**Documentação API**

### 1.0 Visão Geral e Objetivos

Este documento detalha a arquitetura e implementação de uma pipeline de dados completa, robusta e automatizada. O objetivo principal do projeto é extrair, transformar e carregar (ETL) dados públicos do Operador Nacional do Sistema Elétrico (ONS) para um ambiente de nuvem, disponibilizando-os de forma estruturada e otimizada para análises e aplicações de Machine Learning.

A solução foi projetada utilizando práticas modernas de engenharia de dados e DevOps, incluindo uma arquitetura de microsserviços, infraestrutura como código (IaC) e um ciclo de integração e implantação contínuas (CI/CD).

### 2.0 Arquitetura da Solução

A arquitetura foi implementada na Google Cloud Platform (GCP) e é composta pelos seguintes fluxos e componentes principais:

1. Fonte de Dados: A pipeline extrai dados de duas fontes principais do portal de dados abertos da ONS: "ENA (Energia Natural Afluente) Diária por Bacia" e "Reservatórios". Os dados são consumidos no formato Parquet.
2. API de Orquestração (Cloud Run): Uma API RESTful, desenvolvida em Python com o framework FastAPI, serve como o núcleo da pipeline. Ela é executada em um container Docker no Cloud Run, um ambiente serverless que garante escalabilidade e baixo custo. A API expõe endpoints para iniciar a ingestão de dados e para consultar os dados já processados.
3. Data Lake (Cloud Storage): Os dados processados pela API são armazenados no Cloud Storage. Os dados de ENA (fatos) são salvos de forma particionada no estilo Hive (ano=/mes=/dia=/) para otimizar consultas, enquanto os dados de Reservatórios (dimensão) são salvos em um caminho separado. O formato de armazenamento padrão é o Parquet.
4. Data Warehouse (BigQuery): Para a camada de análise, foram utilizadas Tabelas Externas no BigQuery. Essas tabelas funcionam como "ponteiros" para os dados no Cloud Storage, permitindo que sejam consultados com SQL de alta performance sem a necessidade de duplicar o armazenamento. O BigQuery reconhece automaticamente as partições dos dados de ENA, garantindo que as consultas sejam rápidas e econômicas.
5. Automação de Ingestão (Cloud Scheduler e Cloud Functions): Para a atualização contínua dos dados, um job no Cloud Scheduler é executado diariamente. Ele aciona uma Cloud Function leve, cujo único propósito é chamar o endpoint de ingestão da API principal com a data do dia anterior, garantindo que a pipeline seja autônoma.
6. Automação de Implantação (CI/CD com GitHub Actions): O projeto é totalmente integrado com o GitHub Actions. Qualquer alteração no código-fonte (push para a branch main) aciona um workflow que automaticamente:
   * Constrói e publica a imagem Docker da API no Artifact Registry.
   * Empacota e envia o código da Cloud Function para o Cloud Storage.
   * Executa o Terraform para provisionar e/ou atualizar toda a infraestrutura na GCP de forma segura e declarativa.

### 3.0 Stack de Tecnologias

* Linguagem de Programação: Python 3.10
* Framework da API: FastAPI
* Manipulação de Dados: Pandas
* Plataforma de Nuvem: Google Cloud Platform (GCP)
  + Computação: Cloud Run, Cloud Functions
  + Armazenamento e Dados: Cloud Storage, BigQuery, Artifact Registry
  + Automação: Cloud Scheduler
  + Segurança: IAM (Identity and Access Management)
* Infraestrutura como Código (IaC): Terraform
* Containerização: Docker
* CI/CD: GitHub Actions

### 4.0 Guia de Instalação e Execução Local

Para configurar e executar a API em um ambiente de desenvolvimento local:

1. Pré-requisitos: Ter o Python 3.10+, o [Google Cloud SDK](https://cloud.google.com/sdk/docs/install) e o Docker instalados.
2. Clonar o Repositório: git clone <URL\_DO\_REPOSITORIO>
3. Ambiente Virtual:
   * Criar: python3 -m venv .venv
   * Ativar: source .venv/bin/activate
4. Instalar Dependências: pip install -r api/requirements.txt
5. Autenticação: Autenticar a máquina local com o GCP: gcloud auth application-default login
6. Variáveis de Ambiente: Criar um arquivo .env na raiz do projeto com as chaves e valores dos recursos na nuvem (ID do projeto, nome do bucket, etc.).
7. Executar a API: uvicorn api.main:app --reload. A API estará disponível em http://127.0.0.1:8000.

### 5.0 Detalhamento da API (Endpoints)

A API expõe dois endpoints principais, com documentação interativa (Swagger UI) disponível na rota /docs.

#### 5.1 Endpoint de Ingestão (POST /ingest)

* Função: Inicia o processo de ETL para um período de datas especificado. Baixa os dados de ENA e Reservatórios da ONS, os processa e salva no Cloud Storage.
* Método: POST
* Path: /ingest

Corpo da Requisição: application/json  
JSON  
{

"start\_date": "2024-01-01",

"end\_date": "2024-01-31"

}

Resposta de Sucesso (200 OK):  
JSON  
{

"status": "success",

"message": "Ingestion process completed.",

"records\_saved": 5890

}

#### 5.2 Endpoint de Consulta (GET /data)

* Função: Consulta os dados já processados no BigQuery, realizando um JOIN entre os dados de ENA e os de Reservatórios. Os resultados são retornados de forma paginada.
* Método: GET
* Path: /data
* Parâmetros da Query:
  + page: (inteiro) O número da página. Padrão: 1.
  + size: (inteiro) A quantidade de itens por página. Padrão: 50.
* Exemplo de Chamada: /data?page=1&size=10

Resposta de Sucesso (200 OK):  
JSON  
{

"total\_records": 150320,

"total\_pages": 15032,

"current\_page": 1,

"page\_size": 10,

"data": [

{

"nom\_bacia": "PARANA",

"ena\_data": "2024-09-24",

"ena\_bruta\_bacia\_mwmed": 1350.75,

"nom\_reservatorio": "ITAIPU"

}

]

}

### 6.0 Infraestrutura e Conclusão

A infraestrutura do projeto é totalmente gerenciada pelos arquivos Terraform localizados na pasta infra/, garantindo a reprodutibilidade e o controle de versão do ambiente de nuvem. A automação com GitHub Actions (definida em .github/workflows/) assegura que qualquer alteração no código seja implantada de forma segura e consistente, sem intervenção manual.

Este projeto demonstra com sucesso a construção de uma pipeline de dados moderna, escalável e automatizada, pronta para suportar análises de dados complexas e aplicações de Machine Learning.