PM4IPLeiria – Implementação de um Data Warehouse e de técnicas de Process Mining para análise de processos do Politécnico de Leiria

Mestrado em Engenharia Informática – Computação Móvel

Bruno Vieira Silva

Leiria, setembro de 2025

PM4IPLeiria - Implementação de um Data Warehouse e de técnicas de Process Mining para análise de processos do Politécnico de Leiria

Mestrado em Engenharia Informática – Computação Móvel

Bruno Vieira Silva

Relatório de Projeto da unidade curricular de Projeto realizado sob a orientação do Professor Doutor Ricardo Filipe Gonçalves Martinho

Leiria, setembro de 2025

Agradecimentos

Agradeço em primeiro lugar à minha família – aos meus pais e à minha irmã – por todo o apoio, amor e por terem criado as bases para que eu pudesse chegar até aqui. Um agradecimento especial à minha namorada, pela sua paciência, carinho e por ser o meu suporte nos momentos de maior pressão.

Ao meu orientador, Professor Doutor Ricardo Filipe Gonçalves Martinho, agradeço profundamente pela orientação, pelo rigor e pela disponibilidade demonstrada ao longo de todo o projeto. Agradeço também à minha colega Valeria Balitkaia, pela excelente colaboração e pelo espírito de equipa que foram fundamentais para o sucesso deste trabalho.

À empresa Link Consulting, a minha gratidão pela flexibilidade e pelo apoio que me permitiram conciliar o trabalho com os estudos, uma ajuda que se revelou essencial.

Por último, um muito obrigado ao Instituto Politécnico de Leiria e a todo o corpo docente do Departamento de Engenharia Informática, pela formação de excelência que me foi proporcionada e que levarei para o meu futuro profissional.

Resumo

O presente projeto, desenvolvido no âmbito do Mestrado em Engenharia Informática – Computação Móvel, aborda a análise e otimização dos processos administrativos do Instituto Politécnico de Leiria (IPLeiria). O ponto de partida foi a identificação de significativas ineficiências e não conformidades nos processos atuais da instituição, o que motivou a criação do projeto PM4IPLeiria.

Para responder a este desafio, foi concebida e implementada uma solução completa de *Business Intelligence*, assente numa arquitetura de Data Warehouse. O objetivo central foi consolidar os dados dispersos de 16 processos de negócio distintos, provenientes de Vistas SQL. Para tal, foi desenvolvido um pipeline de Extração, Transformação e Carga (ETL) automatizado, orquestrado pelo Azure Data Factory. Este pipeline executou uma lógica de transformação complexa em Python, que realizou operações de limpeza, normalização de dados, cálculo de métricas de performance baseadas em horas úteis e de custos associados. Adicionalmente, foram aplicados algoritmos de *Process Mining* da PM4Py, uma biblioteca *open-source* de referência nesta área, nomeadamente a descoberta de *Directly-Follows Graphs* (DFG) e a análise de variantes, e foram carregados os dados processados para o Data Warehouse final. A estrutura de dados resultante foi um Esquema em floco de neve composto por nove tabelas (oito de dimensão e uma de factos), populado através de estratégias de carga híbridas: incremental para dimensões simples e "delete-and-insert" por processo para garantir a consistência das métricas.

Com esta abordagem, pretende-se dotar os dirigentes do IPLeiria de uma ferramenta robusta de suporte à decisão. A solução permite monitorizar indicadores de desempenho, identificar atempadamente anomalias e, consequentemente, fundamentar ações de melhoria contínua dos processos administrativos.

Palavras-chave: Gestão de Processos, *Business Intelligence*, Data Warehouse, ETL, Power BI, Apoio à Decisão

Abstract

This Project, developed as part of the Master's program in Computer Engineering – Mobile Computing, addresses the analysis and optimization of administrative processes at the Polytechnic University of Leiria (IPLeiria). The project was prompted by the identification of significant inefficiencies and non-conformities within the institution's current workflows, which motivated the creation of the PM4IPLeiria project.

To address this challenge, a complete Business Intelligence solution was designed and implemented, based on a Data Warehouse architecture. The central objective was to consolidate scattered data from 16 distinct business processes, sourced from SQL Views. To this end, an automated Extract, Transform, Load (ETL) *pipeline* was developed, orchestrated by Azure Data Factory. This *pipeline* executed a complex transformation logic in Python, which performed data cleaning and normalization operations, calculated performance metrics based on business hours, and their associated costs. Additionally, it applied Process Mining algorithms from the PM4Py library—namely the discovery of Directly-Follows Graphs (DFG) and variant analysis—and loaded the processed data into the final Data Warehouse. The resulting data structure was a Snowflake Schema composed of nine tables (eight dimension tables and one fact table), populated through hybrid loading strategies: incremental for simple dimensions and 'delete-and-insert' per process to ensure metric consistency. Data exploration and visualization are performed using interactive dashboards developed in the Microsoft Power BI tool.

With this approach, the goal is to provide IPLeiria's managers with a robust decision-support tool. The solution enables the monitoring of performance indicators, the timely identification of anomalies, and consequently, supports data-driven actions for the continuous improvement of administrative processes.

Keywords: Process Management, Business Intelligence, Data Warehouse, ETL, Power BI, Decision Support

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Acrónimos

ACI	Azure Container Instances
ADF	Azure Data Factory
BD	Base de Dados
BPM	Business Process Management
BI	Business Intelligence
CRM	Client Relationship Management
DAX	Data Analysis Expressions
DFG	Directly-Follows Graphs
DW	Data Warehouse
ETL	Extract Transform Load
ERP	Enterprise Resource Planning
GUID	Globally Unique Identifier
IES	Instituições de Ensino Superior
IPLeiria	Instituto Politécnico de Leiria
IR	Integration Runtime
KPI	Key Performance Indicator
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
SHIR	Self-Hosted Integration Runtime
SQL	Structured Query Language
SSMS	SQL Server Management Studio
URL	Uniform Resource Locator
VPN	Virtual Private Network

1. Introdução

O presente relatório descreve o trabalho desenvolvido no âmbito do projeto final do Mestrado em Engenharia Informática – Computação Móvel, do Instituto Politécnico de Leiria, realizado durante o ano letivo de 2024/2025, com início em setembro de 2024 e término em setembro de 2025. O projeto, intitulado "PM4IPLeiria", foi desenvolvido num contexto real, visando a aplicação de tecnologias de *Process Mining* e de monitorização de processos para a sua análise e otimização.

As organizações contemporâneas, sejam elas do setor privado ou da administração pública, operam num ambiente de crescente complexidade. Instituições de ensino superior, como o Politécnico de Leiria, não são exceção. A gestão de dezenas de processos administrativos e académicos, desde a inscrição de um aluno até à submissão de uma tese de mestrado, ou mesmo à realização de uma compra, envolve uma multiplicidade de etapas, intervenientes e sistemas de informação (SI). Estes sistemas, embora eficazes no seu propósito transacional, geram um vasto rasto digital de atividades (*event logs)* que registam com precisão as ações executadas em cada processo, os seus tempos e os recursos envolvidos [1] [2].

No entanto, estes dados valiosos permanecem, na sua maioria, inexplorados. O IPLeiria, à semelhança de muitas outras organizações, enfrenta o desafio de não possuir as ferramentas analíticas adequadas para extrair conhecimento destes registos. A monitorização da performance é frequentemente reativa e baseada em métricas agregadas, faltando uma visão clara e objetiva do fluxo real dos processos *end-to-end*. Esta lacuna alinha-se com um princípio fundamental da gestão: o que não se consegue medir, não se consegue gerir nem otimizar. Estudos recentes reforçam que técnicas de *Process Mining* permitem detetar atempadamente gargalos, desvios e tempos de espera em processos concretos, muito além de métricas agregadas [3]. A dificuldade é agravada pela burocracia inerente à administração pública e pelo elevado número de processos, o que torna a análise manual impraticável e ineficiente.

A aplicação de *Process Mining* no ensino superior, é ainda uma área emergente e com pouca expressão documentada. Embora existam estudos de caso internacionais sobre a otimização de processos académicos (tal como análise de percursos de estudantes), a

aplicação sistemática desta disciplina para a melhoria de processos administrativos internos em instituições portuguesas representa uma oportunidade significativa de inovação [2] [4].

1.1.Objetivos

O objetivo principal deste projeto é desenhar e implementar uma arquitetura de *Business Intelligence* que permita a análise e otimização dos processos de negócio do Politécnico de Leiria. Para tal, pretende-se, em primeiro lugar, conceber um Data Warehouse centralizado, estruturado segundo um modelo dimensional otimizado para análise, que servirá como a "fonte única da verdade". Este repositório será populado através de um pipeline de ETL (Extração, Transformação e Carga – do inglês *Extract, Transform and Load*) automatizado, cuja responsabilidade será a de extrair, limpar e transformar os dados provenientes dos diversos sistemas de origem/fontes de dados de processos.

Com uma base de dados consolidada, o foco da análise será a aplicação de técnicas de *Process Mining*. O objetivo é utilizar os dados para descobrir os fluxos reais de processos, identificar as diferentes variantes, detetar gargalos e analisar não conformidades. Esta análise será enriquecida através do desenvolvimento de um modelo de custos e de performance temporal, agregando valor de negócio aos dados técnicos. Finalmente, os *insights* gerados por esta arquitetura de dados servirão de base para a camada de visualização, a ser materializada nos *dashboards* desenvolvidos em colaboração com a mestranda Valeria Balitkaia, culminando numa ferramenta de suporte à decisão para os gestores e dirigentes do IPLeiria. A validação da arquitetura proposta será realizada através da sua aplicação a múltiplos processos administrativos, demonstrando a sua escalabilidade e eficácia prática.

1.2. Estrutura do documento

Este documento está organizado nos seguintes capítulos: O Capítulo 1 introduz o tema, contextualiza o problema e define os objetivos do projeto. O Capítulo 2 explora o estado da arte, abordando os conceitos fundamentais de Gestão de Processos de Negócio (*Business Process Management* – BPM) e técnicas de *Process Mining*, e Data Warehousing. O Capítulo 3 detalha a metodologia de desenvolvimento, enquanto o Capítulo 4 se foca na análise de requisitos e de dados, passando pela arquitetura de software da solução e culminando no desenho detalhado e na justificação do modelo de dados do Data Warehouse. O Capítulo 5 descreve o percurso prático de implementação do Data Warehouse e o Capítulo

6 da implementação do *pipeline*, orquestração e automação da solução. O Capítulo 7, aborda a verificação da solução com a comparação de ambiente local *versus cloud* e também a demonstração visual final da solução. Finalmente, o Capítulo 8 apresenta as conclusões do trabalho, reflete sobre as limitações encontradas e aponta direções para trabalho futuro.

2. Contexto e estado da arte

Este capítulo estabelece o enquadramento teórico e conceptual que fundamenta o projeto PM4IPLeiria. O objetivo é apresentar o estado da arte nas áreas de Gestão de Processos e Process Mining, contextualizando a sua relevância e aplicando estes conceitos à realidade específica das Instituições de Ensino Superior. A secção 2.1 foca-se na caracterização dos processos no setor do ensino superior. A secção 2.2 introduz os fundamentos da Gestão de Processos de Negócio (Business Process Management - BPM) e do Process Mining, incluindo uma análise comparativa das ferramentas de mercado. A secção 2.3 apresenta uma análise de trabalhos relacionados. Finalmente, a secção 2.4 sintetiza a análise, delineando os desafios estruturais concretos do Politécnico de Leiria, que servem como a principal motivação e justificação para a solução desenvolvida.

2.1. Caracterização de processos em instituições de ensino superior

As Instituições de Ensino Superior (IES) são organizações complexas que executam um vasto e diversificado portfólio de processos. Estes podem ser categorizados em processos académicos (por exemplo, matrículas, gestão de notas, acreditação de cursos) e processos administrativos de suporte (dos quais, gestão de recursos humanos, aquisições, contabilidade). Particularmente no setor público, estes processos são caracterizados por um elevado grau de burocracia, impulsionado pela necessidade de cumprir regulamentos internos e externos rigorosos, garantindo a transparência e a auditabilidade de cada decisão.

Um dos sistemas centrais em uso no IPLeiria é um Sistema de Gestão Documental que, pela sua natureza, já incorpora características de um Sistema de Gestão de Processos de Negócio (Business Process Management System – BPMS). Cada passo de um processo formal (como "Submeter Pedido") é registado como uma etapa, com informação sobre o caso em geral, a atividade, o recurso e os tempos de entrada e de saída para a próxima etapa. Consequentemente, o IPLeiria possui um repositório rico e detalhado de dados processuais, representando um ativo de elevado valor para a análise e otimização de processos [5].

No entanto, estes sistemas são desenhados para a execução e controlo operacional, não para a análise agregada. Eles conseguem indicar o estado de um caso específico, mas não respondem a questões de gestão, tais como: "Qual o custo médio de processamento de uma bolsa de investigação no último ano?", "O tempo de aprovação de pedidos de teletrabalho

aumentou no último trimestre?" ou "Quais são as variantes mais comuns e as mais ineficientes no processo de justificação de faltas?". Esta incapacidade de medir e monitorizar a performance com indicadores específicos de processos representa o problema central que este projeto se propõe a resolver.

2.2. Fundamentos

A Gestão de Processos de Negócio, do inglês *Business Process Management* (BPM), é uma disciplina de gestão que visa descobrir, modelar, analisar, medir, melhorar e otimizar os processos de negócio de uma organização. O objetivo fundamental da BPM não é focar em tarefas individuais, mas sim na gestão do fluxo de trabalho completo, *end-to-end*, para garantir que os processos estão alinhados com os objetivos estratégicos da organização, aumentando a eficiência e a agilidade [6].

Tradicionalmente, a gestão de processos segue um ciclo de vida iterativo, como o popularizado por van der Aalst [5], que é composto por várias fases, desde o desenho do processo até à sua monitorização e otimização contínua (Figura 1). Cada fase deste ciclo possui um propósito bem definido, visando a melhoria contínua e o alinhamento dos processos com os objetivos estratégicos da organização [6].

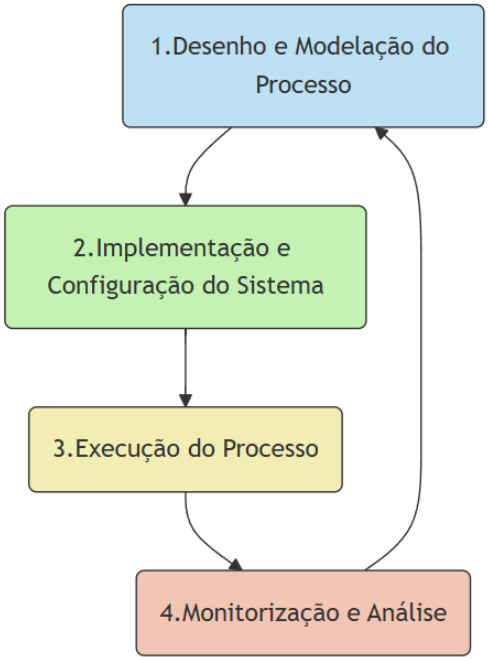


Figura 1 - Ciclo de Vida da Gestão de Processos de Negócio

As fases deste ciclo são as seguintes:

1. Desenho e Modelação do Processo: Esta é a fase conceptual, onde os processos de negócio são identificados, definidos e formalmente documentados. O objetivo é criar um modelo de processo "tal como deve ser" (*to-be model*). Este trabalho é tipicamente realizado através de *workshops* e entrevistas com os *stakeholders* (gestores, colaboradores) para mapear as atividades, as regras de decisão, os intervenientes e os fluxos de informação. O resultado é um diagrama de processo, frequentemente numa notação *standard* como a *Business Process Model and Notation (BPMN*), que serve como um plano partilhado por todas as partes interessadas.

2. Implementação e Configuração do Sistema: Nesta fase, o modelo de processo teórico é transformado numa solução executável. Isto pode envolver a configuração de um *Business Process Management System* (BPMS), o desenvolvimento de software customizado, ou a adaptação de sistemas de informação existentes (como Enterprise

Resource Planning (ERP) ou Customer Relationship Management) para suportar o fluxo de trabalho desenhado. O objetivo é automatizar o máximo possível do processo e garantir que os sistemas estão prontos para o executar conforme o modelo.

3. Execução do Processo: Esta é a fase operacional, onde o processo implementado entra em produção. Instâncias individuais do processo (chamadas de "casos") são iniciadas e executadas por intervenientes humanos e/ou sistemas automatizados, seguindo as regras definidas. É durante esta fase que os sistemas de informação geram os dados transacionais que registam cada passo da execução, criando os *event logs* que são a matéria-prima para o *Process Mining*.

4. Monitorização e Análise: Nesta fase, os dados gerados durante a execução são recolhidos e analisados para avaliar a performance do processo. O objetivo é medir o seu desempenho em função de Indicadores Chave de Performance (Key Performance ndicators - KPI) pré-definidos (por exemplo, tempo médio de execução, custo por caso, taxa de erros). A análise destes dados permite identificar se o processo está a cumprir os seus objetivos e onde existem oportunidades de melhoria. Os *insights* obtidos nesta fase servem de *input* para um novo ciclo, reiniciando a fase de (re)desenho do processo.

Apesar da sua importância, a abordagem tradicional à BPM enfrenta desafios significativos. A fase de Desenho e Modelação é frequentemente baseada em métodos manuais, como entrevistas e workshops, que resultam em modelos de processo idealizados e subjetivos. A fase de Monitorização, por sua vez, depende muitas vezes de indicadores de performance (KPIs) agregados, que não revelam os detalhes do fluxo de execução real. Esta lacuna entre o processo "como ele foi desenhado" e o processo "como ele realmente acontece" (Figura 2) é um dos principais problemas que o *Process Mining* se propõe a resolver [5].

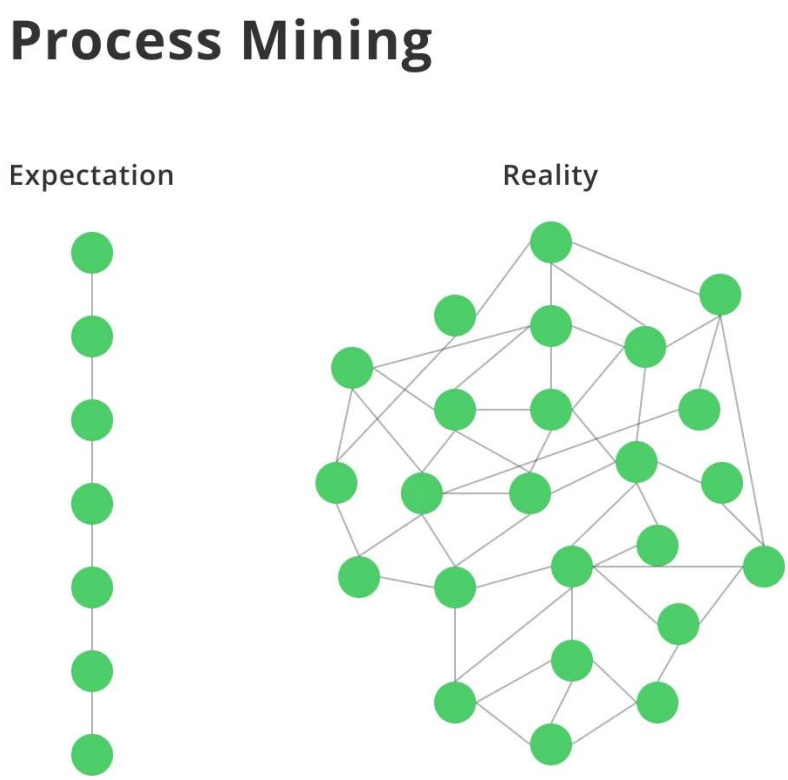


Figura 2 - Comparação entre o modelo de processo percebido (resultante da análise tradicional – à esquerda) e o modelo real (descoberto através de Process Mining – à direita) (https://botpenguin.com/glossary/process-mining)

2.2.1. Ponte entre os dados e os processos

O *Process Mining* é uma disciplina que se situa na interseção entre a ciência de dados e a gestão de processos. O seu objetivo é extrair conhecimento a partir dos *event logs* (registos de eventos) que são gerados pelos sistemas de informação de uma organização (como ERP, CRM) para descobrir, monitorizar e melhorar os processos reais [5]. Em vez de se basear em modelos teóricos, o *Process Mining* constrói modelos de processo com base em factos, ou seja, nos dados que registam cada passo da execução.

Um *event log* é, na sua essência, uma tabela onde cada linha representa um evento único que ocorreu num determinado momento, como parte de um caso específico. Para ser minimamente analisável, um *event log* deve conter, pelo menos, três colunas:

- ID do Caso**:** Um identificador que agrupa todos os eventos pertencentes à mesma instância do processo (ex: o número de um pedido).

- Atividade**:** A descrição da tarefa que foi executada (ex: "Validar Documentação").
- *Timestamp***:** A data e hora em que a atividade foi concluída.

A tabela abaixo (Tabela 1) ilustra um exemplo simplificado de um *event log* para um processo de "Pedido de Estatuto de Trabalhador-Estudante", com dois casos distintos.

Tabela 1 - Exemplo Simplificado de um *Event Log*

ID do Caso	Atividade	Timestamp	Recurso
101	Submeter Pedido	2025-09-02 10:15:00	Aluno A
101	Validar Documentação	2025-09-03 11:30:00	Secretaria
101	Emitir Parecer	2025-09-05 16:45:00	Diretor de Curso
101	Notificar Aluno	2025-09-05 17:00:00	Sistema Automático
102	Submeter Pedido	2025-09-04 09:05:00	Aluno B
102	Validar Documentação	2025-09-04 14:20:00	Secretaria
102	Pedir Doc. Adicional	2025-09-05 10:00:00	Secretaria
102	Submeter Doc. Adicional	2025-09-08 11:10:00	Aluno B
102	Validar Documentação	2025-09-09 09:30:00	Secretaria
102	Notificar Aluno	2025-09-10 15:05:00	Sistema Automático

A partir destes dados, um algoritmo de *Process Mining* consegue, por exemplo, descobrir que o Caso ‘101’ seguiu o "caminho feliz" e direto, enquanto o Caso ‘102’ teve um ciclo de *rework* (retrabalho), onde a atividade "Validar Documentação" foi executada

duas vezes. É esta capacidade de extrair fluxos de processo diretamente dos dados que confere ao *Process Mining* o seu poder analítico.

Existem três tipos principais de técnicas de *Process Mining* (Figura 3):

- Descoberta de Processos (*Process Discovery*): Esta é a técnica mais conhecida. A partir de um *event log* bruto, um algoritmo de descoberta gera automaticamente um modelo de processo (por exemplo, um *Directly-Follows Graph*). O objetivo é responder à pergunta: "O que está realmente a acontecer?".
- Verificação de Conformidade (*Conformance Checking*): Esta técnica compara um modelo de processo existente (o modelo idealizado ou prescrito) com o *event log* do mesmo processo. O objetivo é identificar desvios, violações de regras e não conformidades, respondendo à pergunta: "Estamos a seguir as regras?".
- Melhoria de Processos (*Enhancement*): Esta técnica utiliza a informação do *event log* para enriquecer um modelo de processo com dados de performance. O objetivo é identificar gargalos, analisar tempos de espera, medir a utilização de recursos e encontrar oportunidades de otimização, respondendo à pergunta: "Onde podemos melhorar?".

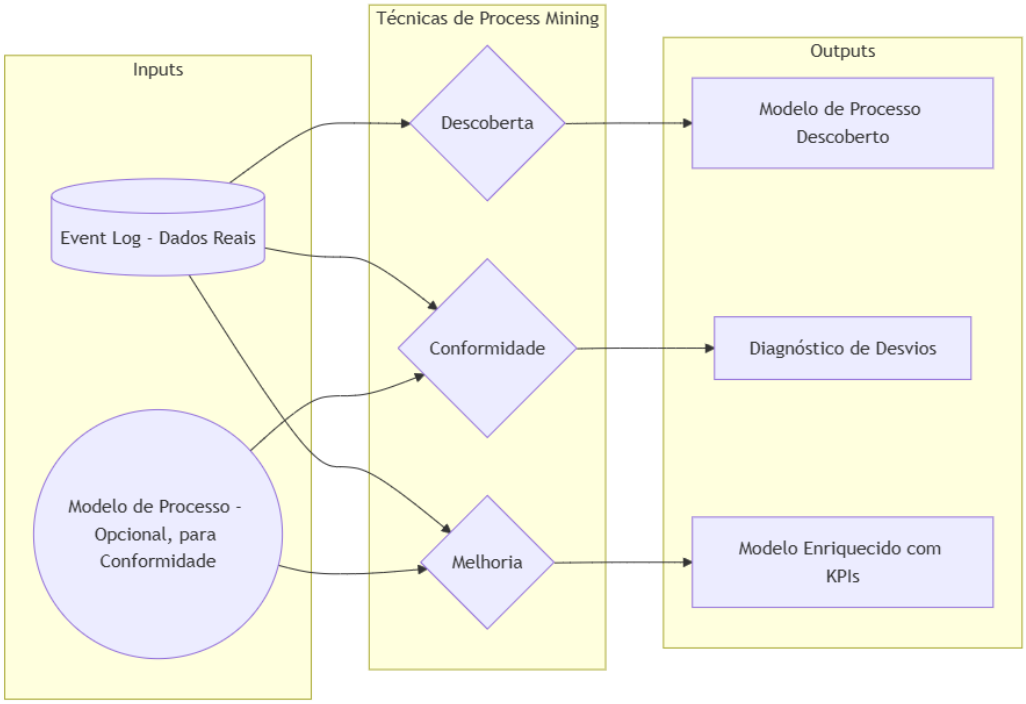


Figura 3 - Os três tipos de *Process Mining*

2.2.2. Estudo de ferramentas do mercado

Para contextualizar a solução desenvolvida, foi realizado um estudo sobre o ecossistema de ferramentas e bibliotecas de *Process Mining*. O mercado é diversificado, abrangendo desde plataformas comerciais de grande escala até ferramentas *open-source* e soluções integradas em plataformas de *Business Intelligence* (BI). A análise foi estruturada em três categorias principais, de seguida analisadas e também comparadas diretamente na tabela do Anexo A.

Plataformas comerciais integradas (*Enterprise Platforms*)

Esta categoria inclui as soluções mais robustas e líderes de mercado, que oferecem funcionalidades bastante completas que vão desde a análise à automação de processos.

- Celonis[1]: Considerada a líder de mercado, a Celonis Execution Management System (EMS) oferece uma plataforma poderosa que vai além da análise, permitindo a

[1] https://www.celonis.com/

automação de ações para otimizar processos em tempo real. É conhecida pela sua interface intuitiva e pela vasta gama de conectores para sistemas como SAP e Salesforce.

- UiPath Process Mining[2]: Integrada na plataforma de automação da UiPath, esta ferramenta foca-se na descoberta de oportunidades de automação. Ao analisar os processos existentes, a plataforma identifica os melhores candidatos para serem automatizados com *Robotic Process Automation* (RPA).

- SAP Signavio[3]: Adquirida pela SAP, a Signavio oferece um conjunto de ferramentas abrangente para a gestão de processos de negócio (BPM), onde o *Process Mining* é uma componente chave. A sua grande vantagem é a integração nativa com os ecossistemas SAP.

Ferramentas académicas e bibliotecas *open-source*

Este grupo representa a base académica e de investigação da área, oferecendo máxima flexibilidade e transparência.

- ProM (Process Mining Framework)[4]: É a ferramenta *open-source* original e a mais extensa, desenvolvida no meio académico (liderada pela Eindhoven University of Technology). Funciona como uma *framework* com centenas de *plug-ins* que implementam uma vasta gama de algoritmos. É extremamente poderosa para a investigação, mas a sua curva de aprendizagem é acentuada.

- PM4Py[5]: É a principal biblioteca *open-source* de *Process Mining* para o ecossistema Python. Mantida por académicos e pela comunidade, implementa os algoritmos mais importantes da área e é desenhada para se integrar perfeitamente com outras bibliotecas de *Data Science* como o Pandas, sendo ideal para a construção de soluções customizadas como a desenvolvida nesta tese.

[2] https://www.uipath.com/product/process-mining

[3] https://www.signavio.com/

[4] https://promtools.org/

[5] https://processintelligence.solutions/pm4py

- Disco[6]: Desenvolvida pela Fluxicon, uma empresa com fortes raízes académicas, a Disco é uma ferramenta de *desktop* focada na usabilidade e na análise exploratória rápida. É conhecida pela sua performance e interface intuitiva, sendo uma excelente porta de entrada para a disciplina.

Soluções integradas com Microsoft Power BI

Com a popularização do Power BI, surgiram soluções de *Process Mining* desenhadas para operar diretamente dentro do seu ecossistema.

- Power Automate Process Mining[7]: A solução nativa da Microsoft. Integrada no ecossistema Microsoft Fabric, permite a criação de relatórios de *Process Mining* dentro do Power BI, utilizando o Power Automate como motor de análise. Como avaliado neste projeto, a sua utilização implica custos de licenciamento associados à "Fabric Capacity".

- PAFnow[8]: Uma solução de terceiros que funciona como um conjunto de visuais customizados (*custom visuals*) e um modelo de dados para o Power BI. Permite aos utilizadores construir relatórios de *Process Mining* diretamente no Power BI Desktop, aproveitando o motor VertiPaq para a performance.

2.3. Trabalhos relacionados

A aplicação de técnicas de Data Science no ensino superior tem vindo a crescer, mas com um foco predominante na análise preditiva do sucesso académico, um campo conhecido como Educational Data Mining (EDM). No entanto, a aplicação de Process Mining para a otimização de processos administrativos internos em IES, especialmente em Portugal, é uma área consideravelmente menos explorada.

Internacionalmente, vários estudos recentes ilustram o potencial da *Process Mining* em diferentes componentes académicos e operacionais:

[6] https://fluxicon.com/disco/

[7] https://www.microsoft.com/power-platform/products/power-automate

[8] https://appsource.microsoft.com/product/power-bi-visuals/wa200000072

- No estudo *Cross-course Process Mining of Student Clickstream Data – Aggregation and Group Comparison* [7], analisaram *logs* de atividades de Moodle em diversos cursos, comparando grupos de estudantes de alto e baixo desempenho. Demonstraram que estudantes com melhor desempenho interagem mais com as atividades disponibilizadas, exploram caminhos de aprendizagem mais dinâmicos, e que a agregação de eventos a nível de secções de curso facilita a comparação entre cursos e a descoberta de padrões mais evidentes.
- No estudo *Leveraging Process Mining to Identify Bottlenecks in Online Applications* [8], aplicaram *Process Mining* a um processo administrativo de candidaturas online, identificando gargalos no fluxo de submissão que poderiam ser genéricos a outras IES ou contextos similares. Este trabalho mostra que *Process Mining* não serve apenas para perfis académicos, mas também para otimização de processos institucionais.
- No artigo *Understanding Student Learning Pathways in Traditional Online History Courses* [9], utilizaram dados de *clickstream* para identificar táticas de aprendizagem em cursos com opção de modalidades diferentes (instrutor-liderado *versus* autodirigido). Os resultados indicam que escolhas de percurso e interação com recursos variam fortemente com a estrutura do curso e o *design* pedagógico.
- Outras investigações, como *Detecting Changes in Student Behavior from Clickstream Data* (*Learning Analytics*) [10], aplicam métodos estatísticos e de aprendizagem para observar alterações no comportamento dos estudantes ao longo do tempo, relacionando-as com desempenho ou abandono. Estes estudos reforçam que os padrões de uso (interação, frequência, sequência de atividades) são indicadores importantes para além das métricas tradicionais (notas, aprovação).

No contexto nacional, a análise de dados em IES portuguesas tem seguido uma tendência semelhante, com um foco no percurso do estudante. A Universidade do Porto, por exemplo, possui iniciativas consolidadas de análise de dados para monitorizar a trajetória dos seus estudantes e o percurso profissional dos seus diplomados [11].

2.4. Desafios estruturais no contexto do IPLeiria

A análise do estado da arte, conjugada com o diagnóstico do contexto operacional do Politécnico de Leiria, revela um conjunto de desafios estruturais que limitam a capacidade

da instituição para gerir os seus processos de forma eficiente e informada. A superação destes desafios não só constitui a principal motivação para o presente trabalho, como também estabelece os pilares conceptuais sobre os quais a metodologia e os requisitos da solução foram edificados.

A fragmentação dos dados processuais por múltiplos sistemas de informação legados e heterogéneos resulta na formação de "silos de dados". Esta condição inviabiliza a criação de uma visão integrada e longitudinal (*end-to-end*) dos processos, tornando extremamente complexo rastrear a jornada completa de uma instância, desde a sua génese até à sua conclusão, de forma sistemática e fidedigna.

A avaliação da performance processual assenta, predominantemente, em metodologias manuais, como entrevistas e análise documental. Consequentemente, o conhecimento sobre os processos é frequentemente qualitativo, subjetivo e suscetível a vieses cognitivos, representando uma visão idealizada que raramente corresponde à complexidade da execução real. A ausência de uma base factual e quantitativa impede a identificação rigorosa de desvios, gargalos e outras ineficiências. A incapacidade de medir sistematicamente a performance dos processos impede a sua gestão eficaz. A instituição carece de um léxico *standardizado* de Indicadores Chave de Performance (KPI), como o tempo de ciclo (*cycle time*), o custo por instância, ou a taxa de retrabalho (*rework rate*). Esta lacuna métrica fomenta uma cultura de gestão reativa, baseada em eventos isolados, em detrimento de uma abordagem proativa e orientada para a melhoria contínua.

O volume e a complexidade dos processos administrativos geram um fluxo contínuo e massivo de eventos. A aplicação de métodos de análise manual a esta escala é logisticamente impraticável, economicamente inviável e inerentemente não escalável. Para que a análise de processos seja sustentável e contínua, é imperativa a adoção de uma solução automatizada, capaz de processar estes volumes de dados de forma regular e fidedigna.

3. Metodologias e ferramentas

Neste capítulo, são apresentadas as fundações metodológicas e tecnológicas que sustentaram o desenvolvimento do projeto PM4IPLeiria. A secção 3.1 detalha a metodologia de gestão de projeto híbrida e ágil adotada. A secção 3.2 apresenta o *framework* CRISP-DM, que orientou as fases de análise de dados. Finalmente, a secção 3.3 aprofunda o ciclo de vida do desenvolvimento de software implementado, abordando o controlo de versões, a gestão de ambientes e a automação da integração e entrega (CI/CD).

3.1. Metodologia de gestão e desenvolvimento do projeto

O desenvolvimento do projeto PM4IPLeiria foi um esforço colaborativo, envolvendo a sinergia entre duas dissertações de mestrado complementares: a presente, focada na engenharia de dados e na implementação do *pipeline* de ETL e *Process Mining*, e a da colega Valeria Balitkaia, focada na camada de visualização e análise de negócio em Power BI (ferramenta de *Business Intelligence* utilizada para a camada de apresentação, permitindo a criação de um modelo de dados e de relatórios interativos para a exploração dos resultados). Dada a natureza exploratória do projeto e a dimensão da equipa, a adoção de uma *framework* de gestão de projeto revelou-se inadequada.

Em vez disso, foi adotada uma abordagem de desenvolvimento híbrida, fortemente influenciada pelos princípios do manifesto Ágil. Esta metodologia privilegiou a flexibilidade, a comunicação constante e a entrega de valor iterativa, permitindo que a solução fosse construída e refinada de forma orgânica. A gestão do projeto assentou nos seguintes pilares:

- Planeamento iterativo: O projeto foi dividido em fases lógicas e de alto nível (por exemplo, configuração da infraestrutura, ingestão de dados, transformação, modelação dimensional, automação). Dentro de cada fase, o trabalho foi organizado em ciclos curtos e iterativos. Após a conclusão de cada ciclo (como é o caso de, após a implementação bem-sucedida da carga para uma nova tabela de dimensão), o resultado era validado antes de se avançar para o passo seguinte.
- Comunicação e colaboração constantes: Foram realizadas reuniões de acompanhamento semanais com o orientador do projeto. Estas reuniões serviam

como um ponto de controlo fundamental para apresentar o progresso, discutir os desafios técnicos encontrados (como os problemas de integração entre serviços), validar as decisões de arquitetura e planear as próximas etapas. A colaboração com a colega Valeria Balitkaia foi igualmente crucial. A estrutura do Data Warehouse, nomeadamente o esquema das tabelas de dimensão e factos, funcionou como o "contrato" de interface entre os dois projetos. A definição deste esquema foi um processo colaborativo para garantir que a camada de dados respondia eficazmente aos requisitos da camada de visualização.

- Desenvolvimento Focado em Componentes: O trabalho foi claramente dividido em dois grandes componentes:

    1. O pipeline de dados (*backend*): O foco desta tese, responsável por toda a lógica de Extract, Transform and Load (ETL), *Process Mining* e a construção do Data Warehouse.

    2. A camada de visualização (*frontend*): O foco da tese da colega Valeria Balitkaia, responsável por consumir os dados do Data Warehouse e apresentá-los de forma intuitiva no num dashboard de Power BI. Esta separação clara de responsabilidades, com o Data Warehouse a servir de ponto de integração, permitiu que ambos os projetos evoluíssem em paralelo com um mínimo de dependências bloqueantes.

- Validação Contínua: A validação foi tendo um processo contínuo. Após cada etapa de implementação significativa (nomeadamente, a carga bem-sucedida de uma nova fonte de dados), eram executadas queries de verificação diretamente na base de dados para validar a integridade, a consistência e a correção dos dados, garantindo a qualidade da fundação sobre a qual a camada de análise seria construída.

3.2. CRISP-DM

Dada a natureza exploratória e centrada nos dados deste projeto, foi adotada a metodologia CRISP-DM (Cross-Industry Standard Process for Data Mining). Esta metodologia é o padrão industrial mais amplamente utilizado para projetos de ciência de dados e análise, fornecendo um roteiro estruturado que foca na compreensão do negócio e

na iteração contínua [12]. O ciclo de vida do CRISP-DM é composto por seis fases, que foram seguidas de forma iterativa ao longo do projeto (Figura 4).

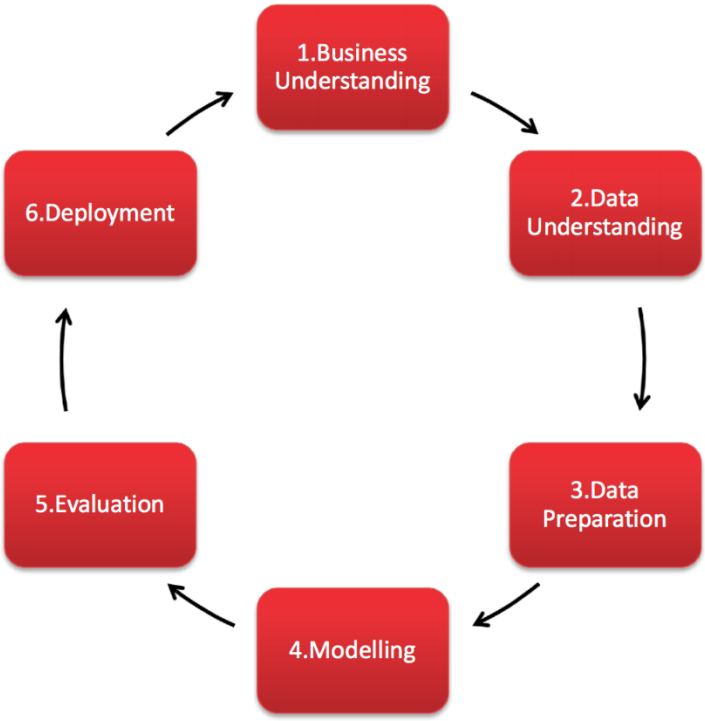


Figura 4 - Fases da metodologia CRISP-DM (https://www.sv-europe.com/product/an-introduction-to-crisp-dm/)

1. Compreensão do negócio (*business understanding*): A fase inicial focou-se na compreensão dos objetivos e requisitos do Politécnico de Leiria. Foram realizadas reuniões para identificar os processos de negócio críticos, as questões analíticas a serem respondidas (por exemplo "Onde estão os gargalos?", "Quais os custos associados?") e os critérios de sucesso do projeto.
2. Compreensão dos dados (*data understanding*): Esta fase envolveu a exploração inicial das fontes de dados. Foram analisadas as bases de dados de origem para identificar as colunas relevantes para o *Process Mining*, como o identificador do caso (id), a atividade (Etapa), os timestamps e os Recursos (atributo Despachado Por). A qualidade inicial dos dados foi avaliada, identificando desafios como valores nulos, inconsistências textuais e a necessidade de normalização.
3. Preparação dos dados (*data preparation*): Esta foi uma das fases mais intensivas do projeto, correspondendo à etapa de Transformação (T) do pipeline de ETL. O código desenvolvido realiza múltiplas tarefas de preparação, incluindo a limpeza de dados (tratamento de nulos e espaços em branco), normalização de chaves de negócio

(como, nomes de atividades), cálculo de novas métricas (nomeadamente, tempos de execução em horas úteis, custos) e a estruturação dos dados para a carga no Data Warehouse.

4. Modelação (*modeling*): Nesta fase, foram aplicadas as técnicas de *Process Mining*. Foram gerados modelos de processo, e foram calculados atributos analíticos, como as variantes de processo. Simultaneamente, foi desenhado e implementado o modelo de dados do Data Warehouse, seguindo a metodologia dimensional.
5. Avaliação (*evaluation*): Os resultados gerados foram continuamente avaliados em função dos objetivos de negócio definidos na primeira fase. Os modelos de processo foram analisados para verificar se respondiam às questões dos gestores. A estrutura do Data Warehouse foi validada para garantir que suportava as análises pretendidas através de uma ferramenta de *Business Intelligence*. Esta fase levou a vários ciclos de iteração, refinando os cálculos de custo e a estrutura das dimensões.
6. Implementação (*deployment*): A fase final consistiu em operacionalizar toda a solução.

3.3. Ambientes e ciclo de vida do desenvolvimento

Para além das metodologias de gestão e de análise, a implementação de um software robusto requer um ciclo de vida de desenvolvimento estruturado, com uma clara separação de ambientes e um controlo de versões rigoroso.

3.3.1. Controlo de versões

Todo o código-fonte do projeto foi gerido através do sistema de controlo de versões distribuído Git, com um repositório centralizado na plataforma GitHub. Foi adotada uma estratégia de ramificação (*branching strategy*) simples, onde o ramo main representa a versão estável e testada do código, e ramos secundários (*feature branches*) são criados para o desenvolvimento de novas funcionalidades ou para a correção de erros. Todas as alterações foram registadas em *commits* atómicos e depois integradas no ramo principal, garantindo um histórico completo e a rastreabilidade de todas as modificações.

3.3.2. Ambientes de desenvolvimento e teste

O trabalho foi organizado em dois ambientes distintos:

1. Ambiente de desenvolvimento (local): A estação de trabalho local serviu como ambiente primário de desenvolvimento. Neste ambiente, o código era escrito, depurado e executado em pequena escala, muitas vezes contra um subconjunto de dados ou uma base de dados de teste local.

2. Ambiente de integração e *staging*: O ambiente na *cloud* serviu como o ambiente de *staging* e de testes de integração. Após o desenvolvimento local, uma nova versão do código era empacotada e implementada neste ambiente. O seu propósito era validar o funcionamento da solução completa numa configuração idêntica à de produção, testando a interação entre todos os serviços *cloud* e a execução do *pipeline* de orquestração completo. É neste ambiente que os testes de carga com volumes de dados realistas eram realizados.

Para garantir que o *pipeline* seria executado de forma consistente e fiável, independentemente do ambiente, foi adotada a tecnologia de containerização Docker (tecnologia utilizada para empacotar a aplicação e todas as suas dependências). O uso de Docker garante que o ambiente de execução é consistente, reprodutível e portátil.

3.3.3. Ciclo de vida do código

O fluxo de trabalho para cada nova alteração seguia um ciclo definido: desenvolvimento em ambiente local, controlo de versões com Git/GitHub, empacotamento do artefacto de software (imagem Docker), e finalmente, a implementação e validação no ambiente de *staging* na *cloud*. Este processo iterativo garantiu que apenas o código validado e estável fosse considerado para a "versão de produção" do *pipeline*.

3.3.4. Automação da integração

Um pilar fundamental deste projeto é a automação dos processos de construção e entrega, uma prática conhecida como CI/CD (Continuous Integration / Continuous Deployment). A adoção de um *pipeline* de CI/CD foi crucial para garantir a consistência, reduzir o risco de erro humano e acelerar o ciclo de desenvolvimento, alinhando-se com os princípios do movimento DevOps [13].

4. Requisitos e arquitetura de software da solução

Com a base metodológica estabelecida no capítulo anterior, este capítulo foca-se no desenho da solução técnica, traduzindo as necessidades do negócio numa arquitetura de software e de dados concreta. A secção 4.1 inicia com a definição dos requisitos funcionais que o sistema deve cumprir. De seguida, a secção 4.2 apresenta uma visão geral da arquitetura da solução e das suas camadas lógicas. A secção 4.3 aprofunda esta visão através de uma decomposição formal com o Modelo C4, detalhando a arquitetura em diferentes níveis de abstração. A secção 4.4 foca-se no coração da solução: a modelação do Data Warehouse, detalhando o seu modelo físico, justificando as decisões de *design* e validando a sua performance. Finalmente, a secção 4.5 conclui o capítulo com uma análise das plataformas alternativas que foram consideradas.

4.1. Levantamento de requisitos

A fase de levantamento de requisitos para um projeto de *Data Warehousing* e de lógica de negócio de Process Mining difere da de uma aplicação transacional. O foco do levantamento de requisitos afasta-se das funcionalidades de inserção ou modificação de dados (típicas de sistemas transacionais), para se concentrar na capacidade do sistema em responder a questões de negócio complexas através da exploração interativa de dados e da visualização de resultados de *Process Mining*.

4.1.1. Requisitos funcionais

Com base nos perfis e nos objetivos do projeto, foram definidos os seguintes requisitos funcionais, que o Data Warehouse e os relatórios do Power BI devem ser capazes de satisfazer:

- **RF01 - Descoberta de processo:** O sistema deve ser capaz de descobrir e visualizar o modelo de processo real a partir dos dados, mostrando as atividades e as transições entre elas. A funcionalidade principal aqui é permitir que um gestor ou analista veja um diagrama (como um DFG) que represente a execução tal como ela é, em contraste com o modelo teórico, identificando visualmente os caminhos mais frequentes e as exceções.

- **RF02 - Análise de variantes:** O sistema deve identificar todas as sequências únicas de atividades (variantes) que ocorrem num processo e permitir a sua análise por frequência e performance. Um analista deve poder, por exemplo, focar-se nas 10 variantes mais comuns para entender o "caminho feliz", ou isolar as variantes mais longas e raras para investigar casos excecionais.

- **RF03 - Análise de performance temporal:** O sistema deve calcular e apresentar métricas de tempo, como a duração total de cada caso, o tempo de execução e o tempo de espera de cada atividade. Isto é crucial para que um gestor consiga, por exemplo, identificar gargalos, respondendo a perguntas como "Qual é o tempo médio de espera entre a 'Atividade A' e a 'Atividade B'?".

- **RF04 - Análise de custos:** O sistema deve ser capaz de atribuir custos a cada evento e agregá-los ao nível do caso e do processo. Este requisito materializa-se num cenário de otimização de recursos: um gestor de departamento deve poder utilizar o *dashboard* para filtrar pela sua Unidade Orgânica, visualizar um gráfico que ordena os processos por custo total médio por caso, e assim identificar o "Processo X" como o mais oneroso, justificando uma análise mais aprofundada.

- **RF05 - Análise de conformidade e *rework*:** O sistema deve fornecer mecanismos para identificar ineficiências como atividades repetidas (*rework*) e desvios do fluxo padrão. Este requisito suporta um cenário de análise de qualidade: um analista que suspeita que o processo de "Justificação de Faltas" tem muitas idas e vindas deve poder filtrar as variantes que contêm *rework* (através da FlagRework) e constatar que, por exemplo, 30% dos casos envolvem repetições que duplicam o tempo médio de resolução, providenciando assim uma justificação baseada em dados para uma revisão do procedimento.

- **RF06 - Análise multidimensional:** O sistema deve permitir que todas as métricas (tempo, custo, contagem) sejam filtradas e segmentadas de forma interativa por diferentes dimensões de negócio. Um utilizador deve poder selecionar um Processo, um Ano e uma Unidade Orgânica específicos e ver todos os KPIs e visuais a serem recalculados em tempo real para refletir apenas esse contexto, permitindo uma exploração de dados fluida e a descoberta de padrões específicos para cada dimensão.

4.2.Arquitetura da solução

Ao contrário da adoção de uma plataforma comercial única, o projeto PM4IPLeiria baseou-se na conceção e implementação de uma arquitetura de dados moderna e customizada, utilizando um *stack* tecnológico flexível e robusto. Esta abordagem permitiu um controlo total sobre o fluxo de dados e a lógica de negócio, resultando numa solução adaptada às necessidades do projeto. A arquitetura é composta pelos seguintes pilares (Figura 5):

- Data Warehouse: Foi implementado um repositório central de dados seguindo as práticas de modelação dimensional de Ralph Kimball [14]. Este modelo, com tabelas de factos e dimensões, está otimizado para consultas analíticas complexas e garante a integridade e consistência dos dados.

- *Pipeline* de ETL: O motor da arquitetura. Um pipeline de ETL foi desenvolvido para orquestrar todo o fluxo de dados. A ingestão é gerida por um serviço de orquestração na *cloud*, enquanto a transformação e a carga para o Data Warehouse são executadas por um conjunto de *scripts* customizados, garantindo a máxima flexibilidade na aplicação de regras de negócio.

- Motor de análise (*Process Mining)*: O cérebro analítico do projeto. Foi utilizada uma biblioteca *open-source* para a aplicação de algoritmos de *Process Mining*. Esta escolha permitiu a implementação de análises customizadas, como a descoberta de fluxos de processo, análise de variantes e cálculo de estatísticas de performance.

- Infraestrutura *cloud*: Toda a solução assenta numa infraestrutura totalmente baseada na *cloud*, que fornece um conjunto de serviços geridos, "*serverless*" e escaláveis. Estes serviços são responsáveis por:

    - Um serviço de computação *serverless* para a execução sob demanda do código de transformação, empacotado em containers.

    - Um registo de containers privado para o armazenamento seguro e versionado das imagens da aplicação.

    - Um serviço de gestão de segredos centralizado para o armazenamento seguro de credenciais.

    - Um serviço de armazenamento de objetos para a persistência de artefactos não estruturados, como os modelos visuais de processo gerados.

- Camada de visualização: A interface final para o utilizador. Os dados do Data Warehouse são consumidos por uma ferramenta de *Business Intelligence* para a criação de *dashboards* interativos, permitindo aos gestores explorar os modelos de processo, analisar KPIs, e filtrar os dados através de múltiplas dimensões. O desenvolvimento deste passo na arquitetura foi realizado pela colega Valeria Balitkaia, estudante do mestrado Ciência de Dados.

A arquitetura da solução foi desenhada para ser moderna, escalável e robusta, seguindo os princípios de um Data Warehouse dimensional. A separação em camadas lógicas garante a manutenibilidade e a flexibilidade do sistema.

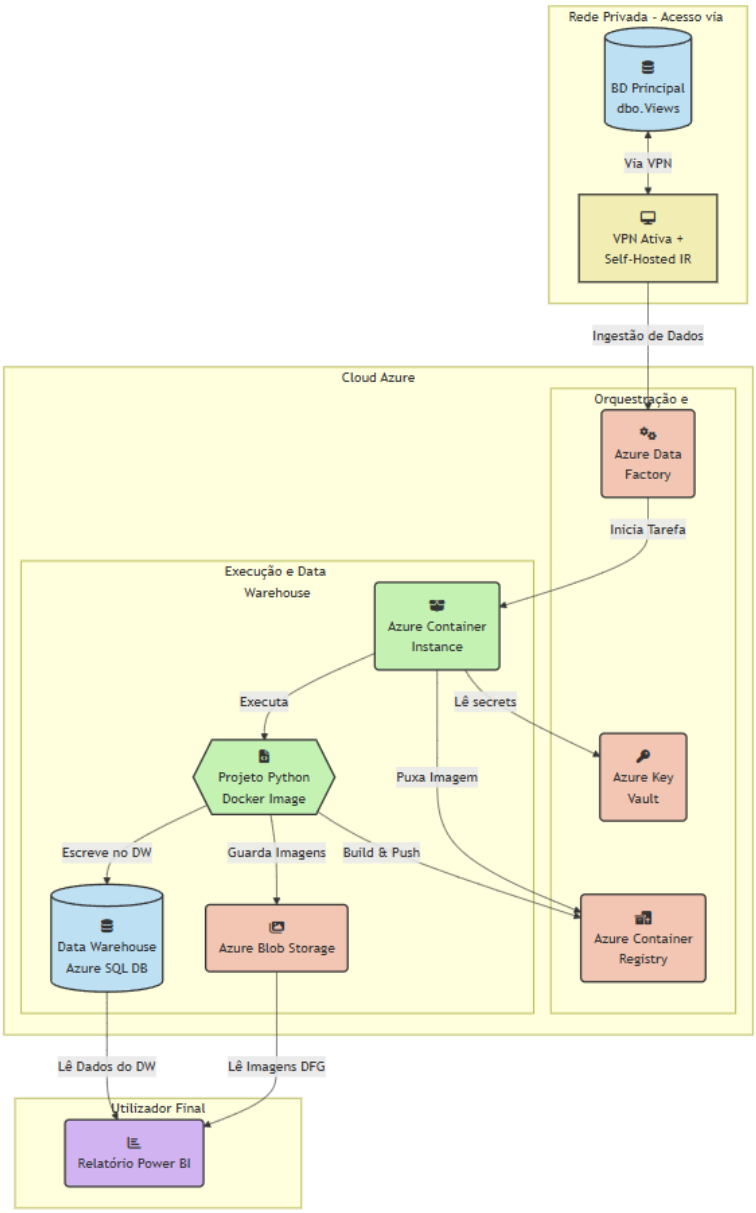


Figura 5 - Diagrama da arquitetura da solução PM4IPLeiria

4.2.1. Camada de ingestão (*Staging*)

A ingestão de dados é orquestrada pelo Azure Data Factory (utilizado como o orquestrador central do *pipeline* de ETL, responsável por agendar e gerir o fluxo de dados desde a ingestão até à transformação) que lida com a extração de dados de múltiplas fontes. Para fontes de dados em redes privadas, é utilizado um *Self-Hosted Integration Runtime* (SHIR) que atua como uma ponte segura. A estratégia de carga para a camada de *staging* na base de dados Azure SQL (serviço de base de dados relacional gerido que aloja o Data Warehouse) foi definida como "*Truncate and Load*", garantindo que, a cada execução, o *staging* é uma fotografia exata e consistente dos dados de origem.

4.2.2. Camada de armazenamento e processamento (Data Warehouse)

O núcleo da arquitetura é um Data Warehouse implementado numa base de dados Azure SQL. O modelo de dados segue a metodologia dimensional de Ralph Kimball, resultando num esquema em floco de neve (*Snowflake Schema*). Esta escolha foi deliberada: enquanto um esquema em estrela (*Star Schema*) é frequentemente preferido pela sua simplicidade, a normalização da dimensão de variantes para uma DIM_VARIANTE separada foi uma otimização crítica. Como a *string* que descreve uma variante pode ser extremamente longa, esta abordagem de "*snowflaking*" reduz drasticamente a redundância de dados e o tamanho da DIM_CASO, melhorando a performance e permitindo análises mais ricas sobre as próprias variantes [14]. O uso de chaves substitutas inteiras para todas as dimensões, em vez de chaves de negócio em texto, otimiza ainda mais a performance das consultas.

Na infraestrutura *cloud* foram utilizados serviços "*serverless*" e escaláveis, incluindo

- Azure Container Instances, serviço de computação que executa os *containers* Docker, eliminando a necessidade de gerir máquinas virtuais, para a execução do código Python;
- Azure Container Registry, repositório privado para armazenar e gerir as imagens Docker da aplicação;
- Azure Key Vault, utilizado para a gestão centralizada e segura de todos os *secrets* da aplicação (como *passwords*, *connection strings*)
- Azure Blob Storage, para o armazenamento de objetos, utilizado para guardar os artefactos visuais gerados pelo processo (as imagens DFG - *directly-follows graph*).

4.2.3. Camada de execução e análise

A lógica de transformação e *Process Mining* é executada por um *script* Python, empacotado numa imagem Docker para garantir a portabilidade e a consistência do ambiente.

Foi criado um Dockerfile (Figura 6) que define a "receita" para construir a imagem da aplicação. Este ficheiro automatiza a criação de um ambiente Linux (Ubuntu 22.04), a instalação de todas as dependências de sistema (como o driver Microsoft ODBC e o Graphviz), a configuração do Python e a instalação de todas as bibliotecas Python listadas no requirements.txt.

```
FROM ubuntu:22.04

# Define variáveis de ambiente para evitar prompts interativos
ENV DEBIAN_FRONTEND=noninteractive
ENV ACCEPT_EULA=Y

# Instala o Python, pip, e TODAS as dependências de sistema, incluindo graphviz      You, 3 weeks ago • Uncommitted c
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    curl \
    gnupg \
    graphviz \
    python3-pip \
    python3.10 \
    unixodbc-dev \
    && rm -rf /var/lib/apt/lists/* && \
    ln -s /usr/bin/python3.10 /usr/bin/python && \
    curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add - && \
    curl https://packages.microsoft.com/config/ubuntu/22.04/prod.list > /etc/apt/sources.list.d/mssql-release.list &&
    apt-get update && apt-get install -y msodbcsql17 && rm -rf /var/lib/apt/lists/*

# Copia e instala as dependências Python
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt


COPY main.py ./
COPY database_connection.py ./
COPY db_utils.py ./
COPY azure_utils.py ./
COPY data_extraction.py ./
COPY process_mining.py ./
COPY dw_builder.py ./
COPY time_utils.py ./


ENTRYPOINT ["python", "main.py"]
```

Figura 6 - Excerto do Dockerfile que demonstra a instalação de dependências de sistema e Python

As imagens Docker construídas são armazenadas num repositório privado no Azure Container Registry. O uso de *tags* de versão (no caso, v1.0) para cada nova *build* da imagem é uma prática fundamental de controlo de versões que garante a rastreabilidade e a capacidade de fazer *rollback* para uma versão anterior estável, se necessário.

Ao ser orquestrado pelo Azure Data Factory (ADF), o Azure Container Instances (ACI) utiliza esta imagem para criar um ambiente de execução limpo, isolado e idêntico a cada vez, eliminando a clássica fonte de erros "no meu PC funciona".

Para CI/CD (Continuous Integration / Continuous Deployment), foi utilizada a plataforma GitHub Actions. Esta ferramenta permite a criação de *workflows* automatizados, definidos em ficheiros de configuração no formato YAML, que são despoletados por eventos no repositório Git, como um *push* para o ramo principal. O *pipeline* de CI/CD implementado, referido na literatura como um *pipeline* de Entrega (*Deployment Pipeline*), é a manifestação

automatizada do processo de levar o software desde o controlo de versões até ao ambiente de execução [15]. Ele executa as seguintes etapas de forma totalmente automática (Figura 7):

1. *Trigger* e integração contínua: O *workflow* é automaticamente iniciado sempre que um novo *commit* é enviado (git push) para o ramo *main*. Esta prática, definida por Martin Fowler, garante que cada alteração ao código é imediatamente integrada e validada por uma construção automática, permitindo a deteção precoce de erros [16].

2. Gestão segura de credenciais**:** A autenticação com os serviços Azure, como o Azure Container Registry (ACR), é feita de forma segura. As credenciais não estão armazenadas no código nem no ficheiro YAML. Em vez disso, são guardadas como GitHub Secrets, um cofre encriptado associado ao repositório.

3. Construção do artefacto imutável: O *workflow* utiliza o Dockerfile do projeto para construir a imagem Docker. Este processo cria um artefacto imutável, que empacota a aplicação Python e todas as suas dependências num pacote único e autocontido. A criação de artefactos que são construídos *uma única vez* e depois promovidos através dos ambientes é um princípio fundamental da entrega contínua [15].

4. Versionamento do artefacto (*tagging*): Em vez de usar uma *tag* genérica como :latest, que é volátil, o *pipeline* atribui à nova imagem uma *tag* única e imutável: o *hash* SHA do *commit* do Git que despoletou a execução (por exemplo, :a1b2c3d4...). Esta prática garante uma ligação inequívoca entre o artefacto e a versão exata do código que o gerou, o que é fundamental para a depuração e para a capacidade de fazer *rollback* de forma fiável.

5. Entrega no registo de artefactos**:** Finalmente, a nova imagem versionada é enviada (docker push) para o repositório privado no Azure Container Registry, onde fica disponível para ser utilizada pelo Azure Data Factory.

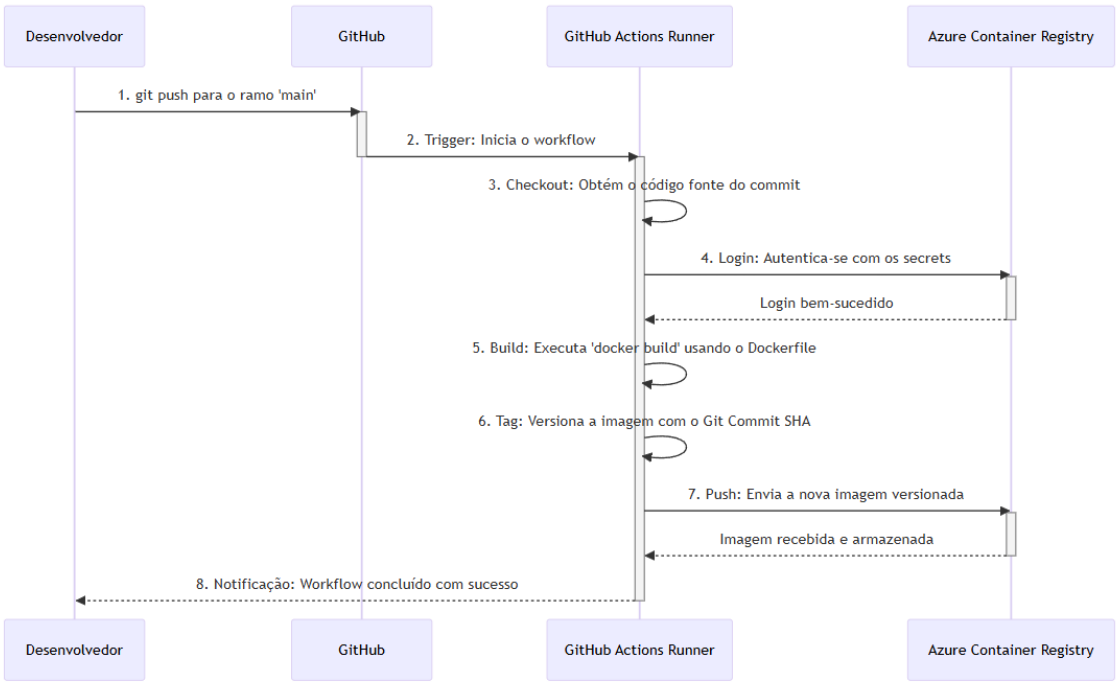


Figura 7 - Diagrama de sequência do *workflow* no GitHub Actions

O resultado final desta *pipeline* de CI/CD é um artefacto de software versionado, testado (implicitamente, pelo sucesso da construção) e pronto para ser implementado. A promoção de uma nova versão para execução no *pipeline* do ADF torna-se um processo controlado, onde o operador simplesmente atualiza o parâmetro image_tag para a nova versão gerada automaticamente. Esta abordagem separa o desenvolvimento do código da sua implementação.

A execução é gerida de forma "*serverless*" pelo Azure Container Instances (ACI), que aloca os recursos de computação (CPU e RAM) necessários para cada processo sob demanda.

4.2.4. Camada de apresentação (BI)

O Microsoft Power BI conecta-se ao Data Warehouse em modo de importação, carregando o esquema em floco de neve para o seu motor em memória (VertiPaq). As relações entre as tabelas de factos e dimensões são criadas no Power BI, permitindo a construção de relatórios interativos e a análise *drill-down* dos dados.

4.3. Decomposição da arquitetura com o modelo C4

Para documentar e comunicar a arquitetura da solução PM4IPLeiria de forma clara e consistente, foi adotado o Modelo C4. Este modelo, criado por Simon Brown, propõe a visualização de uma arquitetura de software em quatro níveis de abstração progressivamente mais detalhados: Contexto, *Containers*, Componentes e Código. Esta abordagem permite que diferentes audiências compreendam a arquitetura ao nível de detalhe que lhes é relevante [17]. As secções seguintes apresentam os três primeiros níveis do modelo aplicados a este projeto.

4.3.1. Nível 1: diagrama de contexto de sistema

O primeiro nível, o Diagrama de Contexto, oferece a visão mais macro da solução. Ele posiciona o sistema PM4IPLeiria no centro e ilustra as suas interações de alto nível com os seus utilizadores (atores) e com os sistemas externos dos quais depende ou com os quais se integra.

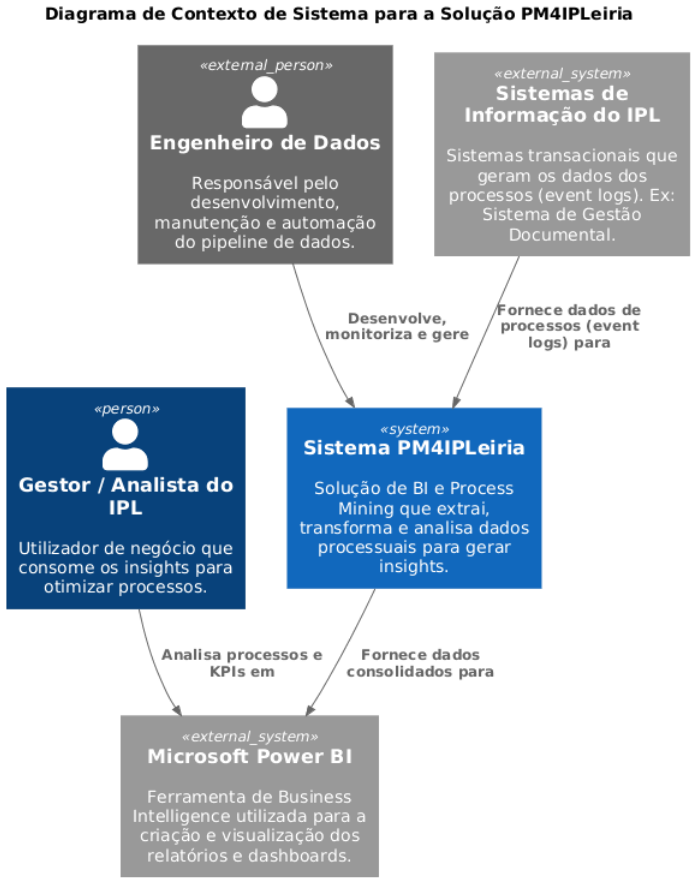


Figura 8 - Diagrama de contexto de sistema para a solução PM4IPLeiria (Nível 1)

Como ilustrado na Figura 8, o sistema interage com dois perfis de utilizador e depende de dois sistemas externos principais. O Gestor/Analista consome os *insights* através do Microsoft Power BI, enquanto o Engenheiro de Dados é responsável pelo desenvolvimento e manutenção do sistema, que, por sua vez, extrai os dados brutos dos Sistemas de Informação do IPL.

4.3.2. Nível 2: diagrama de *c*ontainers

No segundo nível, o Diagrama de Containers aprofunda a análise da macroarquitetura, decompondo o sistema PM4IPLeiria nas suas principais unidades executáveis e de armazenamento. No contexto do Modelo C4, um "contentor" representa uma unidade implementável, como uma aplicação *standalone* ou um repositório de dados.

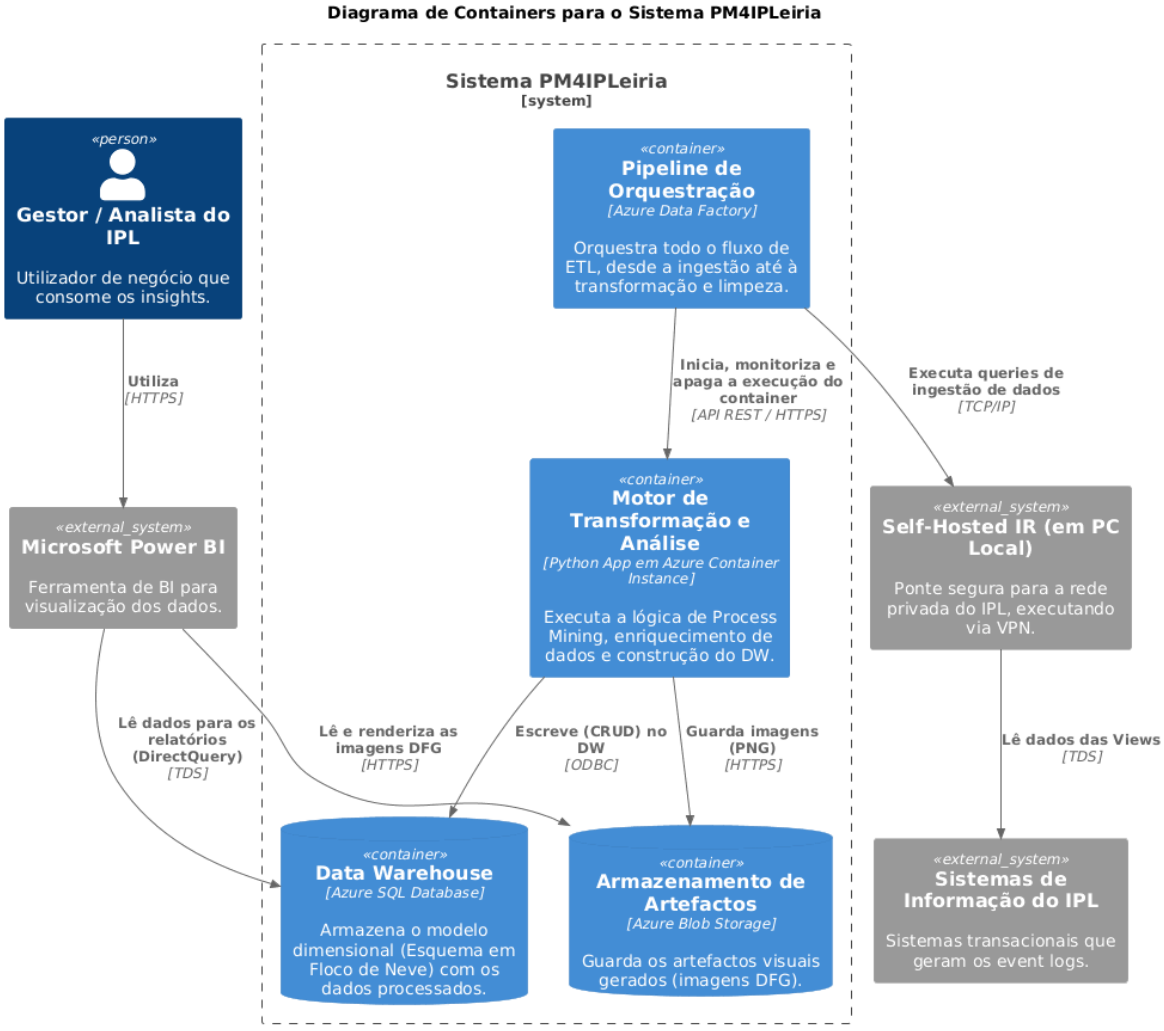


Figura 9 - Diagrama de contentores para o sistema PM4IPLeiria (Nível 2)

A Figura 9, segmenta a solução nos seus quatro contentores tecnológicos principais:

- *Pipeline* de orquestração (Azure Data Factory): O componente central de orquestração, responsável pela gestão, agendamento e monitorização de todo o fluxo de extração, transformação e carga (ETL).

- Motor de transformação e análise (aplicação Python em ACI): Um contentor Docker que encapsula a lógica de processamento de dados e a aplicação dos algoritmos de *Process Mining*, executado num ambiente *serverless* através do Azure Container Instances.

- Data Warehouse (Azure SQL Database): O repositório de dados central e otimizado, que armazena o modelo dimensional e serve como a fonte única da verdade (*single source of truth*) para a análise.

- Armazenamento de artefactos (Azure Blob Storage): Um repositório para o armazenamento de artefactos não estruturados, como os grafos de sequência (*Directly-Follows Graphs*) gerados durante o processamento.

As interações representadas no diagrama ilustram o fluxo de dados e controlo, desde a extração de dados *on-premises* (mediada pelo *Self-Hosted Integration Runtime*) até à persistência dos dados processados no *Data Warehouse*, culminando na sua disponibilização para as ferramentas de BI.

4.3.3. Nível 3: diagrama de componentes

O Diagrama de Componentes (Nível 3) foca-se na arquitetura interna do "Motor de Transformação e Análise", o principal artefacto de software desenvolvido neste projeto. A sua estrutura é decomposta em módulos Python, aderindo a princípios fundamentais de *design* de software como a separação de responsabilidades (*separation of concerns*) e o Princípio da Responsabilidade Única (*Single Responsibility Principle*).

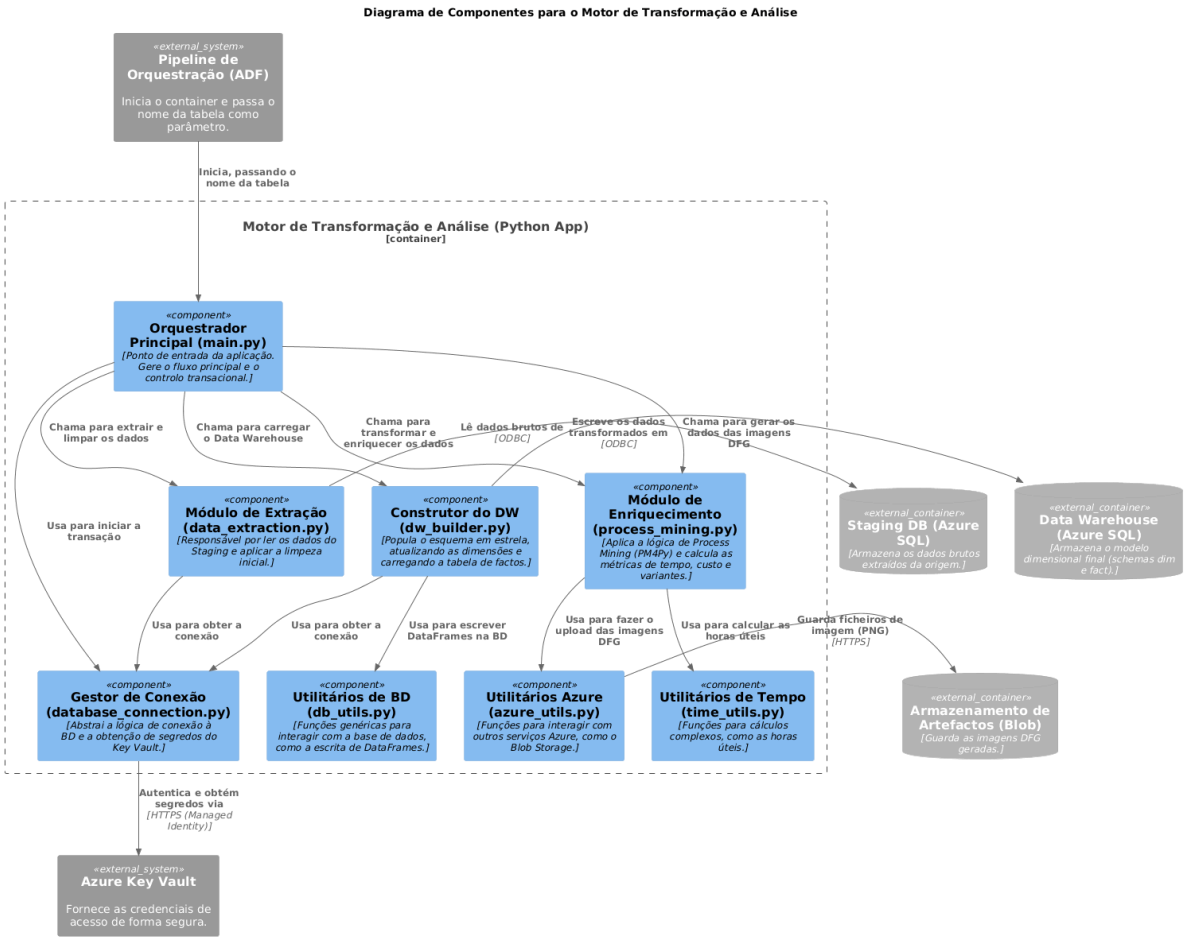


Figura 10 - Diagrama de componentes para o motor de transformação e análise (Nível 3)

O diagrama (Figura 10) evidencia uma arquitetura modular, onde um orquestrador (main.py) gere o fluxo de execução entre os componentes responsáveis pela extração de dados (data_extraction.py), pela transformação e aplicação de algoritmos (process_mining.py), e pela carga no Data Warehouse (dw_builder.py). Estes componentes, por sua vez, dependem de um conjunto de módulos utilitários que abstraem a lógica de acesso a dados e a interação com os serviços Azure, promovendo a reutilização de código e a manutenibilidade da solução.

4.4. Modelação do Data Warehouse

Conforme introduzido no capítulo anterior, a solução de armazenamento é um Data Warehouse. Esta secção detalha o seu modelo físico, justificando as decisões de *design* com base nas metodologias *standard* da indústria de *Business Intelligence*.

4.4.1. Justificação do modelo dimensional (OLAP vs. OLTP)

A primeira decisão fundamental na construção de um Data Warehouse é a escolha do paradigma de modelação. Enquanto os sistemas operacionais do dia-a-dia utilizam modelos OLTP (*Online Transaction Processing*), um Data Warehouse é desenhado para OLAP (*Online Analytical Processing*). Um modelo OLTP, tipicamente em 3ª Forma Normal (3NF), é otimizado para escritas transacionais rápidas e para eliminar a redundância de dados, resultando numa complexa rede de tabelas interligadas. Esta estrutura é ineficiente para as *queries* de agregação massivas que são típicas da análise de negócio [18].

Por este motivo, foi adotada a Modelação Dimensional, popularizada por Ralph Kimball, que é o padrão da indústria para Data Warehouses. Este modelo organiza os dados de forma intuitiva para os analistas, separando-os em tabelas de factos, que contêm as medições numéricas de um processo, e tabelas de dimensão, que fornecem o contexto descritivo. O resultado é um modelo otimizado para performance de leitura, simplicidade de consulta e facilidade de compreensão, que serve de base para ferramentas de BI como o Power BI [14].

4.4.2. O Esquema em floco de neve

Dentro da modelação dimensional, o esquema em estrela (*Star Schema*) é o *design* mais comum, privilegiando a simplicidade ao desnormalizar todos os atributos descritivos para as dimensões que se ligam diretamente à tabela de factos. No entanto, a metodologia de Kimball prevê exceções onde a normalização de uma dimensão (criando um Esquema em floco de neve) é justificada [19].

O nosso modelo implementa precisamente esta exceção para a dimensão de variantes de processo, e para a dimensão das imagens resultantes do processo DFG. A decisão de criar uma DIM_VARIANTE separada, ligada à DIM_CASO, e ANALISE_PROCESSO_DFG_IMAGENS ligada à DIM_PROCESSO foi uma otimização deliberada pelas seguintes razões:

- Atributo de alta cardinalidade e grande volume: A *string* que descreve uma variante de processo (VarianteString) pode ser extremamente longa e o número de variantes únicas pode ser muito elevado. Manter este texto longo e repetido para cada caso na DIM_CASO (a abordagem de esquema em estrela) resultaria numa tabela de dimensão massiva, ineficiente em armazenamento e lenta para consultar.

- Criação de uma entidade analítica: Ao isolar a variante e a informação relativa às imagens na sua própria dimensão, elevamo-la de um simples atributo a uma entidade analítica. Isto permitiu-nos enriquecê-las com os seus próprios metadados, como NumeroDeAtividadesUnicas e FlagRework (no caso da variante), e TipoDFG e DataGeracao (no caso das imagens relativas aos processos) desbloqueando novas vias de análise sobre a complexidade e eficiência dos próprios caminhos do processo.

Esta abordagem de "*snowflaking*" troca um *JOIN* adicional (uma operação de baixo custo para motores de BI modernos) por ganhos significativos em performance, armazenamento e capacidade analítica [14].

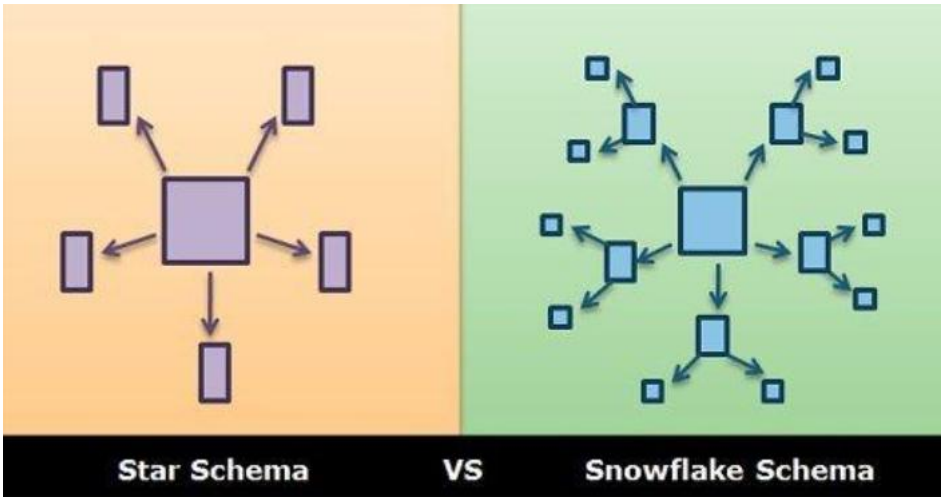


Figura 11 – Comparação visual entre esquema em estrela e esquema em floco de neve (https://techdifferences.com/difference-between-star-and-snowflake-schema.html)

4.4.3. O modelo físico e as suas componentes

O modelo de dados final é composto por uma tabela de factos central e um conjunto de dimensões, incluindo as dimensões que tornam o esquema num esquema floco de neve, a dimensão DIM_VARIANTE, como se pode observar na Figura 12.

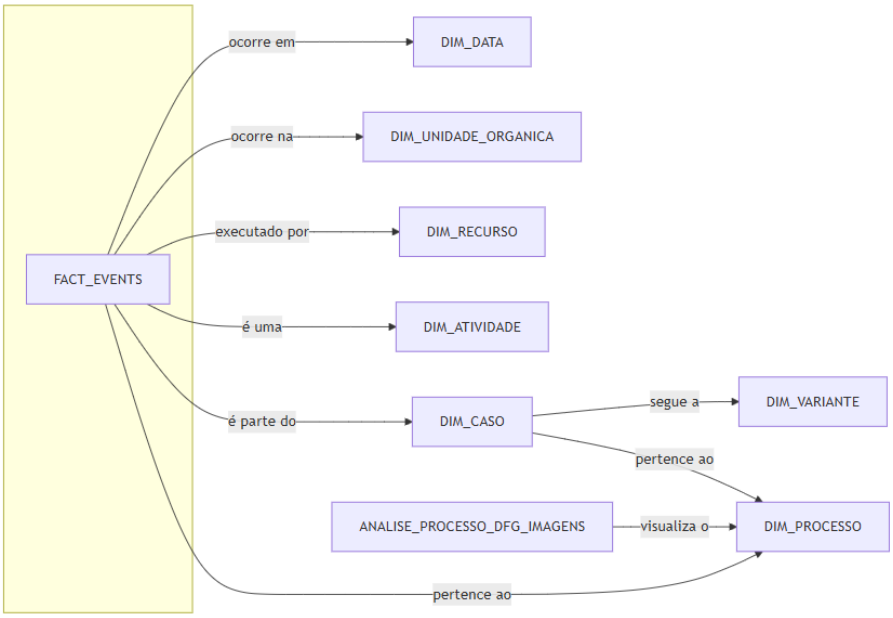


Figura 12 - Diagrama do esquema em floco de neve

- **Tabela de Factos (FACT_EVENTS) [**Anexo B**]:** O núcleo do modelo. A sua granularidade é de um evento de processo. Contém as chaves estrangeiras para todas as dimensões e as métricas numéricas aditivas (factos) que ocorrem a cada evento, como os custos e as durações.
- **Tabelas de Dimensão**
  - **DIM_PROCESSO [**Anexo C**]:** Tabela mestra dos processos analisados. Contém a chave substituta ProcessoID e os atributos descritivos de cada processo. A inclusão da ProcessoID diretamente na tabela de factos, para além da sua presença na DIM_CASO, é uma técnica de otimização intencional. Esta redundância cria um "atalho" de JOIN que acelera significativamente as *queries* agregadas ao nível do processo, um padrão por vezes referido como uma dimensão "outrigger" que é desnormalizada para a tabela de factos para melhorar a performance [19].
  - **DIM_CASO [**Anexo D**]:** Descreve cada instância de processo. É a dimensão mais rica, contendo a chave estrangeira para a `DIM_VARIANTE`, bem como métricas agregadas que descrevem o caso como um todo (tal como, `CustoTotalCaso`, `DuracaoCasoDias`). O `NomeCasoOrigem` (o `id` original) poderia ser considerado uma “dimensão degenerada” se não

tivéssemos outros atributos de caso para justificar uma tabela de dimensão. No entanto, como temos métricas agregadas ricas, a criação de uma `DIM_CASO` completa é a abordagem correta.

- **DIM_VARIANTE [**Anexo E**]:** A dimensão floco de neve. Contém uma lista única de todas as variantes de processo, enriquecida com metadados como o número de atividades únicas e a existência de *rework*. A unicidade é garantida por um *hash* (`VarianteHash`) da *string* da variante, uma técnica necessária para indexar eficientemente a coluna de texto longo `VarianteString`.
- **DIM_ATIVIDADE [**Anexo F**], DIM_RECURSO [**Anexo G**], DIM_UNIDADE_ORGANICA [**Anexo H**]:** Dimensões simples que contêm as listas únicas e normalizadas de cada uma destas entidades de negócio.
- **DIM_DATA [**Anexo I**]:** Uma dimensão de calendário, gerada dinamicamente, com base nos dias dos eventos, e que permite análises temporais ricas (por ano, mês, dia da semana), um pilar fundamental de qualquer Data Warehouse.
- **ANALISE_PROCESSO_DFG_IMAGENS [**Anexo J**]:** Esta é uma tabela de metadados que armazena os artefactos visuais gerados pelo *pipeline*. Em vez de guardar as imagens diretamente na base de dados, o que seria ineficiente, a tabela armazena o URL para cada imagem DFG (Frequência e Performance) guardada no Azure Blob Storage, ligando cada visualização ao processo que ela representa.

4.4.4. Validação de performance: justificação do esquema em floco de neve

A decisão de implementar um Esquema em floco de neve, normalizando a dimensão de variantes, foi uma escolha de *design* deliberada baseada em princípios teóricos de modelação dimensional. Para validar empiricamente esta decisão, foi conduzido um *benchmark* de performance comparando o modelo implementado com uma alternativa em esquema em estrela puro.

Para a comparação, foi criada uma tabela desnormalizada, DIM_CASO_STAR, que representa a estrutura que a dimensão de caso teria num esquema em estrela. Esta tabela foi

populada juntando os atributos da DIM_VARIANTE diretamente na DIM_CASO, replicando assim a redundância de dados inerente a essa abordagem.

Foi então executada uma query analítica representativa em ambos os modelos. A query foi desenhada para calcular a duração média dos casos, agrupando-os por variantes que contêm ou não *rework* (FlagRework), para um processo de alto volume de dados (com aproximadamente 4.000 casos e 13 variantes únicas). A performance de cada *query* foi medida no SQL Server Management Studio, analisando o tempo de CPU, o tempo decorrido e, crucialmente, o número de leituras lógicas (*logical reads*), que é um indicador chave da eficiência de uma consulta [20]. Para garantir uma comparação justa, a *cache* da base de dados foi limpa (com o comando DBCC DROPCLEANBUFFERS) antes da execução de cada teste.

Os resultados do *benchmark* confirmaram a hipótese de que, para esta carga de trabalho específica, o modelo em floco de neve é significativamente mais performante, como podemos observar na Tabela 2.

Tabela 2 - Resultados do *Benchmark* de performance esquema floco de neve *versus* esquema estrela

Métrica	**Modelo Floco de Neve**	**Modelo Estrela**	**Variação**
Leituras Lógicas (DIM_CASO)	385	N/A	-
Leituras Lógicas (DIM_VARIANTE)	15	N/A	-
Leituras Lógicas (DIM_CASO_STAR)	N/A	11240	-
Total de Leituras Lógicas	400	11240	~28 vezes mais eficiente (Floco de Neve)
Tempo de CPU (ms)	24ms	312ms	~13 vezes mais rápido (Floco de Neve)

Os dados demonstram que, embora o modelo em floco de neve exija um JOIN adicional, a redução massiva no tamanho da DIM_CASO (ao remover a VarianteString redundante) resulta numa diminuição drástica do I/O (Input/Output – de Leituras Lógicas) necessário para satisfazer a query. Desta forma, o motor da base de dados processa as tabelas DIM_VARIANTE e DIM_CASO, mais compactas, com uma rapidez consideravelmente maior do que necessitaria para percorrer a totalidade da DIM_CASO_STAR.

4.5. Avaliação de plataformas alternativas e seleção da abordagem final

Antes de se proceder ao desenvolvimento do pipeline de ETL customizado, foi realizada uma prova de conceito (*Proof of Concept* - PoC) para avaliar as capacidades nativas de *Process Mining* da plataforma Microsoft Power Platform. O objetivo era determinar se as ferramentas out-of-the-box seriam suficientes para atingir os objetivos do projeto.

A solução testada baseou-se na integração entre o Power BI e o Power Automate. A funcionalidade de *Process Mining* da Microsoft, integrada no ecossistema Microsoft Fabric, permite aos utilizadores carregar event logs e gerar automaticamente um mapa de processo interativo dentro de um relatório Power BI. O Power Automate é utilizado como o motor de análise que processa os dados e gera o modelo visual.

Durante este teste, foram seguidos os passos para carregar um dos conjuntos de dados de um processo do Politécnico de Leiria e mapear as colunas id, Etapa e Data de receção para os campos de *Case ID*, *Activity Name* e *Timestamp*, respetivamente. A ferramenta foi capaz de gerar um mapa de processo básico com sucesso.

No entanto, esta abordagem foi descartada por duas razões fundamentais:

1. Requisitos de licenciamento: A utilização plena das capacidades de *Process Mining* da Microsoft está associada ao ecossistema Microsoft Fabric e requer a alocação de uma "Fabric Capacity". Este é um recurso premium que implica custos significativos, tornando-o inviável no contexto de uma subscrição *Azure for Students* e para um projeto de tese focado em demonstrar valor através de soluções eficientes.

2. Incompatibilidade da transição de processos: Durante a prova de conceito, foram encontrados problemas de fiabilidade na transição de dados entre as plataformas Power BI e Power Automate. Mesmo seguindo os procedimentos recomendados, a

sincronização dos dados de processo por vezes falhava, resultando numa experiência de utilizador inconsistente e pondo em causa a robustez da solução para uma implementação automatizada.

5. Implementação do Data Warehouse

Este capítulo descreve o percurso prático de construção da solução PM4IPLeiria, detalhando as etapas técnicas e as decisões de implementação tomadas. A narrativa segue a estrutura da metodologia CRISP-DM, apresentada no capítulo anterior, para demonstrar como os requisitos analíticos foram traduzidos numa arquitetura de dados funcional e automatizada. As secções seguintes irão abordar a exploração inicial dos dados, a preparação do ambiente de Data Warehouse na *cloud*, o desenvolvimento do *pipeline* de ETL e a implementação dos algoritmos de *Process Mining*.

5.1. Compreensão e análise exploratória dos dados

Conforme os princípios do CRISP-DM, a primeira etapa prática do projeto consistiu na imersão e compreensão das fontes de dados. O ponto de partida foi o acesso, através de uma ligação VPN segura, à base de dados principal da instituição, que serve de repositório para os sistemas de informação do Politécnico de Leiria.

Por razões de segurança e privacidade, em conformidade com o Regulamento Geral sobre a Proteção de Dados (RGPD), os dados foram previamente anonimizados pela instituição. O acesso foi concedido não às tabelas operacionais diretas, mas a um conjunto de Vistas SQL (Views), localizadas no esquema dbo. Esta abordagem garantiu que o projeto pudesse operar sobre dados realistas sem comprometer a confidencialidade de informações sensíveis.

Procedeu-se então a uma análise exploratória para identificar as vistas relevantes para a análise de processos. Observou-se a existência de múltiplas vistas, cada uma correspondendo a um processo de negócio distinto, como por exemplo:

- VM_BOLSA_INVESTIGACAO
- VM_PEDIDO_JUSTIFICACAO_FALTAS
- VM_PAP_SC
- VM_SUBMISSAO_TESE_MESTRADO
- (entre outras)

A análise da estrutura destas vistas permitiu identificar um padrão comum e os campos essenciais para a aplicação de técnicas de *Process Mining*, que constituem o *event log* de cada processo:

- **id**: Um identificador único global (GUID) que correlaciona todos os eventos pertencentes a um mesmo caso. Foi identificado como a Chave do Caso (*Case ID*).
- **Etapa**: Uma descrição textual da atividade realizada. Foi identificada como a Atividade (*Activity*).
- **Data de receção**, **Data Leitura**, **Data Envio**: *Timestamps* que marcam o ciclo de vida de cada evento, servindo como base para o cálculo de métricas de tempo. A Data de receção foi definida como o *Timestamp* do Evento.
- **Despachado Por**: O recurso (utilizador ou sistema) responsável pela execução da atividade. Foi identificado como o Recurso (*Resource*).
- **Unidade Orgânica**: O departamento ou escola onde o evento ocorreu, servindo como uma dimensão de negócio crucial.

Esta fase de compreensão dos dados foi fundamental para validar a viabilidade do projeto e para informar o desenho do modelo de dados do Data Warehouse, garantindo que a estrutura final seria capaz de responder aos requisitos analíticos definidos.

5.2. Preparação do ambiente e modelação do Data Warehouse

Com um entendimento claro dos dados de origem, a fase seguinte consistiu na preparação do ambiente de destino e na modelação do Data Warehouse.

A decisão de construir um Data Warehouse centralizado foi motivada pela necessidade de ter uma "fonte única da verdade" otimizada para análise, separada dos sistemas transacionais de origem, conforme as melhores práticas de Business Intelligence [14]. Para a infraestrutura, foi escolhida a plataforma *cloud* Microsoft Azure. Esta escolha foi fundamentada por dois fatores principais: a disponibilidade de uma subscrição Azure *for Students*, que permitiu o desenvolvimento do projeto sem custos de infraestrutura, e a vasta adoção da Azure no mercado empresarial, o que confere ao projeto uma relevância e aplicabilidade alinhadas com as tecnologias mais utilizadas na indústria.

Para o desenvolvimento de scripts SQL foi escolhido o software SSMS (SQL Server Management Studio), embora a carga de dados seja totalmente automatizada pelo *pipeline* Python, o SSMS foi uma ferramenta indispensável durante todo o ciclo de desenvolvimento e validação. Foi utilizado para a criação e gestão inicial do esquema do Data Warehouse (tabelas, chaves, *constraints*), para a execução de *queries* de verificação para validar a integridade dos dados após cada execução do *pipeline*, e para a *debugging* de problemas de dados diretamente na base de dados.

O primeiro passo no provisionamento da infraestrutura foi a criação de um *Resource Group* (Grupo de Recursos) dedicado, denominado ipl-warehouse. Esta prática é fundamental para a gestão e organização de projetos na Azure, pois agrupa todos os recursos relacionados numa única unidade lógica. Isto simplifica a gestão de custos, a monitorização e, crucialmente, a gestão de permissões (IAM - *Identity and Access Management*), permitindo que futuras transferências de propriedade para a instituição sejam feitas de forma centralizada e segura.

Com o grupo de trabalho definido, os pilares do nosso Data Warehouse foram provisionados:

1. **Azure SQL Server:** Na Azure, a criação de uma base de dados SQL requer a existência prévia de um servidor lógico que a irá alojar (como é possível observar na Figura 13). Como tal, foi criado um recurso Azure SQL Server, que atua como o ponto de gestão central para um grupo de bases de dados, definindo a região, as credenciais de login e as regras de firewall.

2. **Azure SQL Database:** Associada ao servidor criado, foi provisionada a base de dados relacional em si, onde o nosso Data Warehouse foi implementado. Foi configurada com um modelo de computação *serverless* para otimizar os custos, adaptando os recursos dinamicamente à carga de trabalho.

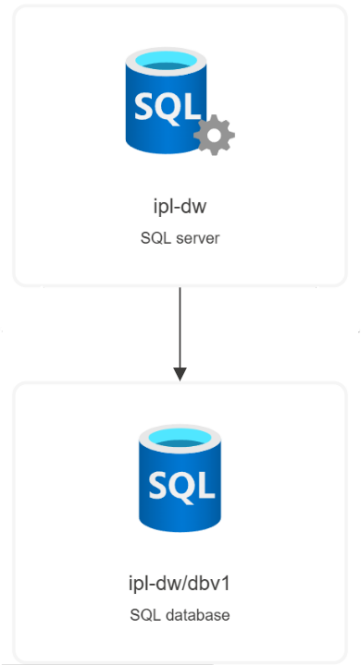


Figura 13 - Diagrama de serviços pertencentes ao Azure SQL Server

Após o provisionamento da infraestrutura, procedeu-se à criação da estrutura lógica do Data Warehouse na base de dados. Foram criados esquemas dedicados para organizar as tabelas e separar as diferentes camadas de dados, uma prática que melhora a organização e a segurança [21]:

- **staging:** Esquema destinado a receber a cópia inicial e inalterada dos dados provenientes das vistas de origem.
- **dim:** Esquema para alojar todas as tabelas de dimensão.
- **fact:** Esquema para alojar a tabela de factos.
- **etl:** Esquema para tabelas de controlo do processo de ETL, como a tabela de "*watermarking*".

5.3. Implementação do esquema do Data Warehouse

Após a definição do modelo dimensional no Capítulo 4, o passo seguinte consistiu na sua implementação física. Utilizando o SQL Server Management Studio (SSMS), foi criada a estrutura de tabelas e esquemas na base de dados Azure SQL provisionada, estabelecendo a fundação que iria receber os dados processados. Esta etapa é crítica, pois a estrutura correta

das tabelas, com as suas chaves, tipos de dados e restrições, é o que garante a integridade e a performance do Data Warehouse.

Conforme o *design*, foram criados os esquemas dim e fact para alojar as tabelas de dimensão e de factos, respetivamente. Adicionalmente, foi criado um esquema etl para conter tabelas de controlo, essenciais para a orquestração do *pipeline*.

5.3.1. Tabela de controlo de carga (*Watermarking*)

Para suportar uma futura implementação de cargas incrementais eficientes no Azure Data Factory, foi criada a tabela LOG_CARGA_INCREMENTAL (Tabela 3). O objetivo desta tabela é registar o progresso da ingestão de dados para cada tabela de origem. Ela armazena uma "marca de água" (*watermark*), que corresponde tipicamente ao *timestamp* ou ID do último registo lido. Nas execuções subsequentes, o *pipeline* de ingestão pode consultar esta tabela para saber de que ponto deve recomeçar a leitura, processando apenas os dados novos. Esta é uma prática *standard* em *pipelines* de ETL para otimizar a performance e reduzir a carga nos sistemas de origem.

Tabela 3 - Tabela LOG_CARGA_INCREMENTAL

Atributos	Propriedades
IDLog	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
NomeTabelaStaging	NVARCHAR(255) NOT NULL UNIQUE
UltimoValorProcessado	DATETIME

5.3.2. Tabelas de dimensão

As tabelas de dimensão foram criadas no esquema dim. Todas as dimensões seguem a boa prática de utilizar uma chave substituta (*surrogate key*) — uma coluna de identidade inteira (como, ProcessoID INT IDENTITY(1,1)) — como chave primária. O uso de chaves substitutas, em detrimento das chaves de negócio naturais (em texto), melhora drasticamente a performance das operações de JOIN e isola o Data Warehouse de alterações nos sistemas de origem [14].

As dimensões DIM_PROCESSO, DIM_ATIVIDADE, DIM_RECURSO, DIM_UNIDADE_ORGANICA são dimensões que contêm as listas mestras de cada uma das respetivas entidades de negócio. Foi aplicada uma restrição UNIQUE nas colunas que contêm a chave de negócio (como, NomeAtividade) para garantir que não existem duplicados. Para colunas de texto com potencial para exceder os limites de indexação, como NomeUnidadeOrganica, foi utilizado o tipo de dados NVARCHAR(MAX).

Na dimensão DIM_VARIANTE, dada a natureza textual e o comprimento variável da string que define uma variante de processo, a restrição UNIQUE não pôde ser aplicada diretamente na coluna VarianteString. Para contornar esta limitação técnica, foi implementada uma solução robusta: a adição de uma coluna VarianteHash do tipo VARBINARY(32). O pipeline Python calcula um hash SHA-256 para cada VarianteString e é nesta coluna de hash, de tamanho fixo e otimizada para indexação, que a restrição UNIQUE é aplicada, garantindo a integridade da dimensão.

A dimensão DIM_CASO descreve cada instância de processo, contém chaves estrangeiras que a ligam à DIM_PROCESSO e à DIM_VARIANTE. Foi implementada uma restrição UNIQUE composta na combinação das colunas (NomeCasoOrigem, ProcessoID), refletindo a regra de negócio de que um identificador de caso só é único dentro do contexto do seu processo.

5.3.3. Tabela de factos

Finalmente, foi criada a tabela FACT_EVENTS. Esta tabela é o centro do esquema em estrela e foi desenhada para máxima eficiência de armazenamento e consulta. A sua estrutura é composta quase inteiramente por:

1. Chaves estrangeiras: Colunas de inteiros (ProcessoID, CasoID, AtividadeID, etc.) que se ligam a cada uma das tabelas de dimensão.

2. Factos: Colunas numéricas (FLOAT ou DATETIME) que contêm as medições de cada evento (por exemplo, DuracaoExecucaoHoras, CustoPorAtividade, TimestampInicio).

Esta estrutura altamente normalizada (no que toca à separação de factos e dimensões) é o que permite que as ferramentas de BI, como o Power BI, executem agregações e filtros sobre milhões de linhas de forma performática.

5.4. Implementação do *Pipeline* de ETL com Python

Com a arquitetura de dados definida e a abordagem tecnológica selecionada, a fase seguinte focou-se no desenvolvimento do *pipeline* de Extração, Transformação e Carga (ETL), que constitui o núcleo da solução. Esta implementação foi realizada integralmente em Python, aproveitando o seu ecossistema robusto para engenharia de dados.

5.4.1. Gestão de código e estrutura do projeto

Para garantir a manutenibilidade, a colaboração e o controlo de versões, o projeto foi desde o início gerido num repositório Git, alojado na plataforma GitHub. Esta prática permitiu um desenvolvimento iterativo e seguro, com um histórico completo de todas as alterações efetuadas.

O projeto Python foi estruturado de forma modular, separando as diferentes responsabilidades lógicas em ficheiros distintos (Figura 14). Esta separação é uma prática fundamental da engenharia de software que facilita a depuração, o teste e a futura expansão do código [22]. A estrutura principal do projeto é a seguinte:

- **main.py:** O orquestrador principal. É o ponto de entrada do *pipeline*, responsável por controlar o fluxo de execução, gerir a transação da base de dados e chamar os outros módulos na sequência correta.
- **database_connection.py:** Módulo centralizado para gerir a ligação à base de dados. Contém a lógica para ler as credenciais de forma segura e criar o motor de conexão SQLAlchemy.
- **data_extraction.py:** Responsável pela fase de Extração (E). Contém a função que se conecta à camada de *staging*, executa a *query* SQL para ler os dados de um processo específico e realiza a primeira camada de limpeza e normalização.
- **process_mining.py:** O coração da fase de Transformação (T). Este módulo recebe os dados brutos e aplica toda a lógica de negócio e de *Process Mining* para enriquecê-los, gerando o DataFrame desnormalizado (master_log_df) em memória.
- **dw_builder.py:** Responsável pela fase de Carga (L). Contém a lógica para pegar no DataFrame enriquecido e popular o esquema em estrela no Data Warehouse, atualizando as tabelas de dimensão e carregando a tabela de factos.

- **time_utils.py:** Um módulo utilitário dedicado aos cálculos de tempo complexos, nomeadamente a função que calcula as horas úteis entre dois *timestamps*, descontando fins de semana e pausas.
- **azure_utils.py:** Módulo utilitário para interagir com os serviços da Azure, como a função para fazer o upload das imagens DFG para o Blob Storage.
- **db_utils.py:** Módulo utilitário para operações de base de dados, como a função genérica para escrever DataFrames em tabelas.

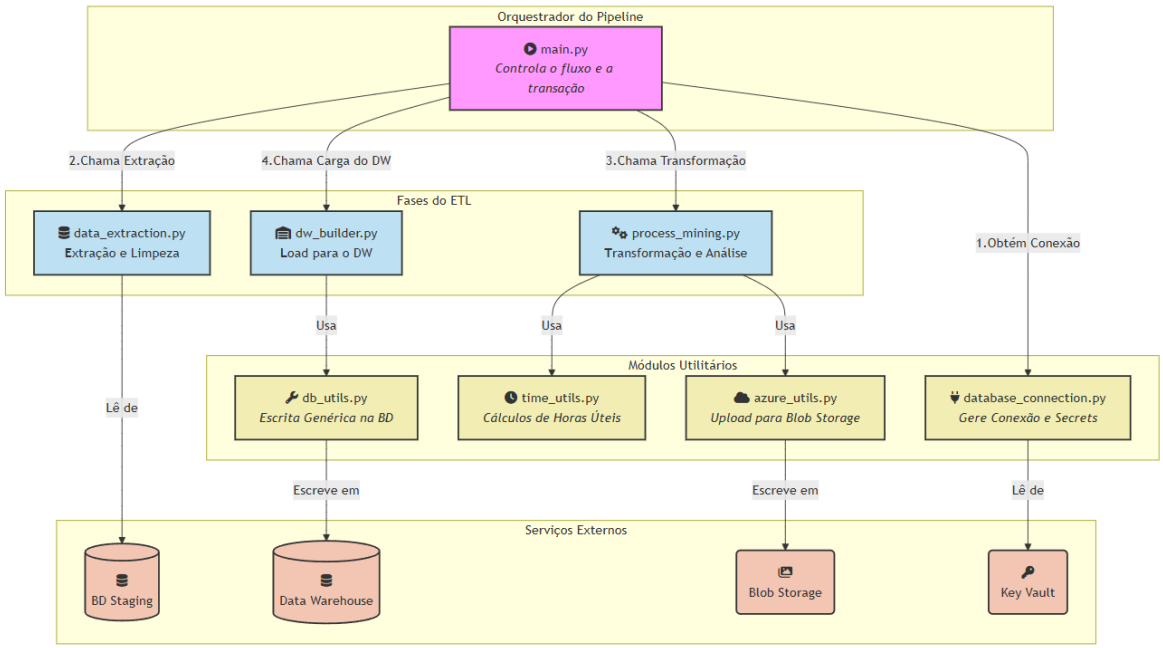


Figura 14 - Diagrama da arquitetura do projeto Python

5.4.2. Orquestração e controlo transacional

O ponto de entrada da aplicação (main.py) atua como o orquestrador de todo o *pipeline*. A sua responsabilidade mais crítica é garantir a atomicidade da operação de carga de dados. Para tal, foi implementado um bloco de transação único utilizando o padrão with engine.begin() as connection: da biblioteca SQLAlchemy. Este mecanismo assegura que todas as operações na base de dados — desde a atualização das dimensões até à inserção na tabela de factos — são tratadas como uma única unidade de trabalho. Se qualquer etapa falhar, a transação inteira é automaticamente revertida (ROLLBACK), garantindo que o Data Warehouse nunca fica num estado inconsistente. Se todas as etapas forem bem-

sucedidas, a transação é confirmada (COMMIT) no final, uma prática essencial para a integridade em sistemas de dados [23].

5.4.3. Extração e limpeza de dados

A primeira fase ativa do pipeline é a extração e limpeza. O processo inicia-se com a leitura dos dados das tabelas do esquema *staging*, que contêm a cópia dos dados de origem. Uma vez carregados para um DataFrame da biblioteca Pandas, é aplicada uma rigorosa política de qualidade de dados.

Foi implementada uma rotina de limpeza dinâmica que identifica automaticamente todas as colunas de texto livre. Para cada uma destas colunas, é aplicada uma sequência de normalização:

1. Remoção de espaços em branco no início e no fim (strip).
2. Substituição de sequências de múltiplos espaços por um único espaço, para corrigir inconsistências como 'validar informação'.
3. Unificação de todos os valores considerados "vazios" (strings vazias, *None*, NaN) para um *placeholder standard* e legível ("Sem informação").

Esta etapa é fundamental para garantir a consistência das chaves de negócio textuais (como nomes de atividades e recursos) antes de estas serem utilizadas para popular as tabelas de dimensão.

5.4.4. Transformação e enriquecimento analítico

Com os dados limpos, o *pipeline* avança para a fase de transformação, onde os dados brutos são convertidos num conjunto de dados ricos em métricas analíticas. Este processo ocorre na totalidade em memória, utilizando o poder do Pandas, e resulta num DataFrame desnormalizado (ao qual, foi intitulado master_log_df) que serve de base para a construção do Data Warehouse.

As principais transformações incluem:

- Cálculo de tempos em horas úteis: Para que as métricas de custo e duração fossem realistas, foi desenvolvida uma lógica para calcular o tempo decorrido em horas úteis,

considerando um horário de trabalho padrão (9h-17h com pausa para almoço das 13h às 14h) e excluindo fins de semana.

- Cálculo de custos: Fatores de custo pré-definidos (nomeadamente, custo por hora de execução, custo de RH) são aplicados às métricas de tempo para calcular os vários tipos de custo associados a cada evento.
- Agregação por caso: São calculadas métricas que descrevem o caso como um todo, como o custo total (cost_per_case_agg) e a duração total (case_duration_days_agg).
- Descoberta da variante de processo: Para cada caso, os seus eventos são ordenados cronologicamente e os nomes das atividades são concatenados para formar uma string que representa a "variante" ou o "caminho" exato que o processo seguiu.

5.4.5. Carga para o Data Warehouse dimensional

A fase final do pipeline é a carga do DataFrame enriquecido para o esquema em floco de neve na base de dados. Este processo é gerido de forma a garantir a eficiência e a integridade.

Para as dimensões simples e independentes (DIM_ATIVIDADE, DIM_RECURSO, DIM_UNIDADE_ORGANICA), foi implementada uma estratégia de carga incremental. O script compara os valores da carga atual com os que já existem na dimensão e insere apenas os que são novos (por exemplo, se uma atividade chamada "Aprovar Pedido" já existe na DIM_ATIVIDADE com AtividadeID = 10, ela nunca mais será tocada ou recriada. Isto garante que as chaves das dimensões são estáveis, o que é fundamental para a integridade do Data Warehouse.). Para a DIM_VARIANTE, uma lógica semelhante é aplicada, mas utilizando um hash SHA-256 da string da variante como chave de unicidade para contornar as limitações de indexação de texto longo.

Para as tabelas DIM_CASO e FACT_EVENTS, foi implementada uma estratégia de "delete-and-insert por processo". A cada execução, o script apaga todos os casos e eventos pertencentes ao processo em questão e insere a nova versão completa. Esta abordagem, embora mais intensiva, foi escolhida para garantir a máxima consistência das métricas agregadas, que podem mudar sempre que um novo evento é adicionado a um caso existente. Isto porque as métricas de *Process Mining* (duração dos casos, variantes, custos) são muito sensíveis. Se um novo evento for adicionado a um caso antigo, ou se uma data for corrigida,

a forma mais segura e robusta de garantir que todas as análises estão 100% corretas é reprocessar o processo inteiro. Tentar fazer um *MERGE* de eventos numa tabela de factos é extremamente complexo e propenso a erros. Ao apagar e recarregar os dados de um processo, garantimos que a "fotografia" desse processo no Data Warehouse é sempre perfeita e consistente com os dados mais recentes provenientes da tabela original.

Antes da carga final, as chaves de negócio em texto (como, case_id, activity) no DataFrame são substituídas pelas suas respetivas chaves substitutas inteiras (como, CasoID, AtividadeID), obtidas a partir das tabelas de dimensão.

5.4.6. Geração de artefactos visuais (*Process Mining*)

Paralelamente à construção do Data Warehouse, o *pipeline* aplica algoritmos de *Process Mining* da biblioteca PM4Py para gerar modelos visuais do processo. São criados *Directly-Follows Graphs* (DFG) de frequência e de performance. Em vez de serem guardados como ficheiros locais, estas imagens são renderizadas em memória, carregadas para o Azure Blob Storage, e os seus URLs são registados na tabela ANALISE_PROCESSO_DFG_IMAGENS, ligando cada artefacto visual ao seu respetivo processo.

5.4.7. Gestão de configuração e segurança

Uma das principais preocupações em qualquer *pipeline* de dados é a gestão segura de configurações, especialmente de *secrets* como credenciais de bases de dados e chaves de API. Seguiu-se a prática recomendada de separar a configuração do código, garantindo que nenhuma informação sensível é armazenada diretamente no repositório Git [13].

- **Desenvolvimento local:** Para facilitar o desenvolvimento local, foi utilizado um ficheiro .env para armazenar as variáveis de ambiente. A biblioteca python-dotenv é usada para carregar estas variáveis no início da execução, simulando o ambiente da cloud. Este ficheiro é explicitamente excluído do controlo de versões através do ficheiro .gitignore.
- **Ambiente de produção (Azure):** Na cloud, a gestão de secrets é delegada ao Azure Key Vault. Este serviço funciona como um cofre-forte centralizado e seguro. O *pipeline* do Azure Data Factory (ADF) injeta apenas uma variável de ambiente não-sensível no container: o nome do Key Vault (KEY_VAULT_NAME).

Para autenticação segura, o código Python, ao ser executado dentro do Azure Container Instance (ACI), utiliza a biblioteca azure-identity. A classe DefaultAzureCredential deteta automaticamente a Identidade Gerida atribuída ao container. Esta identidade tem permissões (IAM Role) para ler os *secrets* do Key Vault. O script autentica-se de forma segura, sem necessitar de qualquer *password* ou chave no código, e obtém as credenciais da base de dados e do *storage* em tempo de execução. Esta abordagem é o padrão da indústria para a autenticação segura entre serviços na *cloud*.

5.4.8. Tratamento de erros e *logging* transacional

Para garantir a robustez e a integridade do Data Warehouse, foi implementada uma estratégia rigorosa de tratamento de erros e *logging*.

A principal salvaguarda é a gestão transacional. Todas as operações de escrita na base de dados (atualização de dimensões, deleção e inserção de factos) são envolvidas num único bloco de transação engine.begin() no módulo main.py. Isto garante a atomicidade: ou todas as operações são concluídas com sucesso e a transação é confirmada (COMMIT), ou, se ocorrer um único erro em qualquer ponto, a transação inteira é automaticamente revertida (ROLLBACK). Esta prática impede que o Data Warehouse fique num estado inconsistente [23].

As funções nos módulos de nível inferior (dw_builder.py, azure_utils.py) foram desenhadas para não "engolir" exceções, utilizam blocos *try...except*, mas no final do bloco except, o erro é levantado novamente (*raise e*). Isto garante que qualquer falha, por mais pequena que seja, é propagada até ao main.py, que aciona o ROLLBACK da transação.

Para a depuração em ambiente de *cloud*, onde o acesso direto a um terminal não é prático, foi implementado um sistema de *logging* robusto com o módulo *logging* do Python. Em vez de simplesmente imprimir para a consola, o script está configurado para escrever todos os *logs* (incluindo *tracebacks* completos de erros) para um ficheiro de texto (pipeline.log). Este ficheiro é escrito num Azure File Share que é "montado" como um volume dentro do container. Desta forma, mesmo que o container falhe e seja destruído, os *logs* persistem, permitindo uma análise *post-mortem* detalhada do que correu mal.

5.4.9. Otimizações de performance na carga de dados

Para lidar com volumes de dados significativos (tabelas com dezenas de milhares de linhas), foram implementadas várias otimizações de performance, especialmente na interação com a base de dados.

Em vez de inserir ou atualizar dados linha a linha, o que seria extremamente ineficiente, o *pipeline* utiliza as capacidades de operações em massa do Pandas e do SQLAlchemy. A função pandas.to_sql com o parâmetro *chunksize* permite enviar os dados para a base de dados em lotes (*chunks*), otimizando a performance da rede e da transação.

Para a atualização das tabelas de dimensão, em vez de uma lógica de SELECT seguida de UPDATE ou INSERT controlada pelo Python, foi utilizada uma única e poderosa declaração MERGE do SQL. O *pipeline* carrega os dados da dimensão para uma tabela temporária e depois executa um MERGE, delegando a lógica de "UPDATE se existir, INSERT se não existir" para o motor da base de dados, que é a forma mais performática de executar esta operação [20].

6. Implementação do *Pipeline* de orquestração

Após a implementação da lógica de transformação em Python, o passo seguinte foi a sua orquestração. O objetivo era criar um *pipeline* automatizado capaz de executar o *script* Python para cada processo de negócio de forma fiável e agendada. A ferramenta escolhida para esta tarefa foi o Azure Data Factory (ADF), devido à sua natureza *serverless*, às suas capacidades de orquestração visual e à sua integração nativa com o ecossistema Azure.

Esta secção detalha o processo iterativo de desenvolvimento do *pipeline* no ADF, incluindo a configuração da infraestrutura de conectividade, a abordagem inicial, os desafios encontrados e a arquitetura final implementada.

6.1. Configuração da conectividade híbrida (*Integration Runtimes*)

Um dos principais desafios deste projeto era a necessidade de extrair dados de uma base de dados de origem que não está publicamente acessível na internet, mas sim numa rede privada do Politécnico de Leiria, acessível apenas através de uma ligação VPN. Para resolver este cenário de conectividade híbrida, foi necessário configurar os *Integration Runtimes* (IR) do Azure Data Factory (Figura 15).

O *Integration Runtime* é a infraestrutura de computação que o ADF utiliza para executar as suas atividades, como a cópia de dados. Foram configurados dois tipos distintos, *Self-Hosted Integration Runtime* (apelidado de integrationRuntimeIPL) e Azure Integration Runtime (apelidado de AutoResolveIntegrationRuntime).

Para aceder à base de dados de origem, foi instalado um *Self-Hosted IR* numa máquina local. Este software atua como uma ponte segura: a máquina permanece ligada à VPN do Politécnico de Leiria, permitindo que o SHIR receba instruções do ADF (que está na *cloud*) e as execute na rede privada, extraindo os dados de forma segura sem nunca expor a base de dados de origem à internet.

O Azure Integration Runtime, é o *runtime* padrão, gerido pela Microsoft e *serverless*, que opera na totalidade dentro da *cloud* Azure. Foi utilizado para todas as operações que ocorrem entre serviços Azure, como a escrita dos dados na base de dados do Data Warehouse (Azure SQL DB). A sua configuração é automática e utiliza a Identidade Gerida (*Managed*

Identities) do Data Factory para se autenticar de forma segura nos outros serviços Azure, não sendo necessárias configurações de permissões complexas para este fim.

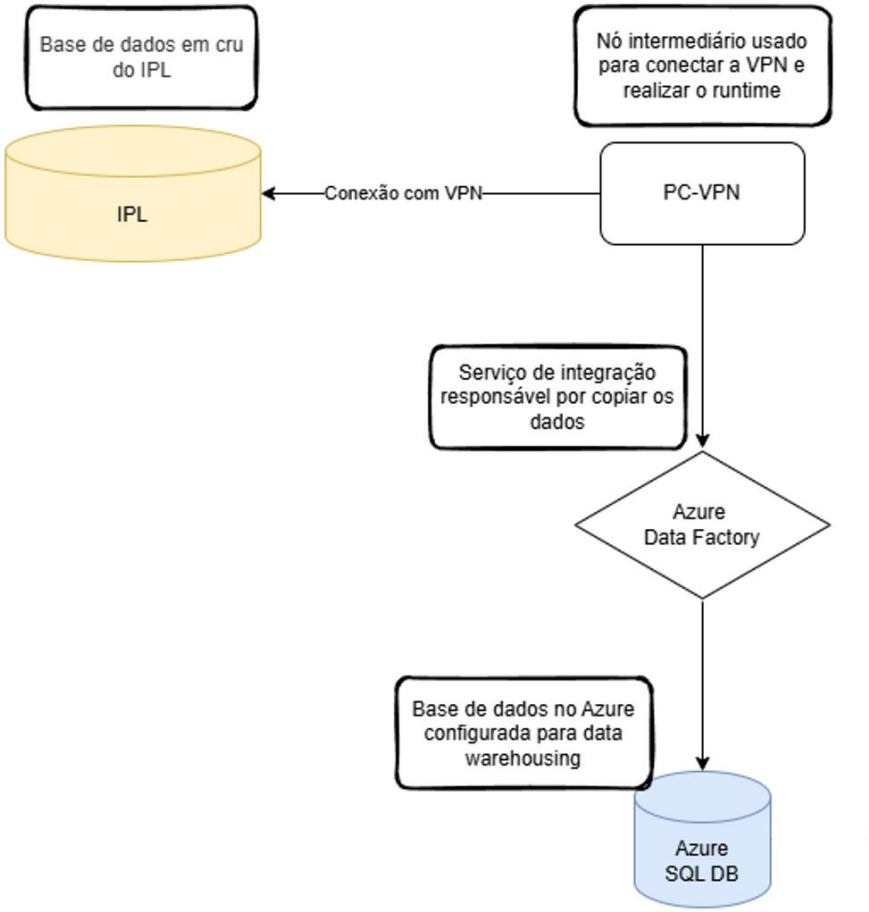


Figura 15 - Diagrama representativo da ligação entre ADF, SHIR e a BD de origem via VPN

6.2. Abordagem inicial e desafios: atividade *custom* com Azure Batch

A abordagem inicial e, teoricamente, mais direta para executar um *container* Docker a partir do ADF é através da utilização de uma Atividade *Custom* ligada a um *pool* do Azure Batch. Esta arquitetura foi implementada e testada, consistindo nos seguintes componentes:

- Um Azure Container Registry (ACR) para armazenar a imagem Docker da aplicação.
- Uma Identidade Gerida para permitir a autenticação segura entre os serviços.
- Um Pool do Azure Batch configurado com nós de computação baseados em Linux e com suporte para Docker, com a imagem da aplicação pré-carregada para otimizar a performance.

- Um *pipeline* no ADF com uma Atividade *Custom* configurada para enviar uma tarefa para o *pool* do Batch.

No entanto, durante a fase de testes, esta abordagem revelou-se consistentemente pouco fiável. O *pipeline* falhava frequentemente com erros genéricos, como *BadRequest*, ou com erros específicos do *Batch*, como *ContainerTaskSettingsNotFound*. A investigação em fóruns da comunidade e na documentação indicou que a interface gráfica da Atividade *Custom* no ADF tem um histórico de instabilidade e de não gerar corretamente a configuração JSON necessária para a execução de *containers*. Mesmo após a edição manual do código JSON do *pipeline*, os problemas de comunicação persistiram.

Face a esta instabilidade, e para garantir a robustez da solução final, foi decidido abandonar a utilização do Azure Batch e da Atividade *Custom* (Figura 16), optando por uma arquitetura mais moderna e fiável.

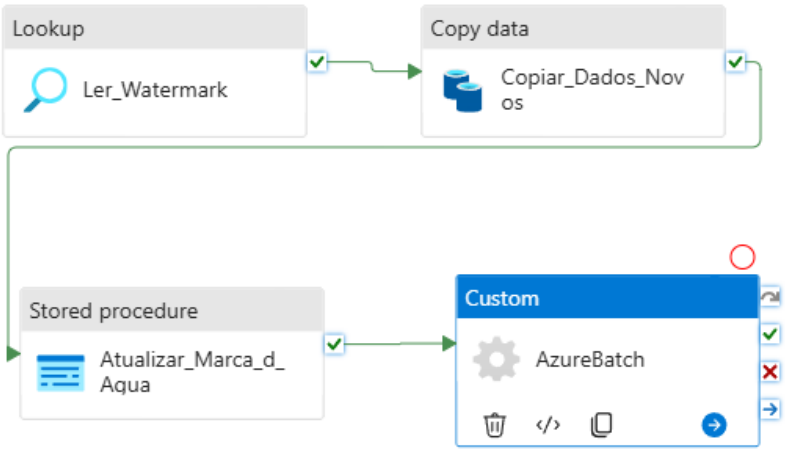


Figura 16 - Fluxo da abordagem inicial para o *pipeline*

6.3. Arquitetura Final

A solução final adotada baseia-se na orquestração de Azure Container Instances (ACI), um serviço *serverless* que executa *containers* sob demanda sem a necessidade de gerir máquinas virtuais ou *pools*. Para contornar a limitação do ADF que impede a utilização de atividades de controlo (no caso da atividade "*Until*") dentro de um *loop ForEach,* como demonstra a Figura 17, foi implementado o padrão de *design* "Pai-Filho".

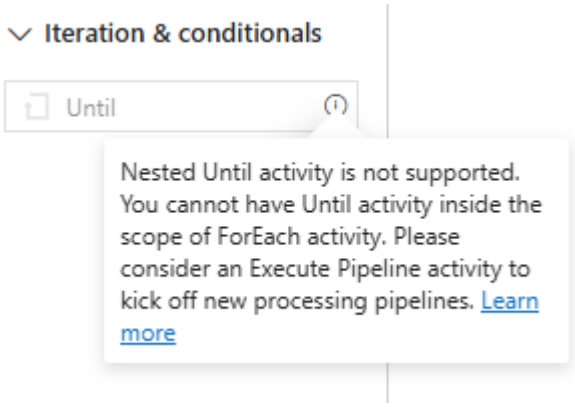


Figura 17 - Mensagem que indica a impossibilidade de adicionar a atividade *'Until'* no interior da atividade *'ForEach'*

O *pipeline* "Pai" (PL_Run_Process_Mining_IPL) atua como o orquestrador principal, como é possível observar na Figura 18. A sua única responsabilidade é iterar sobre a lista de tabelas de processo (fornecida como um parâmetro do tipo Array). Dentro do *loop* ForEach, ele não contém a lógica de execução, mas sim uma única atividade Execute Pipeline que chama o *pipeline* "Filho", passando-lhe o nome da tabela da iteração atual.

O *pipeline* "Filho" (PL_Worker_ProcessSingleTable) é o responsável pelo processamento de uma única tabela, como demonstra a Figura 19. Ele recebe o nome da tabela e a *tag* da imagem Docker como parâmetros vindos do *pipeline* "Pai" e executa o ciclo de vida completo da tarefa: ingestão, transformação e autolimpeza.

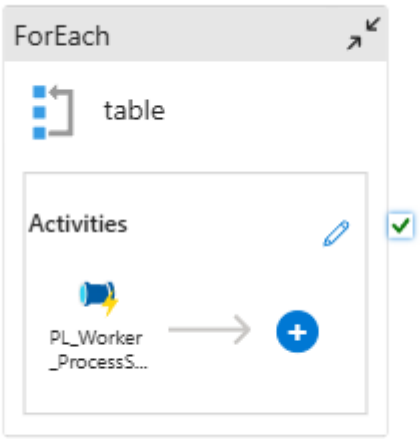


Figura 18 - Diagrama de arquitetura do *pipeline* "Pai"

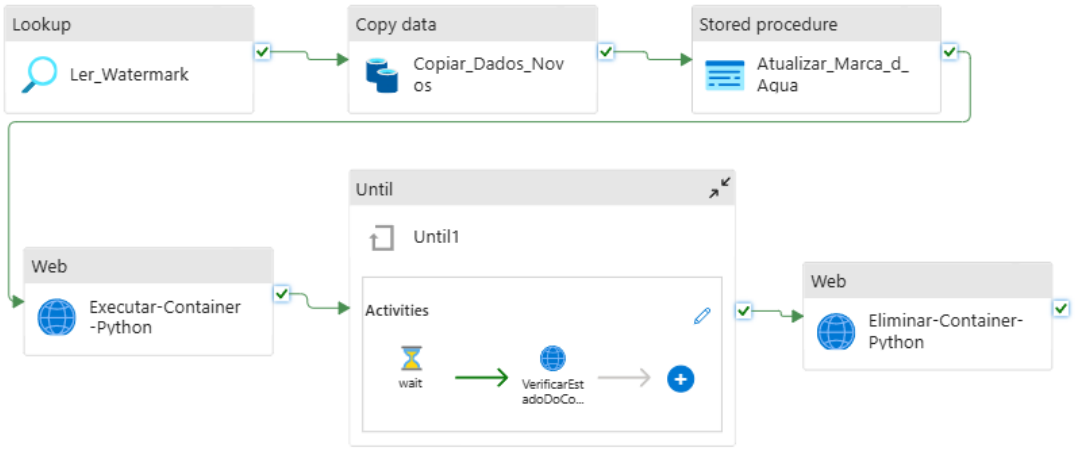


Figura 19 - Diagrama de Arquitetura do *pipeline* "Filho"

A escolha final pelo ACI, em detrimento do Azure Batch, não foi apenas uma solução para a instabilidade encontrada, mas também uma decisão de arquitetura deliberada. O ACI, sendo um serviço *'serverless'*, provou ser mais adequado para este tipo de carga de trabalho transacional e sob demanda, onde cada processo é executado num container efémero. Esta abordagem simplificou a gestão da infraestrutura (eliminando a necessidade de gerir *pools* e nós de computação), reduziu o tempo de arranque por tarefa e otimizou os custos, uma vez que a faturação é feita ao segundo e apenas durante o tempo de execução efetivo do *script* Python.

6.4. Implementação do *pipeline* "Pai" (PL_Run_Process_Mining_IPL)

O *pipeline* PL_Run_Process_Mining_IPL funciona como o orquestrador principal ou "Pai", servindo como o ponto de entrada para a execução automatizada de todo o fluxo de trabalho. A sua lógica é focada na iteração e na delegação de tarefas, recebendo como parâmetros uma lista de tabelas a processar (tables_list) e a versão da imagem Docker a ser utilizada (image_tag).

O elemento central deste *pipeline* é uma atividade *ForEach*, que itera sobre a lista de tabelas fornecida. Uma decisão de *design* crítica foi configurar esta atividade para executar em modo sequencial. Esta abordagem garante que apenas um Azure Container Instance é provisionado de cada vez, o que permite um controlo rigoroso sobre os custos e assegura que as quotas de recursos de CPU disponíveis na subscrição do Azure não são excedidas.

Dentro de cada iteração do *ForEach*, uma atividade Execute Pipeline é responsável por invocar o *pipeline* "Filho" (PL_Worker_ProcessSingleTable). Nesta chamada, são passados os parâmetros necessários para a sua execução: o nome da tabela da iteração atual (através da expressão @item()) e a *tag* da imagem Docker (com a expressão @pipeline().parameters.image_tag), garantindo que cada processo é executado com o contexto correto.

6.5. Implementação do *pipeline* "Filho" (PL_Worker_ProcessSingleTable)

O *pipeline* PL_Worker_ProcessSingleTable representa o componente central e operacional da arquitetura, responsável por executar o ciclo de vida completo do processamento para uma única tabela de processo. A sua conceção foi orientada para a robustez, resiliência e autonomia, integrando padrões de engenharia de dados estabelecidos para garantir a eficiência e a manutenibilidade da solução. O seu fluxo de execução, ilustrado na Figura 18, está segmentado em três fases principais: Ingestão Incremental, Execução da Transformação e, por fim, Monitorização e Limpeza de Recursos.

Para otimizar a performance e minimizar o impacto nos sistemas de origem, a ingestão de dados para a camada de *staging* não realiza uma carga completa (*full load*) a cada execução. Em vez disso, implementa-se o padrão de *High-Watermark*, uma técnica clássica em arquiteturas de *Data Warehouse* para a gestão de cargas incrementais, que consiste em processar apenas os registos adicionados ou modificados desde a última execução bem-sucedida [14]. Este processo é orquestrado por um conjunto de três atividades coordenadas.

6.5.1. Atividade *Lookup* (Ler_Watermark)

O primeiro passo de cada execução é determinar o ponto de partida da carga incremental (Figura 20). Esta atividade estabelece conexão com a *Data Warehouse* e executa uma consulta sobre a tabela de controlo LOG_CARGA_INCREMENTAL (Tabela 3). A consulta é construída dinamicamente para obter o valor da coluna UltimoValorProcessado correspondente à tabela em processamento, retornando o *timestamp* do último evento registado na execução anterior.

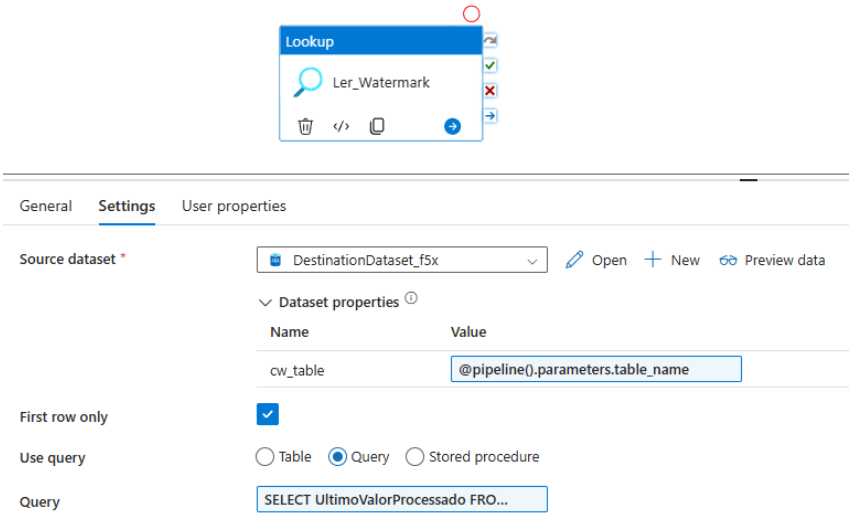


Figura 20 - Atividade *Lookup*

6.5.2. Atividade *Copy Data* (Copiar_Dados_Novos)

Esta atividade constitui o núcleo da ingestão, extraindo seletivamente os novos dados do sistema de origem e carregando-os na tabela de *staging,* com o auxílio da prévia adição dos respetivos *datasets* na secção 'Datasets' do ADF como é demonstrado na Figura 21 e Figura 22. O seu principal componente lógico reside na consulta de origem (*Source*), que é construída dinamicamente em tempo de execução. A cláusula WHERE dessa consulta filtra os registos da vista de origem, selecionando apenas aqueles cujo *timestamp* é posterior ao valor da "marca de água" obtido na atividade Lookup anterior. Para garantir a robustez na primeira execução (quando ainda não existe uma "marca de água"), a lógica recorre a uma data de referência muito antiga (nomeadamente, '1900-01-01') para efetuar uma carga inicial completa (Figura 23).

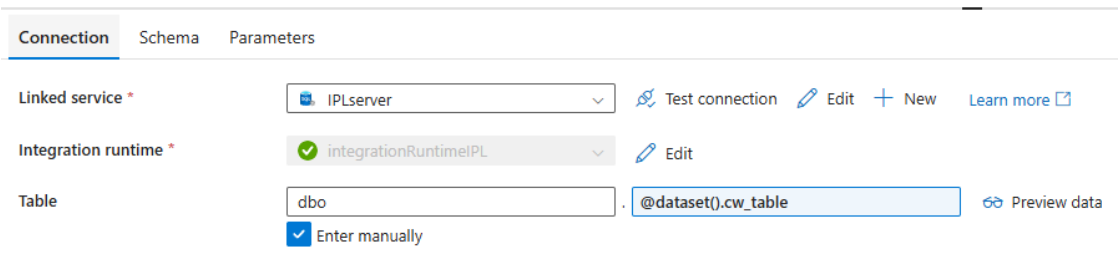


Figura 21 - Configurações do *Dataset* de origem

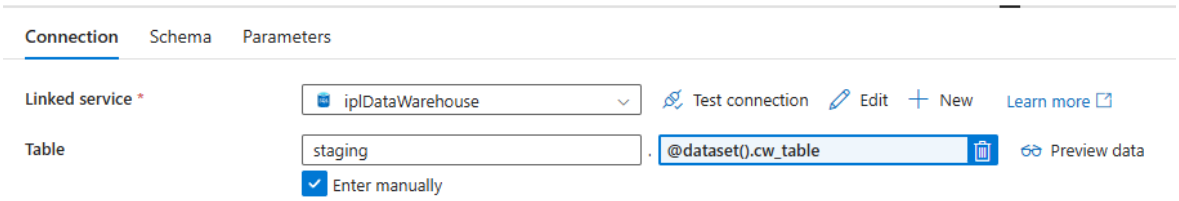


Figura 22 - Configurações do *Dataset* de destino

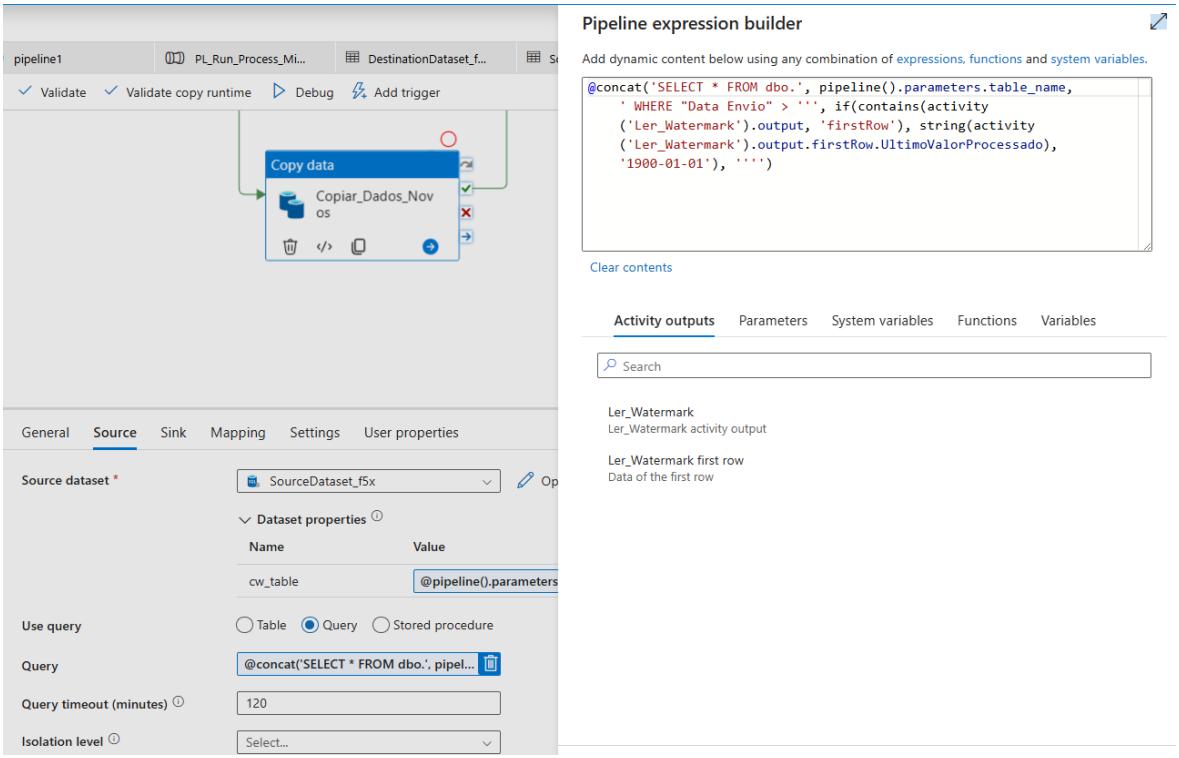


Figura 23 - Atividade *Copy data* e demonstração do uso de "marca de água"

6.5.3. Atividade *Stored Procedure* (Atualizar_Marca_de_Agua)

Após a conclusão bem-sucedida da cópia dos dados, é imperativo registar o novo ponto de paragem (Figura 24). Esta atividade invoca o *stored procedure* sp_update_watermark, passando como parâmetros o nome da tabela processada e o novo valor para a "marca de água". Para garantir consistência, o valor utilizado é o *timestamp* atual da execução do *pipeline* (@utcnow()), assegurando que nenhuma transação seja perdida entre execuções. O uso de um procedimento armazenado encapsula a lógica de UPSERT na base de dados, o que constitui uma prática recomendada para a segurança e manutenibilidade do código [14].

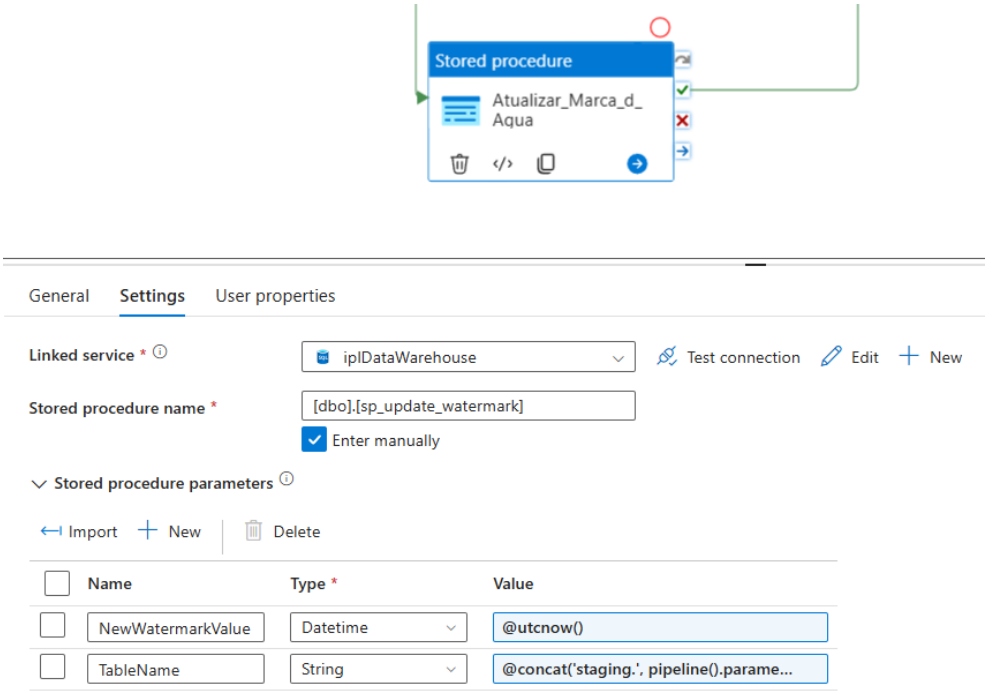


Figura 24 - Atividade *Stored Procedure*

Com os dados na área de *staging* devidamente atualizados, o *pipeline* procede à orquestração da tarefa de computação que executa a transformação. Esta abordagem adere aos princípios da computação *serverless*, em que os recursos são provisionados dinamicamente e sob demanda, abstraindo a gestão da infraestrutura subjacente [24].

6.5.4. Atividade *web* (Executar-Container-Python - método *PUT*)

Esta atividade é responsável por provisionar e iniciar, de forma programática, uma instância de computação no serviço Azure Container Instances (ACI). Para tal, envia um pedido *PUT* para o *endpoint* da *API REST* do *Azure Resource Manager* (Figura 25). O URL do pedido é construído dinamicamente para atribuir um nome único ao *Container Group*, que inclui o nome da tabela e o ID de execução do *pipeline* (@pipeline().RunId), uma prática que facilita a rastreabilidade e a depuração. O corpo (*Body*) do pedido consiste num documento JSON que define exaustivamente o ambiente de execução: a imagem Docker a ser utilizada, o comando a executar (python main.py <nome_da_tabela>), os recursos de CPU e RAM a alocar, e a Identidade Gerida (*Managed Identity*) a ser usada para uma autenticação segura e sem credenciais nos restantes serviços Azure.

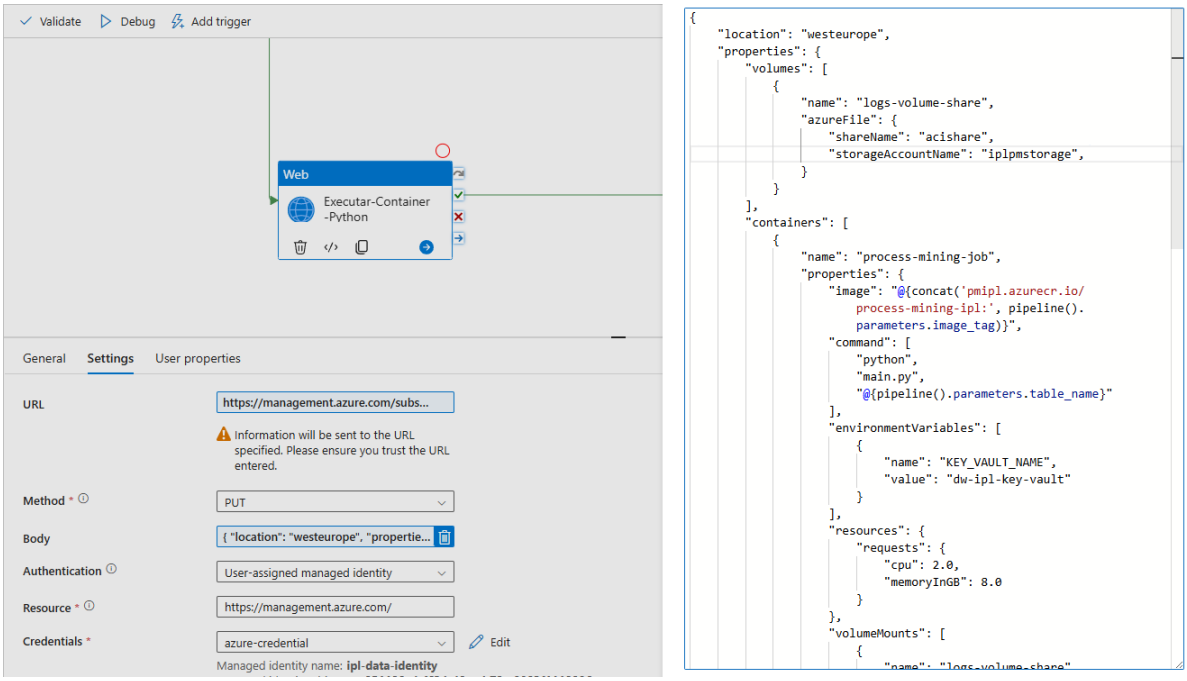


Figura 25 - Atividade *web* (execução do contentor Python)

A iniciação de um *container* através da API é uma operação assíncrona: a API retorna sucesso assim que a ordem de criação é aceite, não esperando que a tarefa termine. Para sincronizar o fluxo do *pipeline* com esta tarefa externa e garantir uma gestão de recursos eficiente, foi implementada uma lógica de monitorização e autolimpeza.

6.5.5. Atividade *Until* (EsperarPelaConclusao)

Esta atividade implementa um mecanismo de *polling* síncrono para monitorizar o estado da tarefa externa. A sua função é colocar em pausa a execução do *pipeline* até que o *container* no ACI tenha concluído o seu trabalho. A condição de saída do ciclo é uma expressão que avalia o estado do *container group*, terminando o *loop* quando este for "*Succeeded*" ou "*Failed*". O ciclo é composto por duas atividades internas, como é possível observar na Figura 26.

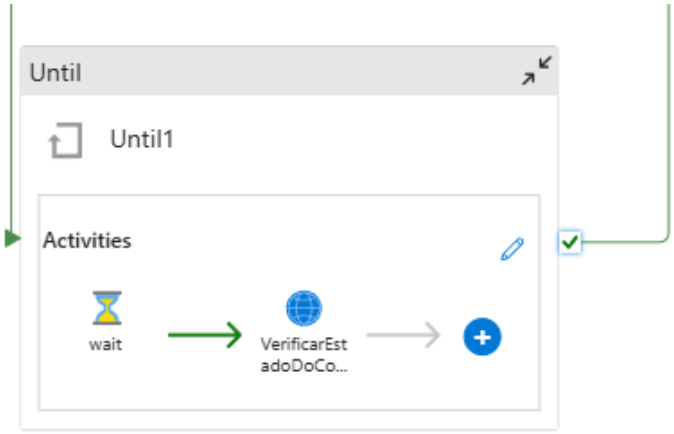


Figura 26 - Atividade *Until* que é executada até o contentor no ACI tenha concluído a sua tarefa

- **Atividade *Wait*:** Introduz uma pausa fixa (atualmente de 45 segundos) entre cada verificação, uma prática recomendada para evitar o envio excessivo de pedidos à API (*API throttling*).
- **Atividade *Web* (VerificarEstadoDoContainer - método *GET*):** Envia um pedido *GET* ao *endpoint* do *container group* para obter o seu estado atual, cujo resultado é usado pela condição de saída do Until (Figura 27).

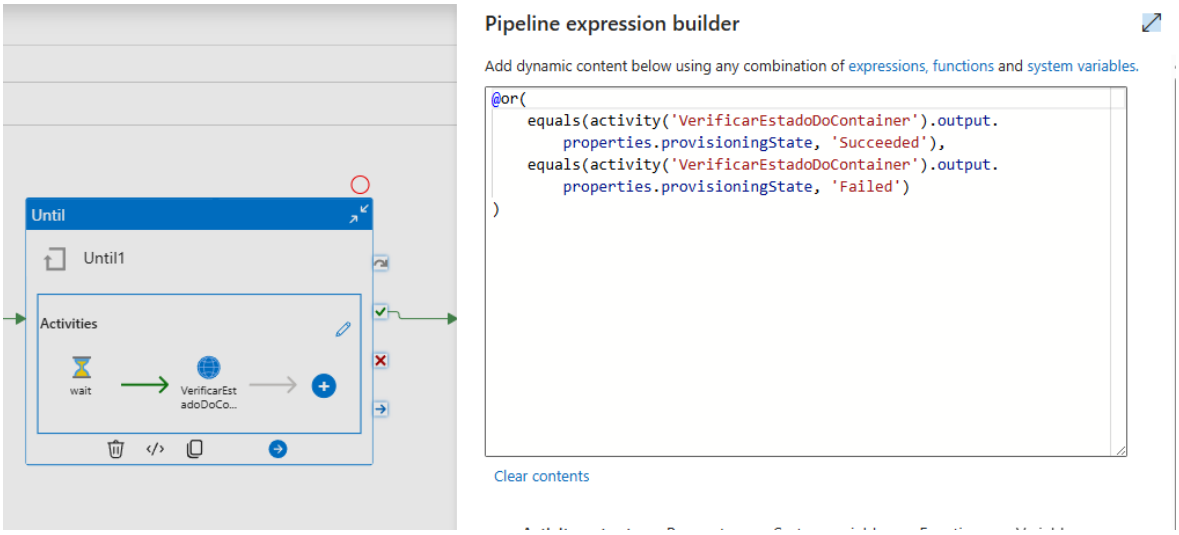


Figura 27 - Condição de saída da atividade *Until*

6.5.6. Atividade *Web* (Eliminar-Container-Python - método *DELETE*)

Assim que o ciclo *Until* termina, esta atividade final é executada. A atividade envia um pedido *DELETE* para o mesmo URL do *container group*, instruindo a Azure a destruir a instância e a libertar todos os recursos de computação associados (Figura 28). Esta etapa implementa um princípio de gestão de recursos efémeros, garantindo que o *pipeline* é "autolimpante" (*self-cleaning*). Desta forma, os custos são otimizados, pois a infraestrutura de computação existe apenas durante o tempo estritamente necessário para a execução da tarefa.

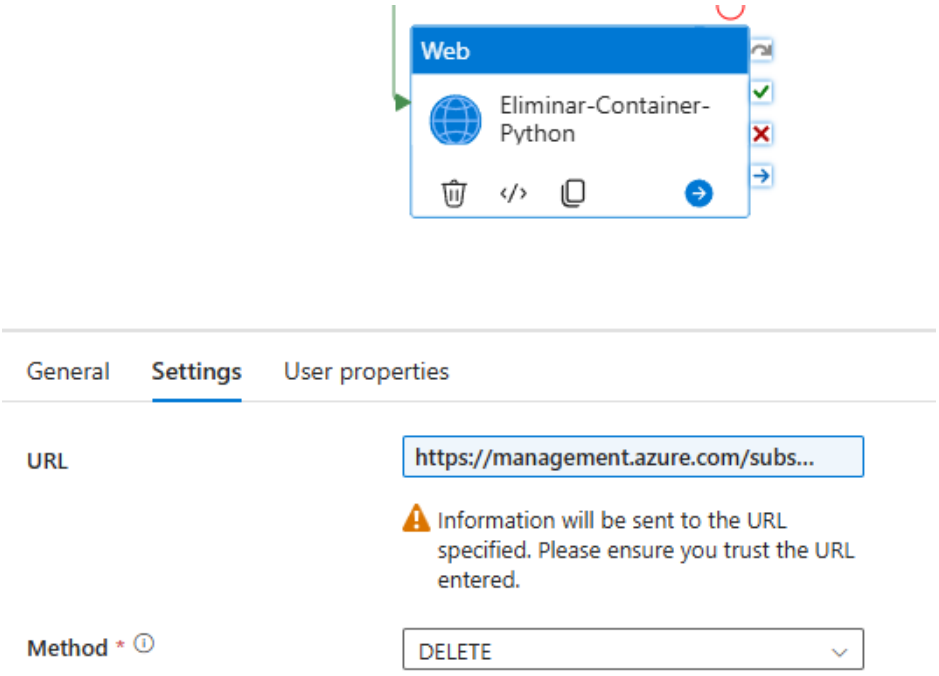


Figura 28 - Atividade *Web* que envia um pedido *DELETE*

6.6. Validação e fiabilidade do *pipeline*

A validação do *pipeline* foi realizada através de uma série de execuções controladas, processando tanto tabelas de pequeno como de grande volume (com mais de 90.000 eventos). A monitorização foi feita através da análise dos *logs* de execução do Azure Data Factory, dos *logs* de eventos dos Azure Container Instances e, crucialmente, dos ficheiros de *log* persistentes gerados pelo *script* Python e guardados no Azure File Share.

Durante a fase de testes, foram detetadas várias classes de erros que são típicas em projetos de engenharia de dados complexos. A resolução destes problemas foi fundamental para a maturação da arquitetura.

1. **Erros de configuração de ambiente:**
    - **Problema:** As primeiras execuções do *container* Docker falharam devido à ausência de dependências de sistema operativo, como o driver Microsoft ODBC e a biblioteca Graphviz. Isto resultava num erro "libodbc.so.2: *cannot open shared object file*", que causava uma falha silenciosa no arranque do *script* Python.
    - **Solução:** O Dockerfile foi refatorizado para usar uma imagem base do Ubuntu 22.04, sobre a qual foi implementada uma instalação explícita e robusta de todas as dependências de sistema necessárias, garantindo um ambiente de execução consistente e completo.
2. **Erros de integridade de dados e concorrência:**
    - **Problema:** A arquitetura inicial, onde cada função geria a sua própria transação, levou a erros de FOREIGN KEY *constraint*. Isto ocorria porque a transação que inseria um novo ProcessoID na DIM_PROCESSO era confirmada antes da transação que tentava inserir um CasoID correspondente, resultando numa condição de corrida (*race condition*) em que a segunda transação não "via" a alteração da primeira a tempo.
    - **Solução:** O *pipeline* foi re-arquitetado para usar uma transação única, controlada pelo orquestrador main.py através do padrão engine.begin(). Todas as operações de escrita na base de dados passaram a ocorrer dentro desta transação. Se qualquer passo falhar, a transação inteira é automaticamente revertida (*ROLLBACK*), garantindo que o Data Warehouse nunca fica num estado inconsistente.
3. **Erros de limitação de recursos (falhas silenciosas):**
    - **Problema:** Verificou-se que os processos com maior volume de dados falhavam sem gerar um erro explícito no Azure Data Factory. A análise dos

logs de autenticação revelou que o *container* ficava tão sobrecarregado ao carregar os dados para a memória com a biblioteca Pandas que os seus serviços internos, como o de metadados de identidade, deixavam de responder. Isto impedia o *script* de obter os segredos do Key Vault, resultando numa falha silenciosa.

- **Solução:** Os recursos alocados ao Azure Container Instance (ACI) no *pipeline* do ADF foram aumentados (no caso de 1 CPU/2GB RAM para 2 CPU/8GB RAM). Adicionalmente, foram ajustados os *timeouts* na *connection string* da base de dados e nas atividades do ADF, tornando o *pipeline* mais resiliente a operações de longa duração.

A eficácia da arquitetura transacional foi claramente validada durante a extensa fase de depuração do projeto. A funcionalidade de *ROLLBACK* automático provou ser um mecanismo de segurança indispensável. Em praticamente todas as execuções de teste que encontraram um erro — quer fosse um erro de integridade de dados, de configuração de ambiente ou de limitação de recursos — a transação foi revertida com sucesso. Isto protegeu eficazmente a integridade do Data Warehouse, impedindo a escrita de dados parciais ou inconsistentes e garantindo que o estado da base de dados permanecia limpo para a tentativa de execução seguinte.

6.7. Agendamento e automação (*triggers*)

A etapa final na operacionalização da arquitetura de dados consistiu na sua completa automação, um passo crucial para garantir a sustentabilidade e a autonomia da solução, eliminando a necessidade de intervenção manual. Para este fim, recorreu-se ao mecanismo de *Triggers* nativo do *Azure Data Factory*, que permite a invocação de *pipelines* com base em agendamentos predefinidos ou em resposta a eventos.

Em vez de uma abordagem monolítica, optou-se por uma estratégia de agendamento diferenciada. Esta foi desenhada para endereçar os diferentes requisitos de negócio no que concerne à atualidade dos dados (*data freshness*), um dos principais atributos de qualidade de um *Data Warehouse* que dita a frequência com que os dados devem ser atualizados para se manterem relevantes para a tomada de decisão [14]. Foram, portanto, implementados dois

triggers distintos que, de forma eficiente, invocam o mesmo *pipeline* "Pai" (PL_Run_Process_Mining_IPL) com diferentes parâmetros.

6.8. *Trigger* de carga diária e consistência

O primeiro *trigger* tem como objetivo assegurar a consistência e a integridade global do *Data Warehouse*, garantindo que todos os processos sejam atualizados de forma completa e regular. Esta execução serve como a linha de base (*baseline*) que certifica a completude dos dados analisados.

Foi configurado um *trigger* do tipo Schedule para execução diária, estrategicamente agendado para um período de baixa atividade nos sistemas de origem (no caso 23h00 visto já se tratar de um espaço temporal fora do horário laboral), de modo a minimizar qualquer impacto de performance (Figura 30). Ao ser associado ao *pipeline*, este *trigger* foi configurado para passar a lista completa de todas as tabelas de processo ao parâmetro tables_list (Figura 29), garantindo uma atualização exaustiva. A *tag* da imagem Docker foi fixada numa versão estável para assegurar a máxima previsibilidade da execução.

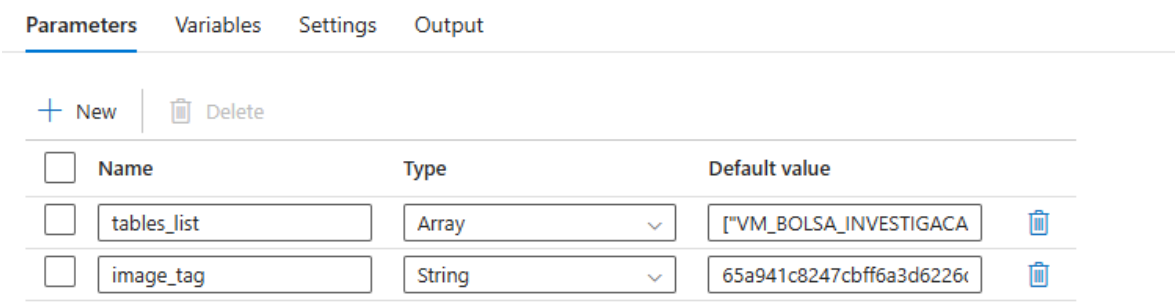


Figura 29 - Exemplo de parâmetros definidos para execução dos *pipelines*

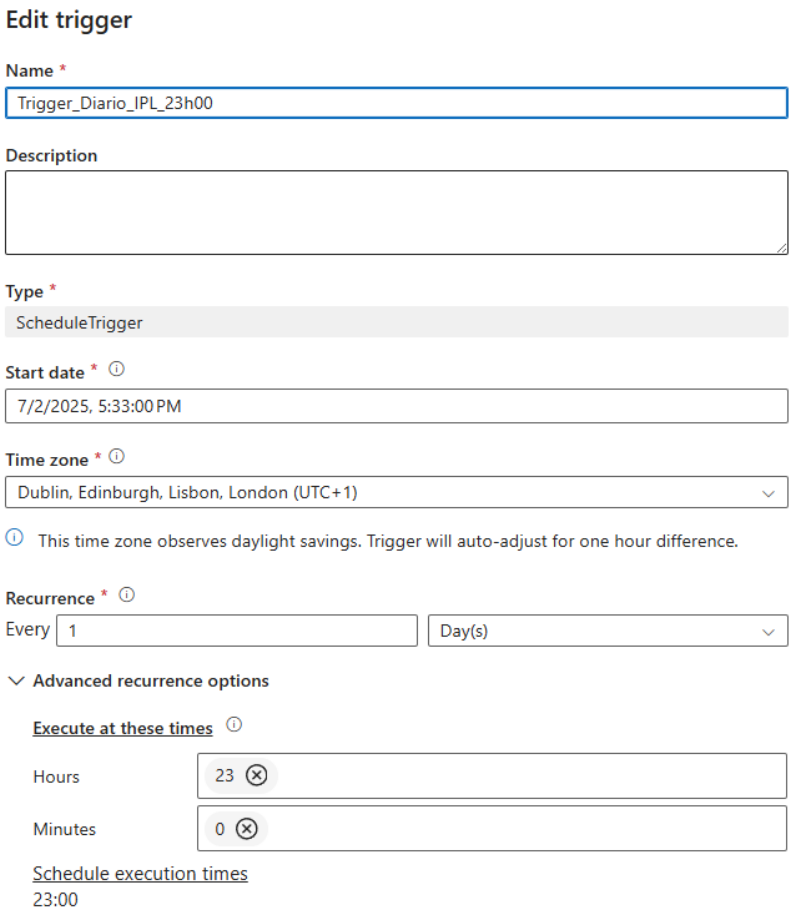


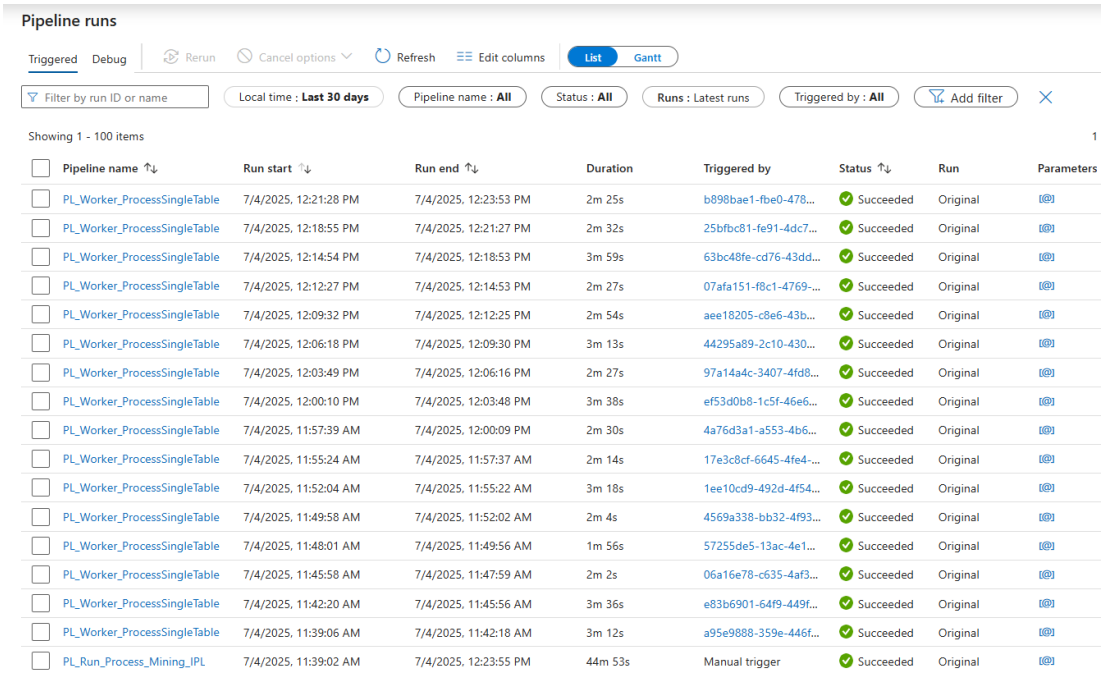
Figura 30 - *Trigger* diário que executa o *pipeline* "pai"

6.9. *Trigger* de alta frequência para processos críticos

A análise exploratória revelou que um subconjunto de processos, nomeadamente os relacionados com os Planos de Atividades do Pessoal (PAP), exibia um volume de dados e uma criticidade de negócio que exigiam uma maior frequência de atualização. O objetivo deste segundo *trigger* é, portanto, fornecer aos gestores uma visão mais tempestiva destes processos de alto impacto.

A flexibilidade da arquitetura parametrizada é aqui demonstrada na sua plenitude. Um segundo *trigger* do tipo Schedule foi criado, configurado para uma cadência de 12 horas. Este *trigger* invoca o mesmo *pipeline* "Pai", mas passa-lhe um conjunto de parâmetros distinto: o parâmetro tables_list é populado apenas com os identificadores das tabelas do PAP. Esta configuração resulta numa execução rápida e focada, que atualiza os dados de maior volume sem o custo computacional de reprocessar todo o *Data Warehouse*.

A implementação desta estratégia de múltiplos *triggers* permite que o sistema responda de forma granular a diferentes Acordos de Nível de Serviço (*Service Level Agreements* - SLAs) relativos à atualidade dos dados, otimizando a alocação de recursos computacionais através de um equilíbrio entre cargas completas, menos frequentes, e cargas parciais, mais regulares e direcionadas aos processos de maior criticidade.



Pipeline name	Run start	Run end	Duration	Triggered by	Status	Run	Parameters
PL_Worker_ProcessSingleTable	7/4/2025, 12:21:28 PM	7/4/2025, 12:23:53 PM	2m 25s	b898bae1-fbe0-478...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 12:18:55 PM	7/4/2025, 12:21:27 PM	2m 32s	25bfbc81-fe91-4dc7...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 12:14:54 PM	7/4/2025, 12:18:53 PM	3m 59s	63bc48fe-cd76-43dd...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 12:12:27 PM	7/4/2025, 12:14:53 PM	2m 27s	07afa151-f8c1-4769-...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 12:09:32 PM	7/4/2025, 12:12:25 PM	2m 54s	aee18205-c8e6-43b...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 12:06:18 PM	7/4/2025, 12:09:30 PM	3m 13s	44295a89-2c10-430...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 12:03:49 PM	7/4/2025, 12:06:16 PM	2m 27s	97a14a4c-3407-4fd8...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 12:00:10 PM	7/4/2025, 12:03:48 PM	3m 38s	ef53d0b8-1c5f-46e6...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:57:39 AM	7/4/2025, 12:00:09 PM	2m 30s	4a76d3a1-a553-4b6...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:55:24 AM	7/4/2025, 11:57:37 AM	2m 14s	17e3c8cf-6645-4fe4-...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:52:04 AM	7/4/2025, 11:55:22 AM	3m 18s	1ee10cd9-492d-4f54...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:49:58 AM	7/4/2025, 11:52:02 AM	2m 4s	4569a338-bb32-4f93...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:48:01 AM	7/4/2025, 11:49:56 AM	1m 56s	57255de5-13ac-4e1...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:45:58 AM	7/4/2025, 11:47:59 AM	2m 2s	06a16e78-c635-4af3...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:42:20 AM	7/4/2025, 11:45:56 AM	3m 36s	e83b6901-64f9-449f...	Succeeded	Original	
PL_Worker_ProcessSingleTable	7/4/2025, 11:39:06 AM	7/4/2025, 11:42:18 AM	3m 12s	a95e9888-359e-446f...	Succeeded	Original	
PL_Run_Process_Mining_IPL	7/4/2025, 11:39:02 AM	7/4/2025, 12:23:55 PM	44m 53s	Manual trigger	Succeeded	Original	

Figura 31 - *Pipelines* realizados com sucesso após *trigger*

7. Verificação e validação da solução

A validação final da arquitetura foi realizada sobre um universo de 16 processos de negócio distintos. O pipeline de ETL processou um total de mais de 200,000 eventos (linhas nas tabelas de staging), que foram consolidados num Data Warehouse dimensional. A execução do *pipeline* para um processo de média dimensão (por exemplo 18,000 eventos) demonstrou ser eficiente, completando o ciclo de ingestão, transformação e carga em aproximadamente 2 minutos e 30s segundos, validando a performance da solução baseada em Azure Container Instances.

O *benchmark* foi executado utilizando um dos processos de maior volume, o VM_PEDIDO_JUSTIFICACAO_FALTAS, que contém aproximadamente 18.000 eventos. O tempo total de execução do *script* main.py foi medido em ambos os ambientes, desde o início da extração dos dados do *staging* até à confirmação final da transação na base de dados.

Os ambientes de teste foram os seguintes:

- **Ambiente *cloud*:** Um Azure Container Instance configurado com 2 vCPUs e 8 GB de RAM, a correr na mesma região Azure (Oeste da Europa) que a base de dados do Data Warehouse.
- **Ambiente local:** Uma estação de trabalho de desenvolvimento típica (processador Intel Core i5, 32 GB de RAM, SSD NVMe), a executar o *script* Python diretamente no sistema operativo Windows, com ligação à base de dados Azure SQL através de uma ligação VPN.

Os tempos de execução foram registados e demonstram um *trade-off* interessante entre a proximidade da rede e o poder de computação bruto, como podemos observar na tabela do Anexo K.

Os resultados do *benchmark* confirmam a observação inicial de que a execução local pode, de facto, ser marginalmente mais rápida em termos de tempo total para um único processo. Isto deve-se principalmente à superioridade do poder de processamento de uma estação de trabalho moderna na execução de tarefas intensivas em CPU e RAM, como as transformações de dados realizadas pela biblioteca Pandas.

No entanto, a análise também revela a principal vantagem do ambiente *cloud*: a eficiência das operações de rede. As fases de extração e carga de dados foram consideravelmente mais rápidas no ACI por este estar localizado na mesma rede de alta velocidade que a base de dados Azure SQL.

Apesar da ligeira vantagem de tempo da execução local para uma única tarefa, a decisão de utilizar o Azure Container Instances para o *pipeline* automatizado permanece a mais correta e robusta pelas seguintes razões estratégicas:

- A arquitetura no ADF permite a execução de múltiplos containers em paralelo (desmarcando a opção "*Sequential*" no *ForEach*). Seria impossível replicar esta escalabilidade horizontal numa única máquina local para processar todas as 16 tabelas em simultâneo.
- O ambiente Docker garante que o *script* é executado sempre exatamente da mesma forma, com as mesmas dependências, eliminando o risco de erros causados por variações no ambiente local.

7.1. Implementação da camada de apresentação (Power BI)

A verificação final de uma arquitetura de *Business Intelligence* reside na sua capacidade de transformar os dados processados em conhecimento acionável e fidedigno. Neste contexto, a implementação da camada de apresentação serve como a validação funcional e de integração de todo o *pipeline* de dados desenvolvido. É nesta fase que se comprova que os dados extraídos, transformados e carregados para o Data Warehouse não só estão tecnicamente corretos, mas também respondem eficazmente aos requisitos analíticos do negócio.

A ferramenta selecionada para esta tarefa foi o Microsoft Power BI, uma plataforma líder de mercado em *Business Intelligence* e análise de dados, reconhecida pela sua capacidade de criar relatórios interativos e pela sua forte integração com o ecossistema de dados da Microsoft [25].

O trabalho nesta camada foi dividido em três etapas principais: a conexão à fonte de dados, a modelação de dados dentro do Power BI e, finalmente, a construção dos relatórios e *dashboards* (Figura 32).

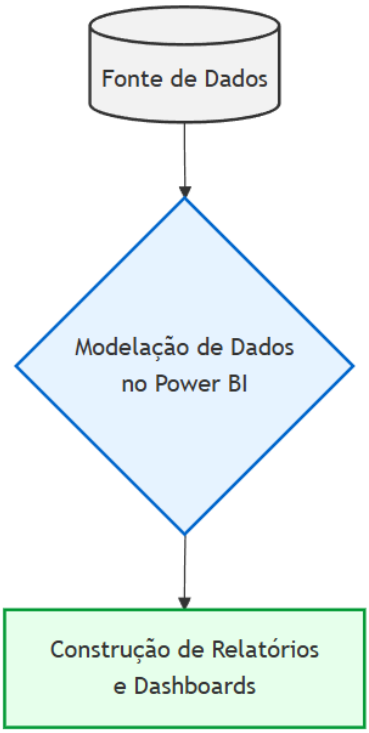


Figura 32 - Três principais etapas percorridas na camada de apresentação

7.1.1. Conexão ao *Data Warehouse* e definição do modo de armazenamento

A primeira etapa na construção da camada de visualização consistiu em estabelecer a ligação entre a ferramenta de desenvolvimento *Power BI Desktop* e a *Data Warehouse* implementada na *Azure SQL Database*. A conexão foi realizada através do conector nativo SQL Server, disponibilizado pelo Power BI.

Nesta fase, foi tomada uma decisão de arquitetura crítica relativamente ao Modo de Armazenamento (*Storage Mode*). Esta escolha determina a forma como o Power BI interage com os dados de origem e tem implicações diretas na performance e na atualidade dos dados. As duas opções principais são:

- **Modo *Import*:** Os dados são extraídos da fonte, comprimidos e carregados para o motor de análise em memória do Power BI, o *VertiPaq*. Este modo, conceptualmente alinhado com arquiteturas MOLAP (*Multidimensional Online Analytical*

Processing), oferece uma performance de consulta extremamente elevada, mas a atualidade dos dados está limitada pela frequência do ciclo de atualização (*refresh*).

- **Modo DirectQuery:** Neste modo, o Power BI atua como uma camada de metadados e visualização, não armazenando uma cópia dos dados. Cada interação do utilizador no relatório (como, aplicar um filtro) é traduzida em tempo real para uma ou mais consultas SQL, que são enviadas diretamente à base de dados de origem. Esta abordagem corresponde ao padrão ROLAP (*Relational Online Analytical Processing*).

Para este projeto, foi selecionado o modo *DirectQuery* (Figura 33). A justificação para esta escolha reside no requisito de negócio de fornecer aos gestores uma visão o mais atualizada possível dos processos, um pilar das estratégias de *Real-Time Business Intelligence* [25]. Para processos de alto volume e criticidade, como os do PAP, que recebem centenas de novas linhas diariamente, o *DirectQuery* garante que os relatórios refletem o estado dos dados na base de dados no momento exato da consulta, eliminando a latência de um ciclo de atualização agendado. Embora esta abordagem possa implicar uma maior dependência da performance da base de dados subjacente, foi considerada a mais adequada para cumprir os requisitos de atualidade dos dados definidos para o projeto.

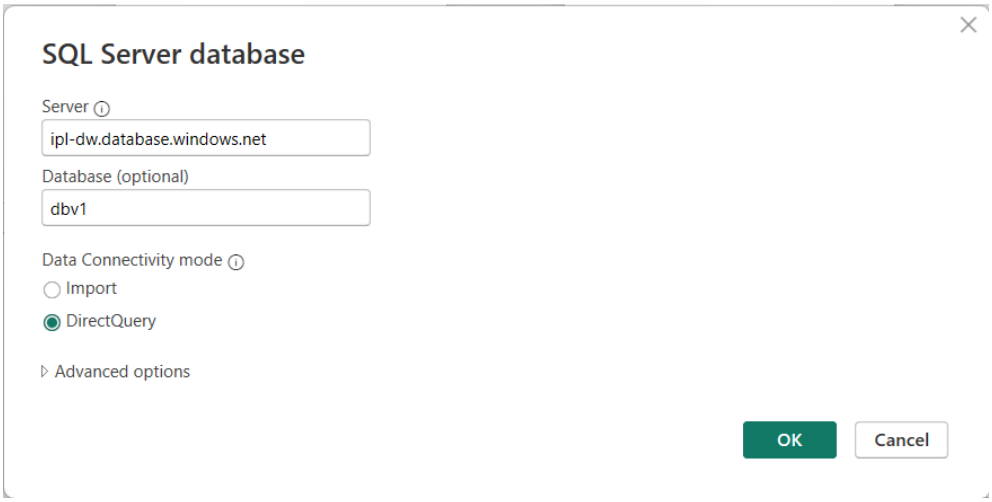


Figura 33 - Ecrã de configuração da conexão SQL Server no Power BI, seleção do modo *DirectQuery*

7.1.2. Modelação de dados no Power BI

Após estabelecer a conexão, a etapa subsequente consistiu na visualização da replicação da estrutura lógica da *Data Warehouse* dentro do ambiente do Power BI. Este processo, realizado na Vista de Modelo (*Model View*), é fundamental para que o motor de análise compreenda as associações entre as tabelas, permitindo que os filtros e as agregações funcionem de forma correta e eficiente. O modelo foi automaticamente construído após a inserção das tabelas da fonte de dados, indicando que o modelo de dados após o sucesso dos *triggers* e *pipelines* serve como uma representação fiel da arquitetura dimensional projetada na *Data Warehouse*, como se pode observar na Figura 34.

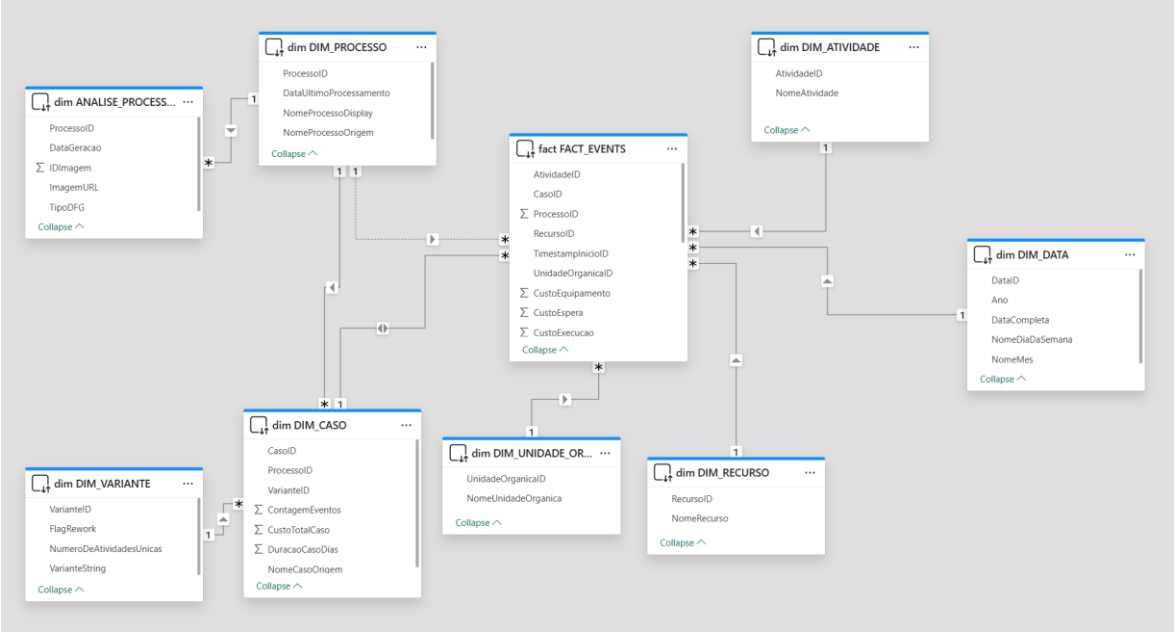


Figura 34 - Vista modelo no Power BI após inserção das tabelas

Com base na Figura 34, podemos observar as seguintes relações:

- Relações do esquema em estrela: Foram criadas as relações primárias de um-para-muitos (1 -> *) entre as tabelas de dimensão principais (DIM_PROCESSO, DIM_ATIVIDADE, DIM_RECURSO, DIM_UNIDADE_ORGANICA, DIM_DATA) e a tabela de factos central (FACT_EVENTS), utilizando as respetivas chaves substitutas inteiras (*surrogate keys*).

- Relação em floco de neve: Para completar o modelo, foi estabelecida a relação de um-para-muitos (1 -> *) entre a DIM_VARIANTE e a DIM_CASO, representando a característica do floco de neve (*snowflaked*) da dimensão de caso.

- Ativação do filtro bidirecional: Um ajuste de modelação crucial foi a ativação do filtro de direção cruzada (*cross-filter direction*) para "Ambos" (*Both*) na relação entre a DIM_CASO e a FACT_EVENTS. Num esquema em estrela puro, os filtros propagam-se das dimensões para a tabela de factos. Contudo, devido à natureza *snowflaked* da dimensão de caso, esta configuração é necessária para permitir que um filtro aplicado numa dimensão principal (tal como DIM_DATA) se propague corretamente através da tabela de factos, para a DIM_CASO, e daí para a DIM_VARIANTE. Esta técnica simplifica drasticamente a experiência de análise interativa do utilizador e a complexidade das medidas DAX necessárias para calcular indicadores ao nível da variante.

7.1.3. Visualização de dados e análise de processos

A fase final consistiu na construção dos relatórios e *dashboards* na Vista de Relatório (*Report View*) do Power BI. Esta etapa foi realizada em estreita colaboração com a colega Valeria Balitkaia, do Mestrado em Ciência de Dados do Politécnico de Leiria, que foi responsável pelo design e desenvolvimento da camada de visualização.

Foram criados cartões para apresentar métricas agregadas de alto nível, como o Custo Total, a Duração Média por Caso, entre outros, onde é possível observar na Figura 35.

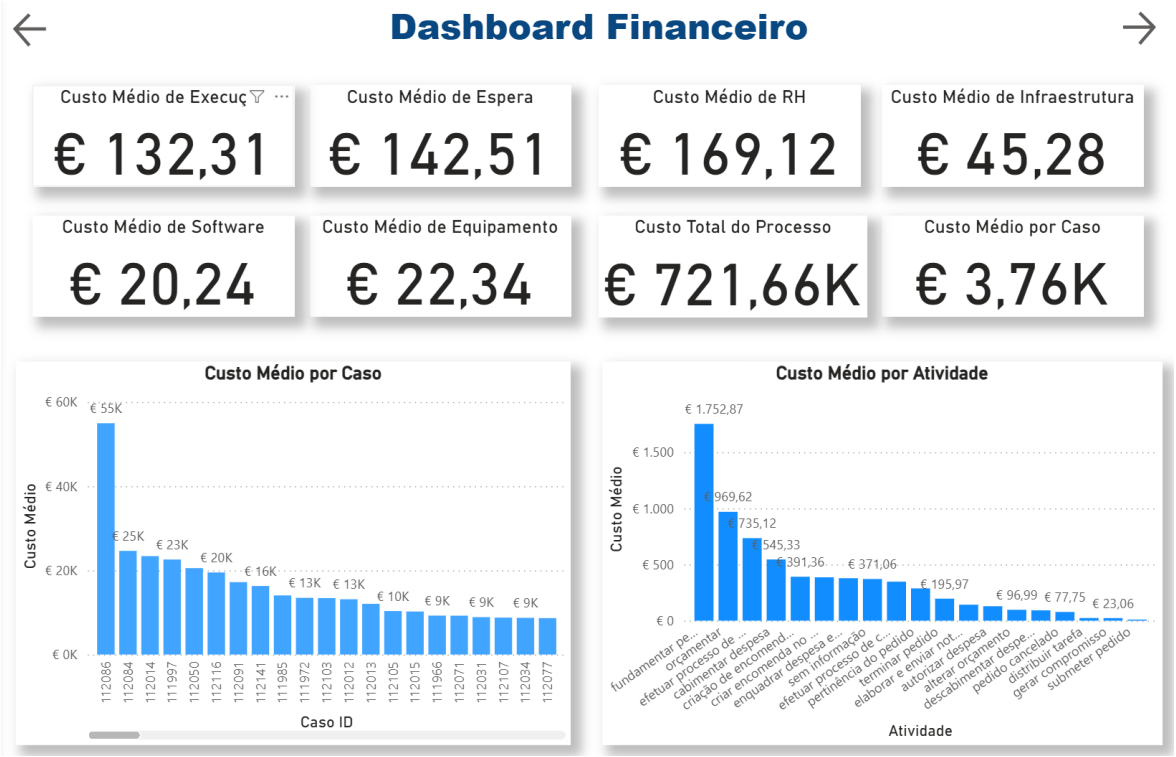


Figura 35 - *Dashboard* financeiro do Power BI

Foram implementados gráficos de barras e tabelas para permitir a análise comparativa, como a frequência de variantes e a duração por caso, permitindo a ordenação e a identificação de *outliers* (Figura 36).

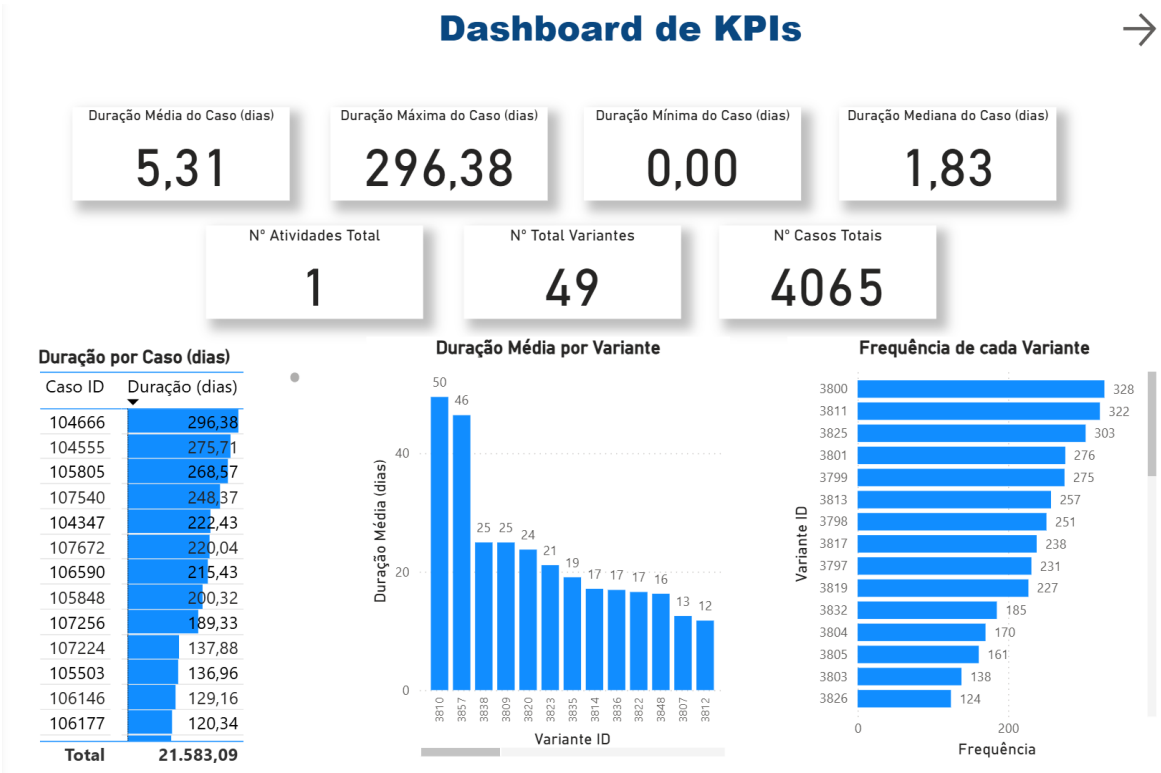


Figura 36 - *Dashboard* de KPI's do Power BI

Foram adicionados filtros para todas as dimensões principais (Processo, Ano, Unidade Orgânica, Recurso), permitindo ao utilizador segmentar os dados e focar a sua análise (Figura 37).

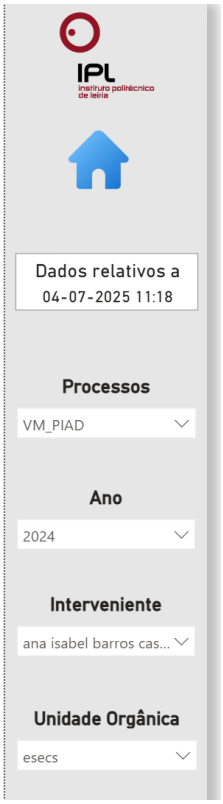


Figura 37 - Filtros presentes nos *dashboards* do Power BI

A integração dos artefactos visuais foi um ponto chave. A coluna ImagemURL da tabela ANALISE_PROCESSO_DFG_IMAGENS foi configurada com a categoria de dados *Web* URL. Isto permite que, ao ser utilizada num visual de tabela ou num visual customizado, o Power BI renderize diretamente a imagem do DFG armazenada no Azure Blob Storage, fornecendo uma representação visual do fluxo do processo lado a lado com as métricas quantitativas. A par do mapa visual, são apresentados KPIs cruciais calculados através de medidas DAX (Referência de Expressões de Análise de Dados), como a percentagem de casos que envolvem retrabalho (*rework*), permitindo uma análise imediata da eficiência do processo (Figura 38). Para casos em que a as imagens ficam impercetíveis devido à limitação do tamanho de imagem do Power BI (512x512 pixéis) é possível clicar na imagem, e esta aparecerá maior e com mais definição no browser predefinido do sistema operativo do utilizador (Figura 39).

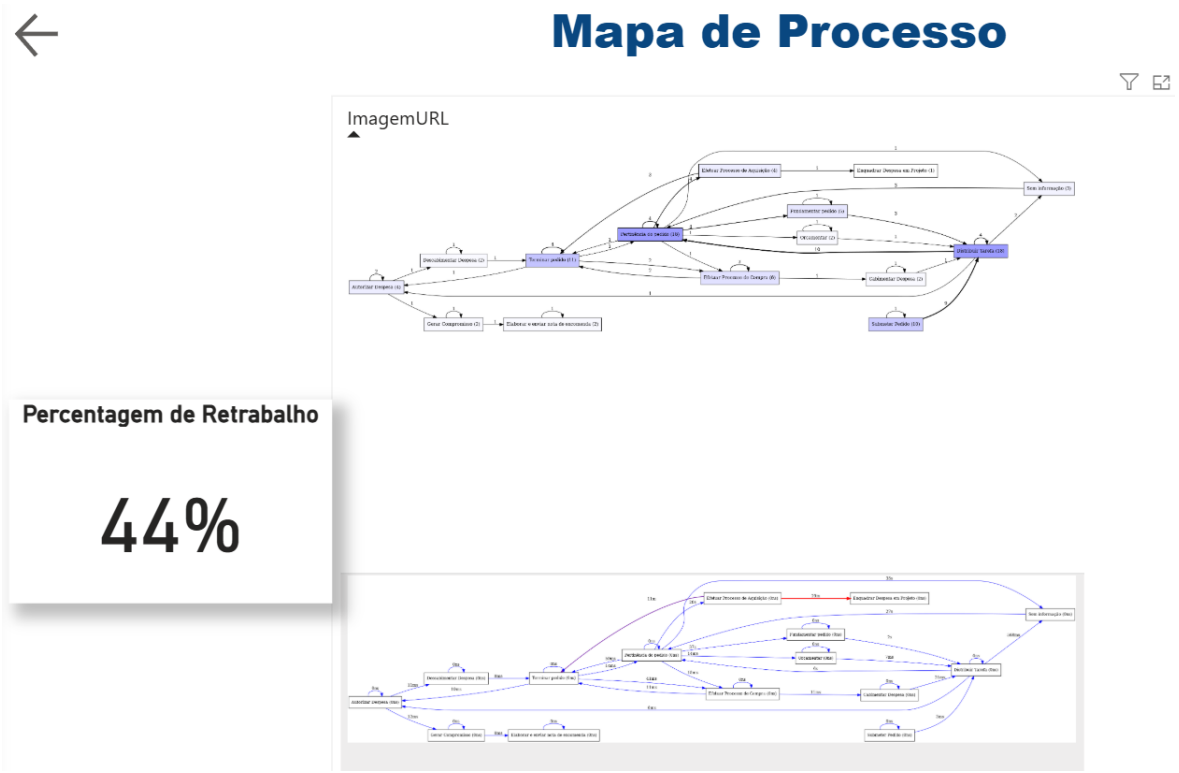


Figura 38 - *Dashboard* das imagens DFG do Power BI

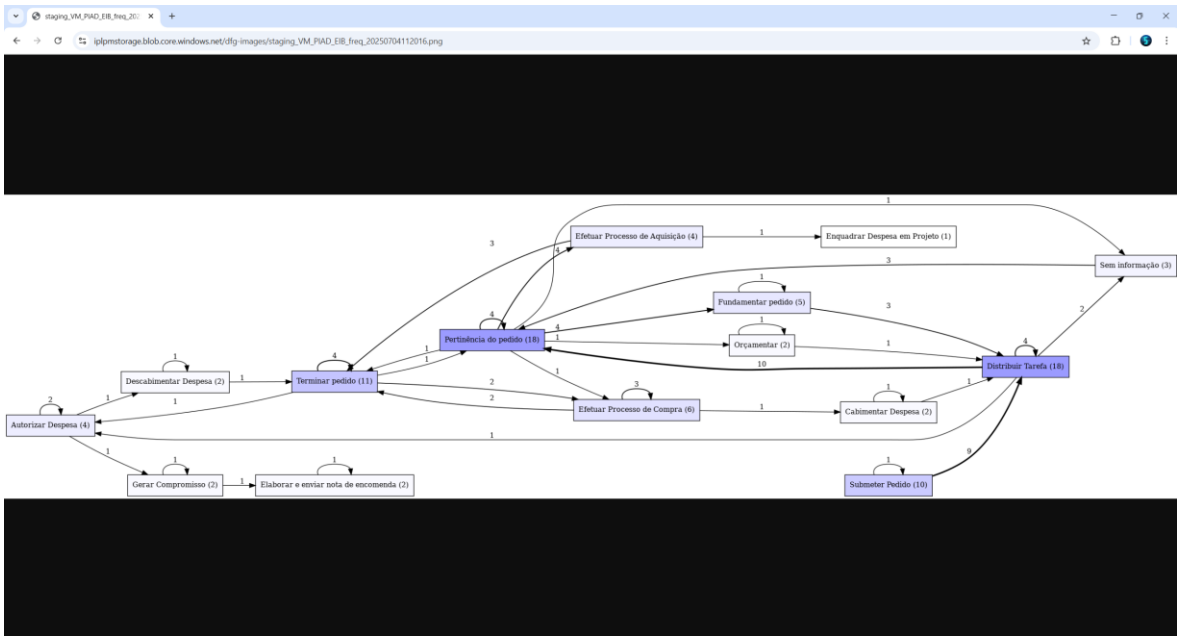


Figura 39 - Imagem DFG no *browser* após clique na imagem do Power BI

8. Conclusões e trabalho futuro

Este capítulo final consolida o trabalho desenvolvido no âmbito da dissertação. A secção 8.1 apresenta uma síntese do projeto, quantificando os seus resultados e avaliando o cumprimento dos objetivos. A secção 8.2 reflete sobre as principais limitações encontradas e as aprendizagens adquiridas. Finalmente, a secção 8.3 delineia possíveis direções para a evolução e expansão futura do projeto PM4IPLeiria.

8.1. Síntese do trabalho desenvolvido

O presente trabalho teve como propósito central a conceção e implementação de uma solução de *Business Intelligence* e *Process Mining*, destinada à análise e otimização dos processos de negócio no Instituto Politécnico de Leiria. A execução do projeto resultou na entrega de uma arquitetura de dados funcional e automatizada, que efetivamente transforma dados operacionais brutos em *insights* analíticos e visuais, endereçando os objetivos técnicos delineados na introdução deste documento.

A materialização desta arquitetura traduziu-se num conjunto de artefactos técnicos quantificáveis, que evidenciam o âmbito e a complexidade da solução desenvolvida:

- Cobertura de processos e volume de dados: Foram integrados e modelados 16 processos de negócio distintos, correspondendo a um volume total superior a 230.000 eventos processados e carregados para a *Data Warehouse* durante a fase de desenvolvimento e validação.

- Arquitetura da *Data Warehouse*: Foi projetada e implementada uma *Data Warehouse* relacional, seguindo um Esquema em floco de neve. A estrutura é composta por três esquemas lógicos (staging, dim e fact) e um total de 12 tabelas relacionais (uma tabela de factos, nove tabelas de dimensão e duas tabelas de controlo para a automação).

- Componente de processamento em Python: Foi desenvolvido um componente de transformação de dados, composto por oito módulos Python reutilizáveis. Este executa uma sequência de operações, incluindo a limpeza e normalização dinâmica de dados, o cálculo de métricas temporais baseadas em horas úteis, a aplicação de

modelos de custos, a descoberta de variantes de processo com recurso a *hashing*, e a geração de grafos de sequência (*Directly-Follows Graphs*).

- Infraestrutura e orquestração na *cloud*: Foi provisionada uma infraestrutura na plataforma *Microsoft Azure* que integra seis serviços principais (*SQL Database*, *Data Factory*, *Container Instances*, *Container Registry*, *Key Vault* e *Blob Storage*). O fluxo de trabalho foi totalmente automatizado através de uma arquitetura "Pai-Filho" no *Azure Data Factory*, suportada por múltiplos gatilhos (*triggers*) agendados que respondem a diferentes requisitos de negócio.

- Integração e entrega contínua (CI/CD): Foi implementado um *pipeline* de CI/CD através de *GitHub Actions*, que automatiza a construção, o versionamento (com *tags* baseadas no *hash* do Git) e a publicação das imagens Docker no *Container Registry*, garantindo assim a consistência e rastreabilidade do ambiente de execução.

8.2. Aquisição de competências e limitações

O desenvolvimento de um projeto desta dimensão permitiu não só atingir os objetivos propostos, mas também identificar um conjunto de desafios e limitações que constituíram importantes oportunidades de aprendizagem.

O maior desafio encontrado foi, sem dúvida, a depuração de erros silenciosos no ambiente de execução na *cloud*. A interação entre múltiplos serviços (ADF, ACI, ACR, Key Vault) revelou problemas subtis de permissões, configuração de rede, gestão de recursos (esgotamento de memória em containers) e *timeouts* que não eram aparentes em ambiente de desenvolvimento local. A resolução destes problemas exigiu a implementação de técnicas de *logging* persistente (com Azure File Shares) e uma depuração metódica e iterativa, o que constituiu uma experiência inestimável sobre a robustez necessária para sistemas em produção.

Outra limitação inerente ao projeto prende-se com a dependência da qualidade dos dados de origem. O pipeline foi desenhado para ser resiliente a inconsistências comuns (nulos, espaços em branco, variações textuais), mas a sua eficácia depende da estabilidade do esquema das vistas SQL de origem. Qualquer alteração estrutural nessas vistas exigiria uma manutenção correspondente no código de extração. Adicionalmente, o facto de os dados

serem anonimizados, embora essencial para a privacidade, limita a profundidade de certas análises, como a correlação da performance individual de recursos com outros atributos demográficos.

A nível pessoal e profissional, o projeto foi fundamental para consolidar a transição de um perfil de desenvolvimento de software para um de engenharia de dados. A necessidade de justificar cada decisão de arquitetura — desde a escolha de um Esquema em floco de neve até à implementação de uma *pipeline* de CI/CD — reforçou o pensamento crítico e o foco em criar soluções que não são apenas funcionais, mas também eficientes, escaláveis e seguras.

8.3. Trabalho futuro

A arquitetura implementada é robusta e foi desenhada para ser escalável, abrindo portas para várias evoluções futuras interessantes:

- Integração com *Machine Learning*: O Data Warehouse agora consolidado é uma fonte de dados ideal para treinar modelos de *Machine Learning*. Poder-se-ia desenvolver um modelo de *Predictive Process Mining* para, por exemplo, prever a duração ou o custo de um caso no momento em que ele é iniciado, ou para detetar anomalias em tempo real.

- Expansão do Data Warehouse para Novos Processos: A arquitetura modular e o *pipeline* parametrizado foram desenhados para facilitar a adição de novos processos de negócio. O próximo passo seria integrar outras fontes de dados do Politécnico de Leiria no Data Warehouse, expandindo o seu alcance e valor analítico.

- Desenvolvimento de capacidades de *Self-Service BI*: Com o modelo de dados bem definido e governado, poder-se-ia dar formação a analistas de negócio do Politécnico de Leiria para que eles próprios pudessem criar os seus próprios relatórios no Power BI, conectando-se ao Data Warehouse como uma fonte de verdade, promovendo uma cultura de tomada de decisão baseada em dados em toda a instituição.

O trabalho desenvolvido cumpriu, assim, o seu objetivo principal: a criação de uma arquitetura de dados automatizada e reutilizável. A solução implementada dota o Politécnico de Leiria da capacidade de extrair conhecimento objetivo a partir dos seus registos digitais,

constituindo uma ferramenta de apoio à decisão e um passo em direção a uma gestão mais informada e orientada para a melhoria contínua dos seus processos.

Bibliografia

[1] R. Hobeck, L. Pufahl, and I. Weber, "EasyChair Preprint Process Mining on Curriculum-Based Study Data-a Case Study at a German University Process Mining on Curriculum-based Study Data: A Case Study at a German University," 2022.

[2] A. Bogarín, R. Cerezo, and C. Romero, "A survey on educational process mining," *Wiley Interdiscip Rev Data Min Knowl Discov*, vol. 8, no. 1, p. e1230, Jan. 2018, doi: 10.1002/WIDM.1230.

[3] D. R. Ferreira and E. Vasilyev, "Using logical decision trees to discover the cause of process delays from event logs," *Comput Ind*, vol. 70, pp. 194–207, Jun. 2015, doi: 10.1016/J.COMPIND.2015.02.009.

[4] M. Wagner *et al.*, "A Combined Approach of Process Mining and Rule-based AI for Study Planning and Monitoring in Higher Education," *Lecture Notes in Business Information Processing*, vol. 468 LNBIP, pp. 513–525, Nov. 2022, doi: 10.1007/978-3-031-27815-0_37.

[5] W. Van der Aalst, "Process mining: Data science in action," *Process Mining: Data Science in Action*, pp. 1–467, Jan. 2016, doi: 10.1007/978-3-662-49851-4/COVER.

[6] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, "Fundamentals of business process management: Second Edition," *Fundamentals of Business Process Management: Second Edition*, pp. 1–527, Mar. 2018, doi: 10.1007/978-3-662-56509-4/COVER.

[7] T. Hildebrandt and L. Mehnen, "Cross-course Process Mining of Student Clickstream Data -- Aggregation and Group Comparison," *Proceedings - 2024 26th International Conference on Business Informatics, CBI 2024*, pp. 90–98, Sep. 2024, doi: 10.1109/CBI62504.2024.00020.

[8] A. Nammakhunt, T. Sooksai, A. Jaidee, W. Milinthapunya, U. Yamchuti, and M. Sukkri, "Leveraging Process Mining to Identify Bottlenecks in Online Applications: A Case Study with Generalizable Insights," *International Conference on ICT and Knowledge Engineering*, 2024, doi: 10.1109/ICTKE62841.2024.10787185.

[9] M. Crosslin, K. Breuer, N. Milikić, and J. T. Dellinger, "Understanding student learning pathways in traditional online history courses: utilizing process mining analysis on clickstream data," *Journal of Research in Innovative Teaching & Learning*, vol. 14, no. 3, pp. 399–414, Nov. 2021, doi: 10.1108/JRIT-03-2021-0024.

[10] J. Park, K. Denaro, F. Rodriguez, P. Smyth, and M. Warschauer, "Detecting changes in student behavior from clickstream data," *ACM International Conference Proceeding Series*, pp. 21–30, Mar. 2017, doi: 10.1145/3027385.3027430.

[11] A. Miguel *et al.*, "Observatório de Ciência da Informação da Universidade do Porto: um projeto colaborativo de sucesso.," *Cadernos BAD*, vol. 0, no. 1, pp. 57–70, Jul. 2015, doi: 10.48798/cadernosbad.1230.

[12] P. Chapman, "CRISP-DM 1.0: Step-by-step data mining guide," 2000.

[13] G. Kim, J. Humble, and J. Willis, "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations by Gene Kim | Goodreads," *IT Revolution Press*, 2016.

[14] R. Kimball and M. Ross, "Kimball DW/BI Lifecycle Overview," *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, pp. 403–427, 2013.

[15] J. Humble and D. Farley, "Continuous delivery : reliable software releases through build, test, and deployment automation," 2010.

[16] P. Duvall, S. Matyas, and a Glover, "Continuous integration: improving software quality and reducing risk," *Vasa*, p. 318, 2007.

[17] S. Brown, "C4 model." Accessed: Sep. 08, 2025. [Online]. Available: https://c4model.com/

[18] W. H. . Inmon, "Building the data warehouse," p. 500, 2005.

[19] "Kimball Dimensional Modeling Techniques".

[20] Itzik. Ben-Gan, "T-SQL querying," 2015.

[21] Vidette. Poe, Patricia. Klauer, and Stephen. Brobst, "Building a data warehouse for decision support," *ACM Digital Library*, no. November, p. 285, 1998.

[22] R. C. Martin, “Clean Architecture: A Craftsman’s Guide to Software Structure and Design (Robert C. Martin Series),” pp. 1–432, 2017.

[23] Hector. Garcia-Molina, J. D. . Ullman, and Jennifer. Widom, “Database systems : the complete book,” p. 1203, 2009.

[24] M. Kleppmann, “Designing Data-Intensive Applications THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS,” 2017.

[25] Ramesh. Sharda, Dursun. Delen, Efraim. Turban, J. E. . Aronson, T.-Peng. Liang, and David. King, “Business intelligence, analytics, and data science : a managerial perspective,” p. 486, 2018.

Anexos

Anexo A - Tabela comparativa de ferramentas de *process mining*

Categoria	**Ferramenta**	**Foco Principal**	**Ecossistema**	**Modelo de Custo**	**Flexibilidade / Customização**
Comercial	Celonis	Otimização e Automação (Execution Management)	Vasta gama de conectores (SAP, etc.)	Comercial	Menor (foco em soluções *out-of-the-box*)
	UiPath	Descoberta de oportunidades de automação (RPA)	Integrada na plataforma de automação UiPath	Comercial	Menor (orientada para o ecossistema UiPath)
	SAP Signavio	Gestão de Processos de Negócio (BPM)	Integração nativa e profunda com o ecossistema SAP	Comercial	Menor (otimizada para processos SAP)
Open-Source	ProM	Investigação e Algoritmos Académicos	Java, com centenas de *plug-ins*	Gratuito (*Open-Source*)	Muito Elevada (requer programação)

	PM4Py (a nossa escolha)	Análise de Dados e Soluções Customizadas	Python, integrada com Pandas, Scikit-learn, etc.	Gratuito (*Open-Source*)	Total (controlo completo sobre os dados e algoritmos)
	Disco	Análise Exploratória e Usabilidade	Aplicação de *Desktop* (Windows, Mac, Linux)	Comercial (Licença por utilizador)	Média (foco na análise, não na integração)
Integrada com Power BI	Power Automate PM	Análise de Processos Nativa da Microsoft	Integrada no ecossistema Microsoft Fabric	Comercial (requer "Fabric Capacity")	Baixa
	PAFnow	Visualização e Análise dentro do Power BI	*Custom visuals* e templates para Power BI	Comercial (Licenciamento)	Média (limitada às capacidades do Power BI)

Anexo B - Tabela FACT_EVENTS

Atributos	Propriedades
EventID	BIGINT IDENTITY(1,1) NOT NULL PRIMARY KEY
ProcessoID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_PROCESSO(ProcessoID)
CasoID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_CASO(CasoID)
AtividadeID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_ATIVIDADE(AtividadeID)
RecursoID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_RECURSO(RecursoID)
UnidadeOrganicaID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_UNIDADE_ORGANICA(UnidadeOrganicaID)
TimestampInicioID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_DATA(DataID)
TimestampInicio	DATETIME
TimestampLeitura	DATETIME
TimestampFim	DATETIME
DuracaoExecucaoHoras	FLOAT
TempoEsperaHoras	FLOAT
CustoExecucao	FLOAT
CustoEspera	FLOAT
CustoRH	FLOAT
CustoInfra	FLOAT
CustoSoftware	FLOAT
CustoEquipamento	FLOAT
CustoPorAtividade	FLOAT

Anexo C - Tabela DIM_PROCESSO

Atributos	Propriedades
ProcessoID	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
NomeProcessoOrigem	NVARCHAR(255) NOT NULL UNIQUE
NomeProcessoDisplay	NVARCHAR(255)
DataUltimoProcessamento	DATETIME

Anexo D - Tabela DIM_CASO

Atributos	Propriedades
CasoID	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
NomeCasoOrigem	NVARCHAR(255) NOT NULL
ProcessoID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_PROCESSO(ProcessoID)
VarianteID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_VARIANTE(VarianteID)
DuracaoCasoDias	FLOAT
CustoTotalCaso	FLOAT
ContagemEventos	INT

Anexo E - Tabela DIM_VARIANTE

Atributos	Propriedades
VarianteID	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
VarianteString	NVARCHAR(MAX) NOT NULL
VarianteHash	VARBINARY(32) NOT NULL UNIQUE
NumeroDeAtividadesUnicas	INT
FlagRework	BIT

Anexo F - Tabela DIM_ATIVIDADE

Atributos	Propriedades
AtividadeID	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
NomeAtividade	NVARCHAR(255) NOT NULL UNIQUE

Anexo G - Tabela DIM_RECURSO

Atributos	Propriedades
RecursoID	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
NomeRecurso	NVARCHAR(255) NOT NULL UNIQUE

Anexo H - Tabela DIM_UNIDADE_ORGANICA

Atributos	Propriedades
UnidadeOrganicaID	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
NomeUnidadeOrganica	NVARCHAR(500) NOT NULL UNIQUE

Anexo I - Tabela DIM_DATA

Atributos	Propriedades
DataID	INT NOT NULL PRIMARY KEY
DataCompleta	DATE NOT NULL UNIQUE
Ano	INT NOT NULL
NomeMes	NVARCHAR(50) NOT NULL
NomeDiaDaSemana	NVARCHAR(50) NOT NULL

Anexo J - Tabela ANALISE_PROCESSO_DFG_IMAGENS

Atributos	Propriedades
IDImagem	INT IDENTITY(1,1) NOT NULL PRIMARY KEY
ProcessoID	INT NOT NULL FOREIGN KEY REFERENCES dim.DIM_PROCESSO(ProcessoID)
TipoDFG	NVARCHAR(50)
ImagemURL	NVARCHAR(1024)
DataGeracao	DATETIME

Anexo K - Tabela comparativa de tempos de execução (ACI vs. local)

Fase do Pipeline	**Tempo em ACI (Cloud)**	**Tempo em Máquina Local**	**Análise da Diferença**
1. Extração de Dados (Staging)	~15 segundos	~35 segundos	ACI é significativamente mais rápido devido à baixa latência de rede para a base de dados Azure. A sobrecarga da ligação VPN no ambiente local é notória nesta fase de I/O intensivo.
2. Transformação (Pandas/PM4Py)	~110 segundos	~60 segundos	A máquina local é significativamente mais rápida. O acesso direto a um CPU de alto desempenho e a uma RAM mais rápida supera a performance da VM de uso geral do ACI.
3. Carga para o DW (Dimensões e Factos)	~25 segundos	~45 segundos	A proximidade da rede do ACI com a base de dados de destino resulta numa escrita de dados mais rápida, mesmo com a lógica complexa de

			DELETE e INSERT.
Total de Execução	~2 minutos e 30 segundos	~2 minutos e 20 segundos	O tempo total é notavelmente semelhante, mas a performance superior da máquina local na fase de transformação compensa a sua maior latência de rede.