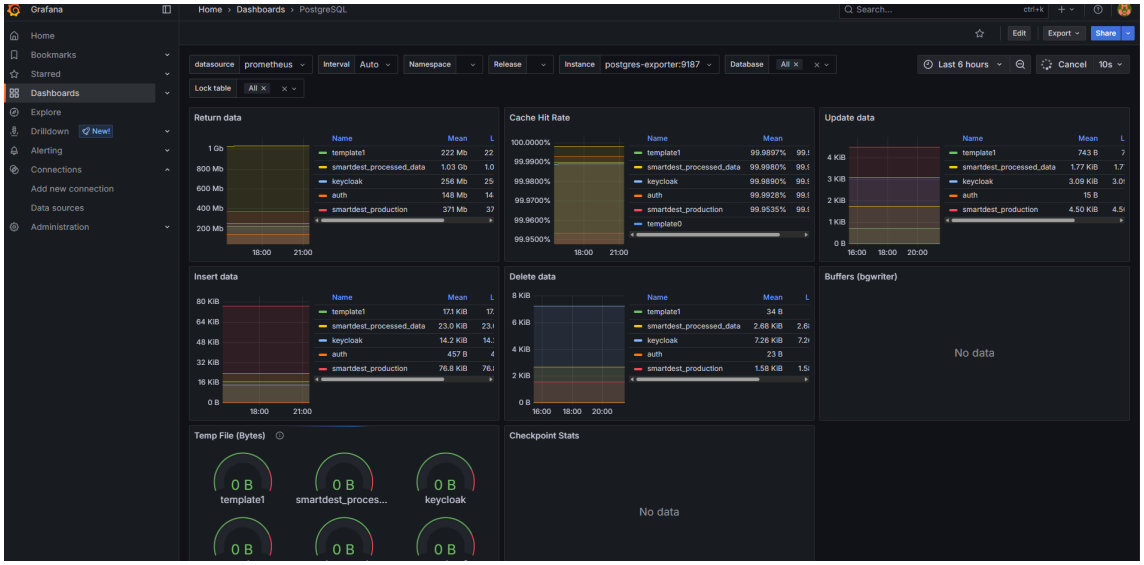

Figura 43: Monitorização do PostgreSQL - Parte 3

O cenário de validação demonstrou a superioridade inequívoca da nova arquitetura de dados em termos de observabilidade, contrastando-a com as limitações da antiga plataforma SMARTDEST. Enquanto a abordagem anterior se restringia a um dashboard rudimentar, que oferecia apenas

um conhecimento superficial sobre o estado das fontes de dados, a solução atual implementa um modelo de monitorização sólido e multifacetado.

A integração de ferramentas como Apache Airflow, Prometheus e Grafana revelou-se um pilar fundamental para esta evolução. Através da monitorização nativa do Airflow, é possível ir além de um simples registo de sucesso ou falha, obtendo uma visibilidade granular sobre a performance das *pipelines* de dados. A capacidade de analisar tempos de execução, inspecionar logs de forma precisa e visualizar o histórico operacional em diferentes formatos, como o calendário ou os gráficos de duração, permite uma gestão proativa e informada.

Além disso, a observabilidade da nova arquitetura estende-se a toda a infraestrutura subjacente. A monitorização da base de dados PostgeSQL, do MinIO e do próprio orquestrador Airflow, garante que o estado de cada componente do sistema é continuamente supervisionado fornecem uma visão holística e profunda da saúde do ecossistema de dados.

Em suma, a transição para esta nova arquitetura representa uma mudança paradigmática. A observabilidade moderna não é apenas um complemento, mas uma componente intrínseca que assegura a fiabilidade, a escalabilidade e a eficiência operacional

5.4 Análise Comparativa entre Requisitos e Sistema Implementado

Após a demonstração prática da arquitetura através dos cenários de validação apresentados nas secções anteriores, torna-se imperativo realizar uma validação formal face aos requisitos definidos na fase de metodologia. Esta análise visa confirmar se a solução implementada respondeu eficazmente às necessidades funcionais e não funcionais identificadas.

A apresentação da validação destes requisitos encontra-se dividida nas quatro áreas principais identificadas no levantamento inicial: Recolha, Armazenamento, Processamento e Monitorização.

Validação dos Requisitos de Recolha de Dados

Tabela 9: Requisitos de Recolha de Dados - Validação

ID	Estado	Observações
RID1	Cumprido	Implementado via *HttpSensor* do Apache Airflow
RID2	Cumprido	Implementado através do Airflow com a ajuda do Selenium para o *Web Scrapping*.
RID3	Cumprido	Configuração nativa de *retries* e *retry delay* no Airflow.
RID4	Cumprido	Utilização do *Scheduler* do Airflow com expressões CRON.
RID5	Cumprido	Integração com MinIO para persistência imediata dos dados originais (JSON, HTML).
RID6	Cumprido	Registo na base de dados de metadados do Airflow e na tabela data_source no PostgreSQL.
RID7	Cumprido	Logs detalhados de execução acessíveis via Airflow Web UI e persistidos em volume Docker.
RID8	Cumprido	Arquitetura modular baseada em DAGs Python, permitindo reutilização de código.

Validação dos Requisitos de Armazenamento de Dados

Tabela 10: Requisitos de Armazenamento de Dados - Validação

ID	Estado	Observações
AD1	Cumprido	Cluster MinIO configurado com múltiplos nós garantindo redundância.
AD2	Cumprido	Camada de *Serving Data* no PostgreSQL e *Data Marts*.
AD3	Cumprido	Validação de esquemas com Pandera antes da inserção no PostgreSQL.
AD4	Cumprido	Uso do MinIO para dados não estruturados e suporte a JSONB no PostgreSQL.
AD5	Cumprido	Índices configurados nas tabelas principais do PostgreSQL para suporte aos *dashboards*.
AD6	Cumprido	MinIO (HTML, JSON, PDF) e PostgreSQL (Tabelas relacionais).
AD7	Cumprido	Capacidade de adicionar novos nós ao MinIO.
AD8	Cumprido	Protegidos por autenticação e isolamento de redes via Docker Compose.
AD9	Não validado	A flexibilidade do modelo de dados não foi avaliada através de testes formais.
AD10	Não Cumprido	Não foi implementado backups automáticos.

Validação dos Requisitos de Processamento de Dados

Tabela 11: Requisitos de Processamento de Dados - Validação

ID	Estado	Observações
PD1	Cumprido	Criação de DAGs no Airflow para todas as fontes de dados.
PD2	Cumprido	Uso do PythonOperator do Airflow.
PD3	Cumprido	Funcionalidade nativa do Airflow.
PD4	Cumprido	Definição de dependências nos DAGs.
PD5	Cumprido	Utilização do LocalExecutor permitindo múltiplas tarefas simultâneas.
PD6	Cumprido	Arquitetura preparada para execução distribuída (testado com CeleryExecutor).
PD7	Cumprido	Captura de exceções e registo centralizado de logs no Airflow.
PD8	Cumprido	Funcionalidade nativa do Airflow.
PD9	Parcialmente	Arquitetura suporta via CeleryExecutor.
PD10	Não validado	Não foi avaliado através de testes formais.
PD11	Não Cumprido	Não foi configurado alertas.
PD12	Cumprido	Código organizado e documentado.
PD13	Cumprido	Métricas de duração de tarefas e taxas de sucesso visíveis no Grafana.
PD14	Cumprido	Tempos de processamento validados e adequados ao volume atual. A execução paralela de tarefas foi demonstrada eficazmente em **workflows** com múltiplas dependências.

Validação dos Requisitos de Monitorização e Qualidade

Tabela 12: Requisitos de Monitorização e Qualidade - Validação

ID	Estado	Justificação da Implementação
MTQ1	Cumprido	*Dashboards* do Airflow e Grafana indicam sucesso/falha por DAG.
MTQ2	Cumprido	*Dashboards* do Airflow e Grafana
MTQ3	Não Cumprido	Não foi configurado alertas no Prometheus.
MTQ4	Cumprido	Funcionalidade Nativa do Airflow.
MTQ5	Cumprido	Funcionalidade Nativa do Airflow.
MTQ6	Não Cumprido	Não foi configurado alertas.
MTQ7	Cumprido	Funcionalidade Nativa do Airflow.
MTQ8	Cumprido	*Dashboards* implementados no Grafana.
MTQ9	Não Validado	Não foi avaliado através de testes formais.
MTQ10	Cumprido	Definido no docker a política *restart always*.
MTQ11	Cumprido	*Retries* automáticos garantem entrega *at least once*.
MTQ12	Cumprido	Exporters (StatsD e Postgres) alimentam o Prometheus em tempo real.
MTQ13	Cumprido	Configurado no Prometheus para guardar os dados durante 6 meses.
MTQ14	Cumprido	Visibilidade *full-stack* assegurada. Os *dashboards* do Grafana monitorizam os recursos do sistema e a interface nativa do Airflow detalha o estado operacional dos processos de recolha e transformação.

A análise dos requisitos definidos e do sistema implementado permite concluir que a reestruturação da arquitetura da plataforma SMARTDEST foi bem-sucedida na maioria dos seus objetivos fundamentais.

No domínio da Recolha e Armazenamento, a validação dos requisitos (RID1-RID8 e AD1-AD8) confirma que a introdução do Apache Airflow e do MinIO permitiu criar um ecossistema independente da origem dos dados, capaz de ingerir e persistir tanto dados estruturados como não estruturados sem perda de informação. A estratégia de *Raw Data Storage* revelou-se um mecanismo de segurança vital, garantindo a preservação histórica dos dados mesmo perante falhas de processamento

Relativamente ao Processamento, o sistema cumpriu com distinção os requisitos de orquestração e transformação (PD1-PD8). A capacidade de gerir dependências complexas e executar tarefas em paralelo representa um salto qualitativo face ao sistema anterior. Contudo, a validação expôs limitações nos requisitos não funcionais de escalabilidade (PD9) e alertas (PD11). A opção pelo LocalExecutor e a ausência de notificações proativas (como emails ou mensagens automáticas em caso de erro no processamento) são deficiências que devem ser priorizadas em iterações futuras.

Quanto à Monitorização, a arquitetura preencheu a lacuna crítica de observabilidade do projeto original. O cumprimento dos requisitos MTQ1 a MTQ14 (com exceção dos alertas proativos em caso de erro de algum componente da arquitetura) assegura que a equipa de gestão tem agora uma visão granular e em tempo real sobre a saúde da infraestrutura e o estado dos dados.

Em síntese, embora existam aspetos operacionais a refinar, a arquitetura implementada satisfaz os requisitos críticos. A solução entrega uma plataforma estável e preparada para evoluir, constituindo uma base sólida.

6 Conclusão

Neste capítulo, resumem-se os principais contributos da reestruturação da plataforma SMARTDEST, as limitações deste trabalho e trabalho futuro.

6.1 Síntese do Trabalho Realizado

A presente dissertação teve como principal objetivo a reestruturação da arquitetura da plataforma de dados turísticos SMARTDEST, visando superar as limitações de escalabilidade, robustez e observabilidade da sua implementação original. O trabalho partiu de uma análise crítica ao sistema existente, identificando as suas fragilidades, e culminou no desenvolvimento e validação de uma nova arquitetura, fundamentada em tecnologias open-source e nas melhores práticas de engenharia de dados.

A reestruturação da arquitetura de dados resultou numa plataforma modular e resiliente, que superou as limitações do sistema anterior. A transição para ferramentas como o Apache Airflow permitiu uma gestão de dados mais flexível e robusta, integrando com sucesso fontes heterogéneas, conforme demonstrado no Cenário de Validação 1. O Airflow foi crucial para orquestrar de forma fiável a recolha de dados via APIs (ex: IPMA), por *web scraping* (ex: eDreams) e através de fontes documentais abertas (ex: DREM), provando a sua eficácia como orquestrador, facilitando a monitorização e a manutenção contínua exigidas pela diversidade e pela natureza volátil de cada fonte.

A introdução de uma camada de armazenamento de dados brutos (*raw data*) usando o MinIO foi crucial para garantir a integridade dos dados, prevenindo a sua perda em caso de falhas de processamento. Esta abordagem foi validada no Cenário de Validação 2, que provou que, mesmo com um erro na camada de validação do pipeline de web scraping da eDreams, os dados brutos foram preservados, permitindo uma futura recuperação e processamento. Complementarmente, a implementação de mecanismos de *retry* automática no Airflow aumentou a fiabilidade dos *pipelines*, mitigando falhas temporárias, como os *connection timeouts* observados na fonte de dados docks_booking.

Além disso, a monitorização e a observabilidade foram radicalmente melhoradas. A integração do Airflow com o Prometheus e o Grafana proporcionou uma visão completa e detalhada do

desempenho dos *pipelines* e da saúde da infraestrutura, algo que era inexistente na arquitetura original conforme foi analisado no Cenário de Validação 3.

6.2 Limitações do Estudo

Apesar dos resultados positivos alcançados com a reestruturação da plataforma SMARTDEST, é crucial reconhecer as limitações inerentes a este projeto.

O sistema proposto não foi otimizado para o processamento de volumes de dados verdadeiramente massivos. A exploração de modelos de processamento distribuído mais avançados, como a utilização de *clusters* Apache Spark, não foi aprofundada neste trabalho. Assim, a arquitetura, apesar de pronta para futuras integrações, ainda não está preparada para um processamento em larga escala.

A solução foi desenvolvida para um paradigma de processamento em *batch*, uma escolha alinhada com os requisitos identificados. Contudo, esta abordagem restringe a análise de dados em tempo real ou quase em tempo real, como os provenientes de sensores de IoT ou redes sociais, áreas para as quais o Apache Airflow não é ideal. Para a orquestração em tempo real, seria mais adequado usar uma ferramenta como o Apache NiFi, especificamente otimizado para este tipo de cargas de trabalho, conforme explorado no Capítulo 2.

Outra limitação reside no facto de a implementação atual, embora funcional e com mecanismos de validação e rastreabilidade, não adotar ferramentas de governança de dados mais sofisticadas, como catálogos de dados. Esta abordagem ainda não incorpora o espectro completo das melhores práticas de governança e qualidade de dados para um ambiente de *big data* em larga escala.

6.3 Trabalho Futuro

Com base nos resultados alcançados e nas limitações identificadas, diversas avenidas de desenvolvimento podem ser exploradas para expandir e otimizar a plataforma SMARTDEST, consolidando o seu papel como um pilar para um turismo mais inteligente.

Uma das prioridades é a otimização do processamento de dados em larga escala. A atual implementação, que utiliza o LocalExecutor do Apache Airflow, pode ser migrada para modelos mais robustos e escaláveis, como o Celery Executor ou o Kubernetes Executor. Esta transição, em conjunto com a utilização do Apache Spark, permitirá uma execução distribuída de tarefas, au-

mentando significativamente a capacidade de processamento paralelo e a resiliência da plataforma para lidar com volumes de dados turísticos cada vez maiores.

Além disso, é fundamental o desenvolvimento de capacidades analíticas avançadas. Ao utilizar o repositório de dados estruturados e brutos (MinIO e PostgreSQL), é possível desenvolver e implementar modelos de *machine learning*. Estes modelos podem focar-se na previsão da procura turística, na segmentação de perfis de visitantes com base em padrões de comportamento e na identificação de tendências emergentes.

Outro ponto de trabalho futuro seria seguir uma abordagem híbrida, combinando o Apache NiFi e o Apache Airflow. Com esta arquitetura, o NiFi gere a ingestão de dados em tempo real com baixa latência, enquanto o Airflow mantém o seu papel na orquestração de processos mais pesados, como os *batch jobs* e as tarefas ETL periódicas. Isto torna a ingestão de dados em tempo real uma realidade, algo que atualmente não conseguimos fazer

Por fim, incorporar uma maior diversidade de fontes de dados, de forma a proporcionar análises mais abrangentes que permitam à plataforma compreender rapidamente os cenários e padrões de comportamento turístico em constante mudança.

Em suma, esta dissertação comprovou a viabilidade da arquitetura reestruturada, constituindo um contributo relevante para a gestão de dados turísticos. Ao ultrapassar as limitações do sistema anterior, a nova plataforma não só otimiza os processos de recolha, tratamento e armazenamento de dados heterogéneos, como também estabelece uma base tecnológica robusta e flexível que sustenta a inovação contínua. Desta forma, a plataforma encontra-se apta a responder às exigências futuras do setor, afirmando-se como um recurso essencial para o desenvolvimento do turismo inteligente.

7 Anexos

7.1 Anexo A: Ficheiro docker-compose.yaml para Orquestração de Airflow, Grafana e Prometheus

```yaml
x-airflow-common:
  &airflow-common
  image: ${AIRFLOW_IMAGE_NAME:-extending_airflow:latest_v8}
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: LocalExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN:
    ↪ postgresql+psycopg2://airflow:airflow@postgres/airflow
    AIRFLOW__CORE__FERNET_KEY: 'rgaQPnWgGWtCg_5Kvy8jpU7CsydUyhXbFVgDPuBW4ac='
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
    AIRFLOW__API__AUTH_BACKENDS:
    ↪ 'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    AIRFLOW__WEBSERVER__RBAC: "true"  # Enable Role-Based Access Control
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
    _PIP_ADDITIONAL_REQUIREMENTS: ${_PIP_ADDITIONAL_REQUIREMENTS:-}
    AIRFLOW__METRICS__STATSD_ON: 'true'
    AIRFLOW__METRICS__STATSD_HOST: 'statsd-exporter'
    AIRFLOW__METRICS__STATSD_PORT: '9125'
    AIRFLOW__METRICS__STATSD_PREFIX: airflow
    AIRFLOW__METRICS__METRICS_USE_PATTERN_MATCH: 'false'


  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
    - ${AIRFLOW_PROJ_DIR:-.}/downloads:/opt/airflow/downloads
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    postgres:
      condition: service_healthy

services:
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
```

```
43      healthcheck:
44        test: ["CMD", "pg_isready", "-U", "airflow"]
45        interval: 10s
46        retries: 5
47        start_period: 5s
48      restart: always
49
50    airflow-webserver:
51      <<: *airflow-common
52      command: webserver
53      ports:
54        - "8080:8080"
55      healthcheck:
56        test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
57        interval: 30s
58        timeout: 10s
59        retries: 5
60        start_period: 30s
61      restart: always
62      depends_on:
63        <<: *airflow-common-depends-on
64        airflow-init:
65          condition: service_completed_successfully
66
67    airflow-scheduler:
68      <<: *airflow-common
69      command: scheduler
70      healthcheck:
71        test: ["CMD", "curl", "--fail", "http://localhost:8974/health"]
72        interval: 30s
73        timeout: 10s
74        retries: 5
75        start_period: 30s
76      restart: always
77      depends_on:
78        <<: *airflow-common-depends-on
79        airflow-init:
80          condition: service_completed_successfully
81
82    airflow-triggerer:
83      <<: *airflow-common
84      command: triggerer
85      healthcheck:
86        test: ["CMD-SHELL", 'airflow jobs check --job-type TriggererJob --hostname
          ↪  "$${HOSTNAME}"']
87        interval: 30s
88        timeout: 10s
89        retries: 5
90        start_period: 30s
91      restart: always
```

```
92      depends_on:
93        <<: *airflow-common-depends-on
94        airflow-init:
95          condition: service_completed_successfully
96
97    airflow-init:
98      <<: *airflow-common
99      entrypoint: /bin/bash
100     command:
101       - -c
102       - |
103         if [[ -z "${AIRFLOW_UID}" ]]; then
104           echo
105           echo -e "\033[1;33mWARNING!!!: AIRFLOW_UID not set!\e[0m"
106           echo "If you are on Linux, you SHOULD follow the instructions below to set "
107           echo "AIRFLOW_UID environment variable, otherwise files will be owned by root."
108           echo "For other operating systems you can get rid of the warning with manually
              ↪  created .env file:"
109           echo
110         fi
111         one_meg=1048576
112         mem_available=$$(($$(getconf _PHYS_PAGES) * $$(getconf PAGE_SIZE) / one_meg))
113         cpus_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)
114         disk_available=$$(df / | tail -1 | awk '{print $$4}')
115         warning_resources="false"
116         if (( mem_available < 4000 )) ; then
117           echo
118           echo -e "\033[1;33mWARNING!!!: Not enough memory available for Docker.\e[0m"
119           echo "At least 4GB of memory required. You have $$(numfmt --to iec
              ↪  $$((mem_available * one_meg)))"
120           echo
121           warning_resources="true"
122         fi
123         if (( cpus_available < 2 )); then
124           echo
125           echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for Docker.\e[0m"
126           echo "At least 2 CPUs recommended. You have $${cpus_available}"
127           echo
128           warning_resources="true"
129         fi
130         if (( disk_available < one_meg * 10 )); then
131           echo
132           echo -e "\033[1;33mWARNING!!!: Not enough Disk space available for Docker.\e[0m"
133           echo "At least 10 GBs recommended. You have $$(numfmt --to iec
              ↪  $$((disk_available * 1024 )))"
134           echo
135           warning_resources="true"
136         fi
137         if [[ $${warning_resources} == "true" ]]; then
138           echo
```

```
          echo -e "\033[1;33mWARNING!!!: You have not enough resources to run Airflow (see
          ↪  above)!\e[0m"
          echo "Please follow the instructions to increase amount of resources available:"
          echo
        fi
        mkdir -p /sources/logs /sources/dags /sources/plugins
        chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
        exec /entrypoint airflow version
    environment:
      <<: *airflow-common-env
      _AIRFLOW_DB_MIGRATE: 'true'
      _AIRFLOW_WWW_USER_CREATE: 'true'
      _AIRFLOW_WWW_USER_USERNAME: ${_AIRFLOW_WWW_USER_USERNAME:-airflow}
      _AIRFLOW_WWW_USER_PASSWORD: ${_AIRFLOW_WWW_USER_PASSWORD:-airflow}
      _PIP_ADDITIONAL_REQUIREMENTS: ''
    user: "0:0"
    volumes:
      - ${AIRFLOW_PROJ_DIR:-.}:/sources

  airflow-cli:
    <<: *airflow-common
    profiles:
      - debug
    environment:
      <<: *airflow-common-env
      CONNECTION_CHECK_MAX_COUNT: "0"
    command:
      - bash
      - -c
      - airflow

  selenium:
    image: selenium/standalone-chrome:latest
    shm_size: 2gb
    ports:
      - "4444:4444"
    environment:
      - SE_NODE_MAX_SESSIONS=5
      - SE_NODE_OVERRIDE_MAX_SESSIONS=true
      - SE_NODE_SESSION_TIMEOUT=60
    restart: always

  prometheus:
    image: prom/prometheus
    container_name: "breeze-prometheus"
    user: "0"
    ports:
      - "29090:9090"
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
```

```
188       - '--storage.tsdb.path=/prometheus'
189       - '--storage.tsdb.retention.time=30d'
190     volumes:
191       - ./configs_prometheus/prometheus-config.yml:/etc/prometheus/prometheus.yml
192       - prometheus_data:/prometheus
193     restart: always
194
195   grafana:
196     image: grafana/grafana:12.0.0
197     container_name: "breeze-grafana"
198     restart: always
199     environment:
200       - GF_PLUGINS_PREINSTALL=grafana-clock-panel
201     ports:
202       - "23000:3000"
203     volumes:
204       - 'grafana_storage:/var/lib/grafana'
205     depends_on:
206       - prometheus
207
208   statsd-exporter:
209     image: quay.io/prometheus/statsd-exporter:v0.28.0
210     labels:
211       breeze.description: "Integration required for Statsd hooks."
212     container_name: "breeze-statsd-exporter"
213     restart: always
214     entrypoint: ["/bin/sh", "-c", "--"]
215     command: ["statsd_exporter",
        ↪  "--statsd.mapping-config=/etc/statsd_exporter_mapping.yaml","--interactive","--loglevel=debug"]
216     volumes:
217       -
          ↪  ./configs_prometheus/statsd_exporter_mapping.yaml:/etc/statsd_exporter_mapping.yaml
218     ports:
219       - "9125:9125"
220       - "9125:9125/udp"
221       - "29102:9102"
222
223   postgres-exporter:
224     image: prometheuscommunity/postgres-exporter:latest
225     container_name: "breeze-postgres-exporter"
226     environment:
227       DATA_SOURCE_NAME:
          ↪  "postgresql://postgres:catwater_123@10.2.15.210:5432/smartdest_processed_data?sslmode=disable"
228     ports:
229       - "9187:9187"
230     restart: always
231     labels:
232       breeze.description: "PostgreSQL metrics exporter for Prometheus monitoring."
233
234
```

```
volumes:
  postgres-db-volume:
  prometheus_data:
  grafana_storage:
```

7.2 Anexo B : Ficheiro docker-compose.yaml para Orquestração do MinIO

```
x-minio-common: &minio-common
  image: minio/minio:RELEASE.2025-02-18T16-25-55Z-cpuv1
  command: server --console-address ":9001" http://minio{1...2}/data{1...2}
  expose:
    - "9000"
    - "9001"
  healthcheck:
    test: ["CMD", "mc", "ready", "local"]
    interval: 5s
    timeout: 5s
    retries: 5

services:
  minio1:
    <<: *minio-common
    hostname: minio1
    restart: always
    volumes:
      - data1-1:/data1
      - data1-2:/data2

  minio2:
    <<: *minio-common
    hostname: minio2
    restart: always
    volumes:
      - data2-1:/data1
      - data2-2:/data2

  nginx:
    image: nginx:1.19.2-alpine
    hostname: nginx
    restart: always
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    ports:
      - "9000:9000"
      - "9001:9001"
    depends_on:
      - minio1
```

```
      - minio2
volumes:
  data1-1:
  data1-2:
  data2-1:
  data2-2:
```

7.3 Anexo C : Ficheiro de Configuração nginx.conf para Orquestração de MinIO

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  4096;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
    sendfile        on;
    keepalive_timeout  65;

    # include /etc/nginx/conf.d/*.conf;

    upstream minio {
        server minio1:9000;
        server minio2:9000;
    }

    upstream console {
        ip_hash;
        server minio1:9001;
        server minio2:9001;
    }

    server {
        listen       9000;
        listen  [::]:9000;
```

```
39          server_name  localhost;
40
41
42          ignore_invalid_headers off;
43
44          client_max_body_size 0;
45
46          proxy_buffering off;
47          proxy_request_buffering off;
48
49          location / {
50              proxy_set_header Host $http_host;
51              proxy_set_header X-Real-IP $remote_addr;
52              proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
53              proxy_set_header X-Forwarded-Proto $scheme;
54
55              proxy_connect_timeout 300;
56
57              proxy_http_version 1.1;
58              proxy_set_header Connection "";
59              chunked_transfer_encoding off;
60
61              proxy_pass http://minio;
62          }
63      }
64
65      server {
66          listen       9001;
67          listen  [::]:9001;
68          server_name  localhost;
69
70          ignore_invalid_headers off;
71
72          client_max_body_size 0;
73
74          proxy_buffering off;
75          proxy_request_buffering off;
76
77          location / {
78              proxy_set_header Host $http_host;
79              proxy_set_header X-Real-IP $remote_addr;
80              proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
81              proxy_set_header X-Forwarded-Proto $scheme;
82              proxy_set_header X-NginX-Proxy true;
83
84
85              real_ip_header X-Real-IP;
86
87              proxy_connect_timeout 300;
88
```

```
89
90              proxy_http_version 1.1;
91              proxy_set_header Upgrade $http_upgrade;
92              proxy_set_header Connection "upgrade";
93
94              chunked_transfer_encoding off;
95
96              proxy_pass http://console;
97          }
98      }
99  }
```

7.4 Anexo D : Ficheiro de Configuração Pormetheus prometheus-config.yml

```
1   global:
2     scrape_interval: 60s
3     evaluation_interval: 60s
4     scrape_timeout: 15s
5   scrape_configs:
6     - job_name: prometheus
7       static_configs:
8         - targets:
9             - 'localhost:9090'
10    - job_name: postgres-exporter
11      static_configs:
12        - targets: ['postgres-exporter:9187']
13    - job_name: statsd-exporter
14      scheme: http
15      metrics_path: metrics
16      static_configs:
17        - targets:
18            - 'breeze-statsd-exporter:9102'
19          labels:
20            airflow_id: airflow
21      tls_config:
22        insecure_skip_verify: true
```

7.5 Anexo E : Ficheiro de Configuração statsd_exporter_mapping.yaml

```
1   mappings:
2     - match: "(.+)\\.(.+)_start$"
3       match_metric_type: counter
4       name: "af_agg_job_start"
5       match_type: regex
6       labels:
7         airflow_id: "$1"
```

```
8         job_name: "$2"
9     - match: "(.+)\\.(.+)_end$"
10      match_metric_type: counter
11      name: "af_agg_job_end"
12      match_type: regex
13      labels:
14        airflow_id: "$1"
15        job_name: "$2"
16    - match: "(.+)\\.operator_failures_(.+)$"
17      match_metric_type: counter
18      name: "af_agg_operator_failures"
19      match_type: regex
20      labels:
21        airflow_id: "$1"
22        operator_name: "$2"
23    - match: "(.+)\\.operator_successes_(.+)$"
24      match_metric_type: counter
25      name: "af_agg_operator_successes"
26      match_type: regex
27      labels:
28        airflow_id: "$1"
29        operator_name: "$2"
30    - match: "*.ti_failures"
31      match_metric_type: counter
32      name: "af_agg_ti_failures"
33      labels:
34        airflow_id: "$1"
35    - match: "*.ti_successes"
36      match_metric_type: counter
37      name: "af_agg_ti_successes"
38      labels:
39        airflow_id: "$1"
40    - match: "*.zombies_killed"
41      match_metric_type: counter
42      name: "af_agg_zombies_killed"
43      labels:
44        airflow_id: "$1"
45    - match: "*.scheduler_heartbeat"
46      match_metric_type: counter
47      name: "af_agg_scheduler_heartbeat"
48      labels:
49        airflow_id: "$1"
50    - match: "*.dag_processing.processes"
51      match_metric_type: counter
52      name: "af_agg_dag_processing_processes"
53      labels:
54        airflow_id: "$1"
55    - match: "*.scheduler.tasks.killed_externally"
56      match_metric_type: counter
57      name: "af_agg_scheduler_tasks_killed_externally"
```

```
58      labels:
59        airflow_id: "$1"
60    - match: "*.scheduler.tasks.running"
61      match_metric_type: counter
62      name: "af_agg_scheduler_tasks_running"
63      labels:
64        airflow_id: "$1"
65    - match: "*.scheduler.tasks.starving"
66      match_metric_type: counter
67      name: "af_agg_scheduler_tasks_starving"
68      labels:
69        airflow_id: "$1"
70    - match: "*.scheduler.orphaned_tasks.cleared"
71      match_metric_type: counter
72      name: "af_agg_scheduler_orphaned_tasks_cleared"
73      labels:
74        airflow_id: "$1"
75    - match: "*.scheduler.orphaned_tasks.adopted"
76      match_metric_type: counter
77      name: "af_agg_scheduler_orphaned_tasks_adopted"
78      labels:
79        airflow_id: "$1"
80    - match: "*.scheduler.critical_section_busy"
81      match_metric_type: counter
82      name: "af_agg_scheduler_critical_section_busy"
83      labels:
84        airflow_id: "$1"
85    - match: "*.sla_email_notification_failure"
86      match_metric_type: counter
87      name: "af_agg_sla_email_notification_failure"
88      labels:
89        airflow_id: "$1"
90    - match: "*.ti.start.*.*"
91      match_metric_type: counter
92      name: "af_agg_ti_start"
93      labels:
94        airflow_id: "$1"
95        dag_id: "$2"
96        task_id: "$3"
97    - match: "*.ti.finish.*.*.*"
98      match_metric_type: counter
99      name: "af_agg_ti_finish"
100     labels:
101       airflow_id: "$1"
102       dag_id: "$2"
103       task_id: "$3"
104       state: "$4"
105   - match: "*.dag.callback_exceptions"
106     match_metric_type: counter
107     name: "af_agg_dag_callback_exceptions"
```

```
    labels:
      airflow_id: "$1"
  - match: "*.celery.task_timeout_error"
    match_metric_type: counter
    name: "af_agg_celery_task_timeout_error"
    labels:
      airflow_id: "$1"

  # === Gauges ===
  - match: "*.dagbag_size"
    match_metric_type: gauge
    name: "af_agg_dagbag_size"
    labels:
      airflow_id: "$1"
  - match: "*.dag_processing.import_errors"
    match_metric_type: gauge
    name: "af_agg_dag_processing_import_errors"
    labels:
      airflow_id: "$1"
  - match: "*.dag_processing.total_parse_time"
    match_metric_type: gauge
    name: "af_agg_dag_processing_total_parse_time"
    labels:
      airflow_id: "$1"
  - match: "*.dag_processing.last_runtime.*"
    match_metric_type: gauge
    name: "af_agg_dag_processing_last_runtime"
    labels:
      airflow_id: "$1"
      dag_file: "$2"
  - match: "*.dag_processing.last_run.seconds_ago.*"
    match_metric_type: gauge
    name: "af_agg_dag_processing_last_run_seconds"
    labels:
      airflow_id: "$1"
      dag_file: "$2"
  - match: "*.dag_processing.processor_timeouts"
    match_metric_type: gauge
    name: "af_agg_dag_processing_processor_timeouts"
    labels:
      airflow_id: "$1"
  - match: "*.executor.open_slots"
    match_metric_type: gauge
    name: "af_agg_executor_open_slots"
    labels:
      airflow_id: "$1"
  - match: "*.executor.queued_tasks"
    match_metric_type: gauge
    name: "af_agg_executor_queued_tasks"
    labels:
```

```
158         airflow_id: "$1"
159     - match: "*.executor.running_tasks"
160       match_metric_type: gauge
161       name: "af_agg_executor_running_tasks"
162       labels:
163         airflow_id: "$1"
164     - match: "*.pool.open_slots.*"
165       match_metric_type: gauge
166       name: "af_agg_pool_open_slots"
167       labels:
168         airflow_id: "$1"
169         pool_name: "$2"
170     - match: "*.pool.queued_slots.*"
171       match_metric_type: gauge
172       name: "af_agg_pool_queued_slots"
173       labels:
174         airflow_id: "$1"
175         pool_name: "$2"
176     - match: "*.pool.running_slots.*"
177       match_metric_type: gauge
178       name: "af_agg_pool_running_slots"
179       labels:
180         airflow_id: "$1"
181         pool_name: "$2"
182     - match: "*.pool.starving_tasks.*"
183       match_metric_type: gauge
184       name: "af_agg_pool_starving_tasks"
185       labels:
186         airflow_id: "$1"
187         pool_name: "$2"
188     - match: "*.smart_sensor_operator.poked_tasks"
189       match_metric_type: gauge
190       name: "af_agg_smart_sensor_operator_poked_tasks"
191       labels:
192         airflow_id: "$1"
193     - match: "*.smart_sensor_operator.poked_success"
194       match_metric_type: gauge
195       name: "af_agg_smart_sensor_operator_poked_success"
196       labels:
197         airflow_id: "$1"
198     - match: "*.smart_sensor_operator.poked_exception"
199       match_metric_type: gauge
200       name: "af_agg_smart_sensor_operator_poked_exception"
201       labels:
202         airflow_id: "$1"
203     - match: "*.smart_sensor_operator.exception_failures"
204       match_metric_type: gauge
205       name: "af_agg_smart_sensor_operator_exception_failures"
206       labels:
207         airflow_id: "$1"
```

```yaml
  - match: "*.smart_sensor_operator.infra_failures"
    match_metric_type: gauge
    name: "af_agg_smart_sensor_operator_infra_failures"
    labels:
      airflow_id: "$1"

  # === Timers ===
  - match: "*.dagrun.dependency-check.*"
    match_metric_type: observer
    name: "af_agg_dagrun_dependency_check"
    labels:
      airflow_id: "$1"
      dag_id: "$2"
  - match: "*.dag.*.*.duration"
    match_metric_type: observer
    name: "af_agg_dag_task_duration"
    labels:
      airflow_id: "$1"
      dag_id: "$2"
      task_id: "$3"
  - match: "*.dag_processing.last_duration.*"
    match_metric_type: observer
    name: "af_agg_dag_processing_duration"
    labels:
      airflow_id: "$1"
      dag_file: "$2"
  - match: "*.dagrun.duration.success.*"
    match_metric_type: observer
    name: "af_agg_dagrun_duration_success"
    labels:
      airflow_id: "$1"
      dag_id: "$2"
  - match: "*.dagrun.duration.failed.*"
    match_metric_type: observer
    name: "af_agg_dagrun_duration_failed"
    labels:
      airflow_id: "$1"
      dag_id: "$2"
  - match: "*.dagrun.schedule_delay.*"
    match_metric_type: observer
    name: "af_agg_dagrun_schedule_delay"
    labels:
      airflow_id: "$1"
      dag_id: "$2"
  - match: "*.scheduler.critical_section_duration"
    match_metric_type: observer
    name: "af_agg_scheduler_critical_section_duration"
    labels:
      airflow_id: "$1"
  - match: "*.dagrun.*.first_task_scheduling_delay"
```

```
    match_metric_type: observer
    name: "af_agg_dagrun_first_task_scheduling_delay"
    labels:
      airflow_id: "$1"
      dag_id: "$2"
```

7.6 Anexo F : requirements.txt apache airflow

```
apache-airflow-providers-common-sql==1.21.0
apache-airflow-providers-postgres==6.0.0
psycopg2-binary==2.9.10
asyncpg==0.30.0
apache-airflow-providers-amazon==9.4.0
selenium==4.29.0
beautifulsoup4==4.13.3
demjson3==3.0.6
openpyxl==3.1.5
xlrd==2.0.1
pandera==0.23.1
apache-airflow[otel]==2.10.5
apache-airflow[statsd]==2.10.5
```

7.7 Anexo G : Celery Executor Airflow

```
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider packages you can use your
  ↪ extended image.
  # Comment the image line, place your Dockerfile in the directory where you placed the
  ↪ docker-compose.yaml
  # and uncomment the "build" line below, Then run `docker-compose build` to build the
  ↪ images.
  image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.10.5}
  # build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN:
    ↪ postgresql+psycopg2://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'true'
    AIRFLOW__API__AUTH_BACKENDS:
    ↪ 'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
```

```
    _PIP_ADDITIONAL_REQUIREMENTS: ${_PIP_ADDITIONAL_REQUIREMENTS:-}
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    redis:
      condition: service_healthy
    postgres:
      condition: service_healthy

services:
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"]
      interval: 10s
      retries: 5
      start_period: 5s
    restart: always

  redis:
    image: redis:7.2-bookworm
    expose:
      - 6379
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 30s
      retries: 50
      start_period: 30s
    restart: always

  airflow-webserver:
    <<: *airflow-common
    command: webserver
    ports:
      - "8080:8080"
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
      interval: 30s
```

```yaml
    timeout: 10s
    retries: 5
    start_period: 30s
  restart: always
  depends_on:
    <<: *airflow-common-depends-on
    airflow-init:
      condition: service_completed_successfully

airflow-scheduler:
  <<: *airflow-common
  command: scheduler
  healthcheck:
    test: ["CMD", "curl", "--fail", "http://localhost:8974/health"]
    interval: 30s
    timeout: 10s
    retries: 5
    start_period: 30s
  restart: always
  depends_on:
    <<: *airflow-common-depends-on
    airflow-init:
      condition: service_completed_successfully

airflow-worker-1:
  <<: *airflow-common
  command: celery worker
  healthcheck:
    test:
      - "CMD-SHELL"
      - 'celery --app airflow.providers.celery.executors.celery_executor.app inspect
      ↪  ping -d "celery@$${HOSTNAME}" || celery --app
      ↪  airflow.executors.celery_executor.app inspect ping -d "celery@$${HOSTNAME}"'
    interval: 30s
    timeout: 10s
    retries: 5
    start_period: 30s
  environment:
    <<: *airflow-common-env
    DUMB_INIT_SETSID: "0"
  restart: always
  depends_on:
    <<: *airflow-common-depends-on
    airflow-init:
      condition: service_completed_successfully

airflow-worker-2:
  <<: *airflow-common
  command: celery worker
  healthcheck:
```

```
117       test:
118         - "CMD-SHELL"
119         - 'celery --app airflow.providers.celery.executors.celery_executor.app inspect
          ↪  ping -d "celery@$${HOSTNAME}" || celery --app
          ↪  airflow.executors.celery_executor.app inspect ping -d "celery@$${HOSTNAME}"'
120       interval: 30s
121       timeout: 10s
122       retries: 5
123       start_period: 30s
124     environment:
125       <<: *airflow-common-env
126       DUMB_INIT_SETSID: "0"
127     restart: always
128     depends_on:
129       <<: *airflow-common-depends-on
130       airflow-init:
131         condition: service_completed_successfully
132
133   airflow-triggerer:
134     <<: *airflow-common
135     command: triggerer
136     healthcheck:
137       test: ["CMD-SHELL", 'airflow jobs check --job-type TriggererJob --hostname
        ↪  "$${HOSTNAME}"']
138       interval: 30s
139       timeout: 10s
140       retries: 5
141       start_period: 30s
142     restart: always
143     depends_on:
144       <<: *airflow-common-depends-on
145       airflow-init:
146         condition: service_completed_successfully
147
148   airflow-init:
149     <<: *airflow-common
150     entrypoint: /bin/bash
151     command:
152       - -c
153       - |
154         if [[ -z "${AIRFLOW_UID}" ]]; then
155           echo
156           echo -e "\033[1;33mWARNING!!!: AIRFLOW_UID not set!\e[0m"
157           echo "If you are on Linux, you SHOULD follow the instructions below to set "
158           echo "AIRFLOW_UID environment variable, otherwise files will be owned by root."
159           echo "For other operating systems you can get rid of the warning with manually
          ↪  created .env file:"
160           echo
161         fi
162         one_meg=1048576
```

```
163         mem_available=$$(($$(getconf _PHYS_PAGES) * $$(getconf PAGE_SIZE) / one_meg))
164         cpus_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)
165         disk_available=$$(df / | tail -1 | awk '{print $$4}')
166         warning_resources="false"
167         if (( mem_available < 4000 )) ; then
168           echo
169           echo -e "\033[1;33mWARNING!!!: Not enough memory available for Docker.\e[0m"
170           echo "At least 4GB of memory required. You have $$(numfmt --to iec
              ↪  $$((mem_available * one_meg)))"
171           echo
172           warning_resources="true"
173         fi
174         if (( cpus_available < 2 )); then
175           echo
176           echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for Docker.\e[0m"
177           echo "At least 2 CPUs recommended. You have $${cpus_available}"
178           echo
179           warning_resources="true"
180         fi
181         if (( disk_available < one_meg * 10 )); then
182           echo
183           echo -e "\033[1;33mWARNING!!!: Not enough Disk space available for Docker.\e[0m"
184           echo "At least 10 GBs recommended. You have $$(numfmt --to iec
              ↪  $$((disk_available * 1024 )))"
185           echo
186           warning_resources="true"
187         fi
188         if [[ $${warning_resources} == "true" ]]; then
189           echo
190           echo -e "\033[1;33mWARNING!!!: You have not enough resources to run Airflow (see
              ↪  above)!\e[0m"
191           echo "Please follow the instructions to increase amount of resources available:"
192           echo
193         fi
194         mkdir -p /sources/logs /sources/dags /sources/plugins
195         chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
196         exec /entrypoint airflow version
197     environment:
198       <<: *airflow-common-env
199       _AIRFLOW_DB_MIGRATE: 'true'
200       _AIRFLOW_WWW_USER_CREATE: 'true'
201       _AIRFLOW_WWW_USER_USERNAME: ${_AIRFLOW_WWW_USER_USERNAME:-airflow}
202       _AIRFLOW_WWW_USER_PASSWORD: ${_AIRFLOW_WWW_USER_PASSWORD:-airflow}
203       _PIP_ADDITIONAL_REQUIREMENTS: ''
204     user: "0:0"
205     volumes:
206       - ${AIRFLOW_PROJ_DIR:-.}:/sources
207
208   airflow-cli:
209     <<: *airflow-common
```

```
    profiles:
      - debug
    environment:
      <<: *airflow-common-env
      CONNECTION_CHECK_MAX_COUNT: "0"
    command:
      - bash
      - -c
      - airflow

  flower:
    <<: *airflow-common
    command: celery flower
    profiles:
      - flower
    ports:
      - "5555:5555"
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:5555/"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

volumes:
  postgres-db-volume:
```

7.8 Anexo H : API Recolha de dados OOM

```
from datetime import datetime,timedelta

from airflow import DAG

from airflow.providers.http.sensors.http import HttpSensor

from airflow.providers.postgres.hooks.postgres import PostgresHook

from airflow.operators.python import PythonOperator

from airflow.providers.http.hooks.http import HttpHook

from airflow.providers.amazon.aws.hooks.s3 import S3Hook

from airflow.providers.amazon.aws.sensors.s3 import S3KeySensor

from airflow.exceptions import AirflowException

import json
```

```
import tempfile
import os

import pandas as pd
import pandera as pa
from pandera import Column, DataFrameSchema, Check, Index


def upload_file_to_minio(data_to_minio,bucket_name,bucket_key):

    with tempfile.NamedTemporaryFile(mode='w', delete=False) as temp_file:
        json.dump(data_to_minio, temp_file)
        temp_file_path = temp_file.name

    s3_hook = S3Hook(aws_conn_id='minio_s3_conn')

    s3_client = s3_hook.get_conn()

    s3_client.upload_file(
        Filename=temp_file_path,
        Bucket=bucket_name,
        Key=bucket_key
    )
    os.remove(temp_file_path)


def fetch_data_store_raw_data():
    http_hook = HttpHook(method='GET', http_conn_id='ipma_connection')

    response = http_hook.run(endpoint='open-data/forecast/warnings/warnings_www.json')

    response_text = response.text
    all_data = []
    try:
        response_json = json.loads(response_text)
        for data in response_json:
```

```
            if data["idAreaAviso"] == "MCS":
                all_data.append(data)
    except:
        print("Failed to decode response as JSON")
        return None

    now = datetime.now()
    current_year = now.year
    current_month_name = now.strftime("%B")
    current_day = now.day
    current_hour = now.hour

    upload_file_to_minio(all_data,"rawdata","weather_warnings/" + f"{current_year}/"

    + f"{current_month_name}/" + f'weather_warnings_year_{current_year}_month

    _{current_month_name}_day_{current_day}_hour_{current_hour}.json')


def clean_alerts(data):
    # Iterate over each alert and remove 'text' if it is an empty string
    for alert in data:
        if alert.get("text") == "":
            del alert["text"]
    return data

def get_raw_data_transform_load_temp_file():
    s3_hook = S3Hook(aws_conn_id='minio_s3_conn')

    now = datetime.now()
    current_year = now.year
    current_month_name = now.strftime("%B")
    current_day = now.day
    current_hour = now.hour
    bucket_key = f"weather_warnings/{current_year}/

    {current_month_name}/weather_warnings_year_{current_year}
```

```
    _month_{current_month_name}_day_
    {current_day}_hour_{current_hour}.json"

    s3_object = s3_hook.get_key(
        key= bucket_key,
        bucket_name='rawdata'
    )

    file_content = s3_object.get()['Body'].read().decode('utf-8')
    weeather_data = json.loads(file_content)


    clean_data = clean_alerts(weeather_data)

    upload_file_to_minio(clean_data,"tempfiles",'weather_warnings.json')


def validation_layer():
    s3_hook = S3Hook(aws_conn_id='minio_s3_conn')

    s3_object = s3_hook.get_key(
        key='weather_warnings.json',
        bucket_name='tempfiles'
    )

    if s3_object is None:
        raise AirflowException("File 'weather_warnings.json' not found in 'tempfiles' bucket")

    file_content = s3_object.get()['Body'].read().decode('utf-8')

    weather_data = json.loads(file_content)

    df = pd.DataFrame(weather_data)

    if df.empty:
        raise AirflowException("DataFrame is empty after parsing JSON data")
```

```
weather_schema = DataFrameSchema(
    {
        "text": Column(
            str,
            nullable=True,
        ),
        "awarenessTypeName": Column(
            str,
            nullable=False,
        ),
        "idAreaAviso": Column(
            str,
            nullable=False,
        ),
        "startTime": Column(
            str,
            nullable=False,
        ),
        "awarenessLevelID": Column(
            str,
            nullable=False,
        ),
        "endTime": Column(
            str,
            nullable=False,
        ),
    },
    # Add a check to ensure endTime is after startTime
    checks=[
        pa.Check(
            lambda df: pd.to_datetime(df["endTime"]) > pd.to_datetime(df["startTime"]),
            error="End time must be after start time"
        )
```

```
        ],
    )

    try:
        if 'text' not in df.columns:
            df['text'] = None
        validated_df = weather_schema.validate(df)
    except pa.errors.SchemaErrors as e:
        print(json.dumps(e.message, indent=2))
        raise AirflowException("Error during validation")



def upload_data_to_postgress():
    try:
        s3_hook = S3Hook(aws_conn_id='minio_s3_conn')
        pg_hook = PostgresHook(postgres_conn_id='smartdest_production')

        s3_object = s3_hook.get_key(
            key='weather_warnings.json',
            bucket_name='tempfiles'
        )

        if s3_object is None:
            raise AirflowException("File 'weather_warnings.json'
            not found in 'tempfiles' bucket")

        file_content = s3_object.get()['Body'].read().decode('utf-8')
        weather_data = json.loads(file_content)

        connection = pg_hook.get_conn()
        cursor = connection.cursor()

        for item in weather_data:

            json_data = json.dumps(item, ensure_ascii=False)
```

```
            # 2. Use psycopg2's built-in JSON adaptation instead of bytea
            cursor.execute(
                "INSERT INTO data_lake (data_source_id, data) VALUES (%s, %s::jsonb)",
                (2, json_data)
            )

        connection.commit()
        cursor.close()
        connection.close()

    except Exception as e:
        # Log the specific error
        print(f"Error details: {str(e)}")
        raise AirflowException(f"Error happened inserting data: {str(e)}")


default_args = {
    'owner': 'filipe'
}

with DAG(
    dag_id='weather_warnings',
    default_args=default_args,
    description='Avisos Meteorológicos até 3 dias',
    start_date=datetime(2025, 4, 3),
    schedule_interval='0 13,23 * * *',
    catchup=False
) as dag:

    check_website_status = HttpSensor(
        task_id='check_website_status',
        http_conn_id='ipma_connection',
        endpoint= "open-data/forecast/warnings/warnings_www.json",
        response_check=lambda response: response.status_code == 200,
```

```
    retries=3,
    poke_interval=120,
    timeout=600,
)

testConnection = S3KeySensor(
    task_id="test_minio_connection",
    bucket_name="tempfiles",
    bucket_key="meteo_forecast.json",
    aws_conn_id="minio_s3_conn",
    deferrable=True,
    retries=3,
    retry_delay=timedelta(minutes=5),
)

fetch_data_store_raw_data_task = PythonOperator(
    task_id = "fetch_data_and_load_raw_data_to_minio",
    python_callable=fetch_data_store_raw_data,
    retries=2,
    retry_delay=timedelta(minutes=10),
)

get_raw_data_transform_load_temp_file_task = PythonOperator(
    task_id = "get_raw_data_transform_load_temp_file",
    python_callable=get_raw_data_transform_load_temp_file,
    retries=2,
    retry_delay=timedelta(minutes=10),
)

validation_layer_task = PythonOperator(
    task_id = "validation_layer_task",
    python_callable=validation_layer,
    retries=2,
    retry_delay=timedelta(minutes=10),
```

```
)

getDataFromMinioAndUpload = PythonOperator(
    task_id = "get_data_from_minio_upload_to_postgress",
    python_callable=upload_data_to_postgress,
)

[check_website_status,testConnection] >>

fetch_data_store_raw_data_task >>

get_raw_data_transform_load_temp_file_task

>> validation_layer_task

>> getDataFromMinioAndUpload
```

Referências

[1] D. Apriyandi, "A smart tourism platform reference architecture for developing countries," Master's thesis, University of Twente, 2024. [Online]. Available: http://essay.utwente.nl/98352/

[2] A. Zeqiri, M. Dahmani, and A. B. Youssef, "Digitalization of the tourism industry: What are the impacts of the new wave of technologies," *Balkan economic review*, vol. 2, pp. 63–82, 2020.

[3] T. Pencarelli, "The digital revolution in the travel and tourism industry," *Information Technology & Tourism*, vol. 22, no. 3, pp. 455–476, 2020.

[4] A. Reyes-Menendez, J. R. Saura, and F. Filipe, "The importance of behavioral data to identify online fake reviews for tourism businesses: A systematic review," *PeerJ Computer Science*, vol. 5, p. e219, 2019.

[5] A. Bustamante, L. Sebastia, and E. Onaindia, "Bitour: A business intelligence platform for tourism analysis," *ISPRS International Journal of Geo-Information*, vol. 9, no. 11, p. 671, 2020.

[6] W. Höpken and M. Fuchs, *Business Intelligence in Tourism*. Cham: Springer International Publishing, 2022, pp. 497–527. [Online]. Available: https://doi.org/10.1007/978-3-030-48652-5_3

[7] C. S. Reddy, R. S. Sangam, and B. Srinivasa Rao, "A survey on business intelligence tools for marketing, financial, and transportation services," in *Smart Intelligent Computing and Applications: Proceedings of the Second International Conference on SCI 2018, Volume 2*. Springer, 2019, pp. 495–504.

[8] T. Tran, "In-depth analysis and evaluation of etl solutions for big data processing," 2024.

[9] E. Marques and F. Gonçalves, "Smartdest project: Madeira island implementation and experience," in *International Tourism Congress ITC2022*, Łódź, Poland, November 16–19 2022.

[10] J. Li, L. Xu, L. Tang, S. Wang, and L. Li, "Big data in tourism research: A literature review," *Tourism Management*, vol. 68, pp. 301–323, 2018. [Online]. Available:

https://www.sciencedirect.com/science/article/pii/S0261517718300591

[11] A. De Mauro, M. Greco, and M. Grimaldi, "What is big data? a consensual definition and a review of key research topics," in *AIP conference proceedings*, vol. 1644, no. 1. American Institute of Physics, 2015, pp. 97–104.

[12] J. Li, L. Xu, L. Tang, S. Wang, and L. Li, "Big data in tourism research: A literature review," *Tourism management*, vol. 68, pp. 301–323, 2018.

[13] F. Xu, N. Nash, and L. Whitmarsh, "Big data or small data? a methodological review of sustainable tourism," *Journal of Sustainable Tourism*, vol. 28, no. 2, pp. 144–163, 2020.

[14] H. Li, M. Hu, and G. Li, "Forecasting tourism demand with multisource big data," *Annals of Tourism Research*, vol. 83, p. 102912, 2020.

[15] M. Sigala, R. Rahimi, and M. Thelwall, "Big data and innovation in tourism, travel, and hospitality," *Berlin: Springer. Tao, S., & Kim, HS (2017). A study of comparison between cruise tours in China and USA through big data analytics. Culinary Science & Hospitality Research*, vol. 23, no. 6, pp. 1–11, 2019.

[16] N. D. Line, T. Dogru, D. El-Manstrly, A. Buoye, E. Malthouse, and J. Kandampully, "Control, use and ownership of big data: A reciprocal view of customer big data value in the hospitality and tourism industry," *Tourism Management*, vol. 80, p. 104106, 2020.

[17] E. Kauffmann, J. Peral, D. Gil, A. Ferrández, R. Sellers, and H. Mora, "A framework for big data analytics in commercial social networks: A case study on sentiment analysis and fake review detection for marketing decision-making," *Industrial Marketing Management*, vol. 90, pp. 523–537, 2020.

[18] M. Fuchs, W. Höpken, and M. Lexhagen, "Big data analytics for knowledge generation in tourism destinations–a case from sweden," *Journal of destination marketing & management*, vol. 3, no. 4, pp. 198–209, 2014.

[19] A. Katal, M. Wazid, and R. H. Goudar, "Big data: issues, challenges, tools and good practices," in *2013 Sixth international conference on contemporary computing (IC3)*. IEEE, 2013, pp. 404–409.

[20] H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE access*, vol. 2, pp. 652–687, 2014.

[21] M. Sabou, I. Onder, A. M. Brasoveanu, and A. Scharl, "Towards cross-domain data analytics in tourism: a linked data based approach," *Information Technology & Tourism*, vol. 16, pp. 71–101, 2016.

[22] H. Mehmood, E. Gilman, M. Cortes, P. Kostakos, A. Byrne, K. Valta, S. Tekes, and J. Riekki, "Implementing big data lake for heterogeneous data sources," in *2019 ieee 35th international conference on data engineering workshops (icdew)*. IEEE, 2019, pp. 37–44.

[23] A. Raj, J. Bosch, H. H. Olsson, and T. J. Wang, "Modelling data pipelines," in *2020 46th Euromicro conference on software engineering and advanced applications (SEAA)*. IEEE, 2020, pp. 13–20.

[24] P. Pääkkönen and D. Pakkala, "Reference architecture and classification of technologies, products and services for big data systems," *Big data research*, vol. 2, no. 4, pp. 166–186, 2015.

[25] G. M. Sang, L. Xu, and P. De Vrieze, "A reference architecture for big data systems," in *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*. IEEE, 2016, pp. 370–375.

[26] ——, "Simplifying big data analytics systems with a reference architecture," in *Collaboration in a Data-Rich World: 18th IFIP WG 5.5 Working Conference on Virtual Enterprises, PRO-VE 2017, Vicenza, Italy, September 18-20, 2017, Proceedings 18*. Springer, 2017, pp. 242–249.

[27] S. Werner and S. Tai, "A reference architecture for serverless big data processing," *Future Generation Computer Systems*, vol. 155, pp. 179–192, 2024.

[28] A. Siddiqa, A. Karim, and A. Gani, "Big data storage technologies: a survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, pp. 1040–1070, 2017.

[29] T. Sahama and P. Croll, "A data warehouse architecture for clinical data warehousing," in *ACSW Frontiers 2007: Proceedings of 5th Australasian Symposium on Grid Computing and e-Research, 5th Australasian Information Security Workshop (Privacy Enhancing Technologies), and Australasian Workshop on Health Knowledge Management and Discovery*. Australian Computer Society, 2007, pp. 227–232.

[30] A. A. Harby and F. Zulkernine, "From data warehouse to lakehouse: A comparative review," in *2022 IEEE international conference on big data (big data)*. IEEE, 2022, pp. 389–395.

[31] Y. Bassil, "A data warehouse design for a typical university information system," *arXiv preprint arXiv:1212.2071*, 2012.

[32] D. Solodovnikova and L. Niedrite, "An approach to handle big data warehouse evolution," *arXiv preprint arXiv:1809.04284*, 2018.

[33] L. Yessad and A. Labiod, "Comparative study of data warehouses modeling approaches: Inmon, kimball and data vault," in *2016 International Conference on System Reliability and Science (ICSRS)*, 11 2016, pp. 95–99.

[34] I.-I. Scholtz, "Inmon versus kimball: the agile development of a data warehouse," in *Proceedings of the XYZ Data Conference 2016*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:115012843

[35] R. Liu, H. Isah, and F. Zulkernine, "A big data lake for multilevel streaming analytics," in *2020 1st International Conference on Big Data Analytics and Practices (IBDAP)*. IEEE, 2020, pp. 1–6.

[36] P. Sawadogo and J. Darmont, "On data lake architectures and metadata management," *Journal of Intelligent Information Systems*, vol. 56, no. 1, pp. 97–120, 2021.

[37] R. Hai, C. Koutras, C. Quix, and M. Jarke, "Data lakes: A survey of functions and systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 571–12 590, 2023.

[38] P. Wieder and H. Nolte, "Toward data lakes as central building blocks for data management and analysis," *Frontiers in Big Data*, vol. 5, p. 945720, Aug 19 2022. [Online]. Available: https://doi.org/10.3389/fdata.2022.945720

[39] A. M. Nambiar and D. Mundra, "An overview of data warehouse and data lake in modern enterprise data management," *Big Data Cogn. Comput.*, vol. 6, p. 132, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253428554

[40] M. R. Llave, "Data lakes in business intelligence: reporting from the trenches," *Procedia computer science*, vol. 138, pp. 516–524, 2018.

[41] B. Khan, S. Jan, W. Khan, and M. I. Chughtai, "An overview of etl techniques, tools, processes and evaluations in data warehousing." *Journal on Big Data*, vol. 6, 2024.

[42] U. Nayak, "Comparing aws glue vs. apache airflow for data orchestration: A comprehensive performance and cost analysis," *International Journal of Emerging Trends in Computer Science and Information Technology (IJETCSIT)*, vol. 6, no. 3, pp. 51–55, Sep. 2025, accessed: 2025-12-28. [Online]. Available: https://ijetcsit.org/index.php/ijetcsit/article/view/359

[43] J. Ogeawuchi, A. Uzoka, C. Alozie, O. Agboola, T. Gbenle, and S. Owoade, "Systematic review of data orchestration and workflow automation in modern data engineering for scalable business intelligence," *International Journal of Social Science Exceptional Research*, vol. 1, pp. 283–290, 01 2022.

[44] A. Simitsis, P. Vassiliadis, U. Dayal, A. Karagiannis, and V. Tziovara, "Benchmarking etl workflows," in *Performance Evaluation and Benchmarking: First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers 1.* Springer, 2009, pp. 199–220.

[45] A. Georgiev, V. Valkanov, and P. Georgiev, "A comparative analysis of jenkins as a data pipeline tool in relation to dedicated data pipeline frameworks," in *2024 International Conference Automatics and Informatics (ICAI)*, 2024, pp. 508–512.

[46] J. Wu, D. Bein, J. Huang, and S. Kurwadkar, "Etl and ml forecasting modeling process automation system," in *Human Factors in Software and Systems Engineering*, ser. AHFE Open Access, T. Ahram, Ed., vol. 94. USA: AHFE International, 2023. [Online]. Available: http://doi.org/10.54941/ahfe1003775

[47] A. S. Foundation. (2025) Architecture overview — apache airflow core concepts. Accessed: 2025-09-01. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html

[48] A. Mbata, Y. Sripada, and M. Zhong, "A survey of pipeline tools for data engineering," *arXiv preprint arXiv:2406.08335*, 2024.

[49] S. Wakchaure, "Overview of apache nifi," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 523–530, 12 2023.

[50] J. McPadden, T. J. S. Durant, D. R. Bunch, A. C. Coppi, N. Price, K. Rodgerson, C. J. Torre, W. Byron, H. P. Young, A. L. Hsiao, H. M. Krumholz, and W. L. Schulz, "A scalable data science platform for healthcare and precision medicine research," *ArXiv*, vol. abs/1808.04849, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52009953

[51] A. S. Foundation. (2024) Apache nifi overview. Accessed: 2025-09-01. [Online]. Available: https://nifi.apache.org/docs/nifi-docs/html/overview.html

[52] S. Akhund, "Computing infrastructure and data pipeline for enterprise-scale data preparation: A scalability optimization study," ADA University,George Washington University, Tech. Rep., 2023, available on ResearchGate.

[53] S. K. Singu, "Etl process automation: Tools and techniques," *ESP Journal of Engineering & Technology Advancements*, vol. 2, no. 1, pp. 74–85, 2022.

[54] M. Saxena, B. Sowell, D. Alamgir, N. Bahadur, B. Bisht, S. Chandrachood, C. Keswani, G. Krishnamoorthy, A. Lee, B. Li, Z. Mitchell, V. Porwal, M. R. Chappidi, B. Ross, N. Sekiyama, O. Zaki, L. Zhang, and M. A. Shah, "The story of aws glue," *Proc. VLDB Endow.*, vol. 16, no. 12, p. 3557–3569, Aug. 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611547

[55] S. Kona, "Optimizing data ingestion in the cloud: Leveraging aws technologies like aws s3, emr, and glue for cost efficiency and operational scalability," *Journal of Artificial Intelligence & Cloud Computing*, vol. 3, pp. 1–5, 04 2024.

[56] D. Naphade, "The evolution and modernization of data pipeline architectures," *European Journal of Computer Science and Information Technology*, vol. 13, no. 6, pp. 42–53, 2025.

[57] D. Geng, C. Zhang, C. Xia, X. Xia, Q. Liu, and X. Fu, "Big data-based improved data acquisition and storage system for designing industrial data platform," *IEEE Access*, vol. 7, pp. 44 574–44 582, 2019.

[58] M. Saadoon, S. H. A. Hamid, H. Sofian, H. H. Altarturi, Z. H. Azizul, and N. Nasuha, "Fault tolerance in big data storage and processing systems: A review on challenges and solutions," *Ain Shams Engineering Journal*, vol. 13, no. 2, p. 101538, 2022.

[59] S. Malhotra, F. Yashu, M. Saqib, D. Mehta, J. Jangid, and S. Dixit, "Evaluating fault tolerance and scalability in distributed file systems: A case study of gfs, hdfs, and minio," 2025. [Online]. Available: https://arxiv.org/abs/2502.01981

[60] D. Joiner, M. Clement, S. Chan, K. Pereira, A. Wong, Y. Khmelevsky, J. Mahony, and M. Ferri, "Dw vs oltp performance optimization in the cloud on postgresql (a case study)," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Nov. 2022, pp. 1–8.

[61] E. Simili, G. Stewart, G. Roy, S. Skipsey, and D. Britton, "A hybrid system for monitoring and automated recovery at the glasgow tier-2 cluster," *EPJ Web of Conferences*, vol. 251, p. 02047, 01 2021.

[62] U. Nayak, "Best practices for logging and monitoring big data pipelines with grafana," *International Journal of Multidisciplinary Research and Growth Evaluation*, vol. 6, no. 3, pp. 835–836, 2025, engineering.