

JIRA Data Engineering Pipeline – Medallion Architecture

Este projeto implementa um pipeline de Engenharia de Dados robusto para processar dados do JIRA, utilizando a Arquitetura Medallion para transformar dados brutos em inteligência de negócio sobre SLA (Service Level Agreement).

BR Versão em Português

Objetivo

O objetivo deste projeto é automatizar a ingestão e o processamento de chamados do JIRA para calcular o tempo de resolução em **horas úteis**, desconsiderando finais de semana e feriados nacionais.

Arquitetura do Pipeline

O projeto segue a **Arquitetura Medallion**, garantindo organização e rastreabilidade:

- **Camada Bronze:** Ingestão de arquivos JSON brutos do Azure Blob Storage utilizando autenticação via Service Principal.
- **Camada Silver:** Limpeza e normalização dos dados. Extração de campos aninhados (**assignee**, **timestamps**) e conversão para o formato **Parquet** visando performance e preservação de metadados.
- **Camada Gold:** Aplicação de regras de negócio. Cálculo de SLA baseado na prioridade (High: 24h, Medium: 72h, Low: 120h), integrando com a **BrasilAPI** para identificação de feriados.
- **Data Quality:** Auditoria automática de integridade, volumetria e validação de nulos ao final do processo.

Dashboard e Visualização (Streamlit)

O projeto inclui uma interface interativa para gestão de performance, permitindo:

- **Filtros Dinâmicos:** Seleção por Período, Tipo de Chamado e Analista.
- **Análise de Tendência:** Gráfico mensal de conformidade de SLA.
- **Ranking de Performance:** Classificação automática de analistas com status de suporte (Top Performer, Standard, Needs Support).

Tecnologias e Boas Práticas

- **Python 3.x** e **Pandas** para manipulação de dados.
- **Streamlit & Plotly** para visualização e dashboards interativos.
- **PyArrow:** Engine de escrita Parquet estável.
- **Segurança:** Uso de variáveis de ambiente (**.env**) e proteção de dados sensíveis via **.gitignore**.

Evidências de Execução e Qualidade

O pipeline conta com um orquestrador central que valida cada etapa. Em execuções de teste:

- **Funil de Dados:** Ingestão de 1000 registros → 990 registros válidos → 804 chamados finalizados.
- **Auditoria:** Validação de 100% das regras de prioridade e integridade cronológica.

Como Executar

1. **Clone o repositório:** `git clone https://github.com/alexlourenc/engenharia_dados.git`
2. **Configure o Ambiente Virtual:** `python -m venv venv` e ative-o (`.\venv\Scripts\activate` no Windows ou `source venv/bin/activate` no Linux/Mac).
3. **Instale as dependências:** `pip install -r requirements.txt`
4. **Configure o arquivo .env** na raiz do projeto com suas credenciais.
5. **Execute o orquestrador:** `python main.py`
6. **Inicie o dashboard:** `streamlit run app.py`

us English Version

Objective

This project automates the ingestion and processing of JIRA tickets to calculate resolution time in **business hours**, excluding weekends and national holidays.

Pipeline Architecture

The project follows the **Medallion Architecture**:

- **Bronze Layer:** Raw ingestion from Azure Blob Storage using Service Principal authentication.
- **Silver Layer:** Data cleaning, normalization, and conversion to **Parquet** format for performance.
- **Gold Layer:** Business rules application and SLA calculation based on priority, integrated with **BrasilAPI**.
- **Data Quality:** Automated auditing of integrity, volume, and null validation.

Dashboard & Visualization (Streamlit)

Interactive interface for performance management:

- **Dynamic Filters:** Date Range, Issue Type, and Analyst.
- **Trend Analysis:** Monthly SLA compliance tracking.
- **Performance Ranking:** Automated analyst classification (Top Performer, Standard, Needs Support).

Technologies & Best Practices

- **Python 3.x** and **Pandas** for data manipulation.
- **Streamlit & Plotly** for visualization and interactive dashboards.
- **PyArrow:** Stable Parquet write engine.
- **Security:** Use of environment variables (`.env`) and sensitive data protection via `.gitignore`.

Execution Evidence & Quality

The pipeline features a central orchestrator that validates each stage. In test runs:

- **Data Funnel:** 1000 raw records → 990 valid records → 804 finalized tickets.
- **Auditing:** 100% validation of priority rules and chronological integrity.

How to Run

1. **Clone the repository:** `git clone https://github.com/alexlourenc/engenharia_dados.git`
2. **Setup Virtual Environment:** `python -m venv venv` and activate it.
3. **Install dependencies:** `pip install -r requirements.txt`
4. **Configure the .env file** in the project root with your credentials.
5. **Run the orchestrator:** `python main.py`
6. **Launch the dashboard:** `streamlit run app.py`

Estrutura de Pastas / Project Structure

```
PROJECT-ROOT/
├── data/           # Armazenamento Local / Local Storage (Ignorado no Git)
├── src/            # Código-fonte principal / Source Code
│   ├── bronze/    # Ingestão (JSON Raw) / Ingestion
│   ├── silver/    # Limpeza e Parquet / Cleaning & Parquet
│   ├── gold/      # Regras de Negócio / Business Rules
│   ├── sla_calculation.py # Motor de Cálculo / Calculation Engine
│   └── validate_pipeline.py # Auditoria de Dados / Data Auditing
├── .env            # Credenciais e Variáveis de ambiente (Sensitive data)
├── .gitignore      # Proteção de arquivos sensíveis
├── .python-version # Versão fixa do Python (3.12)
├── app.py          # Interface do Dashboard (Streamlit)
├── main.py         # Orquestrador Central / Main Orchestrator
├── README.md       # Documentação do projeto
└── requirements.txt # Dependências do projeto
```