TECH CHALLENGE - MACHINE LEARNING ENGINEERING - FASE 1

Gustavo Niewerth - RM 366500
Michel de Oliveira Hilgemberg - RM365928

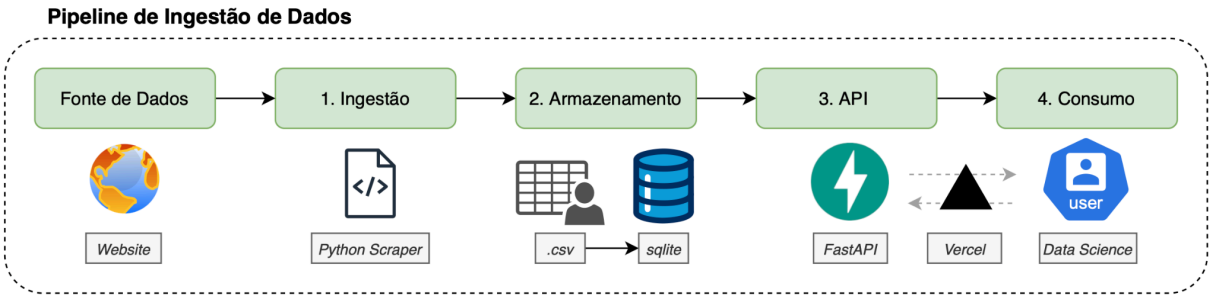
PLANO ARQUITETURAL

Este documento detalha a arquitetura do projeto, o pipeline de dados, os planos de escalabilidade, cenários potenciais de uso e um plano de integração com modelos de Machine Learning, conforme os requisitos do Tech Challenge.

1. PIPELINE DE DADOS

O fluxo de dados do projeto foi desenhado para ser simples, modular e eficaz, seguindo quatro etapas distintas:

FIGURA 1 - PIPELINE DE DADOS



FONTE: OS AUTORES

Ingestão: A ingestão de dados é realizada através de um script Python para web scraping. Utilizando principalmente a biblioteca `BeautifulSoup4` para fazer a varredura (scraping) do site público `books.toscrape.com`. O Script navega por todas as páginas de livros para extrair um conjunto predefinido de atributos de cada produto cadastrado. Sendo esses atributos:

Processamento e Armazenamento: O scraper salva os dados extraídos em um formato simples e universal, o CSV (`books.csv`). Em seguida, ao iniciar a API, um processo automatizado é acionado criando um banco de dados SQLite (`data.db`) e suas tabelas, caso não existam. A API terá acesso direto ao banco de dados, tornando a aplicação mais robusta e possibilitando boas práticas de desenvolvimento como *data models* e *schemas*.

API (Aplicação): Uma API RESTful, construída com FastAPI, serve como a interface principal para os dados. Ela se conecta diretamente ao banco de dados SQLite para realizar consultas, aplicar lógicas de negócio e retornar os dados em formato JSON para os clientes.

Deploy e Acesso Público (Vercel): Para tornar a API publicamente acessível, o projeto é integrado com a plataforma Vercel. Através de um processo de CI/CD (Integração Contínua/Entrega Contínua) conectado ao repositório do GitHub, cada atualização no código principal dispara um deploy automático. A Vercel gerencia a execução da aplicação FastAPI

em um ambiente *serverless*, disponibilizando-a globalmente através de uma URL pública e cuidando de toda a infraestrutura de hospedagem.

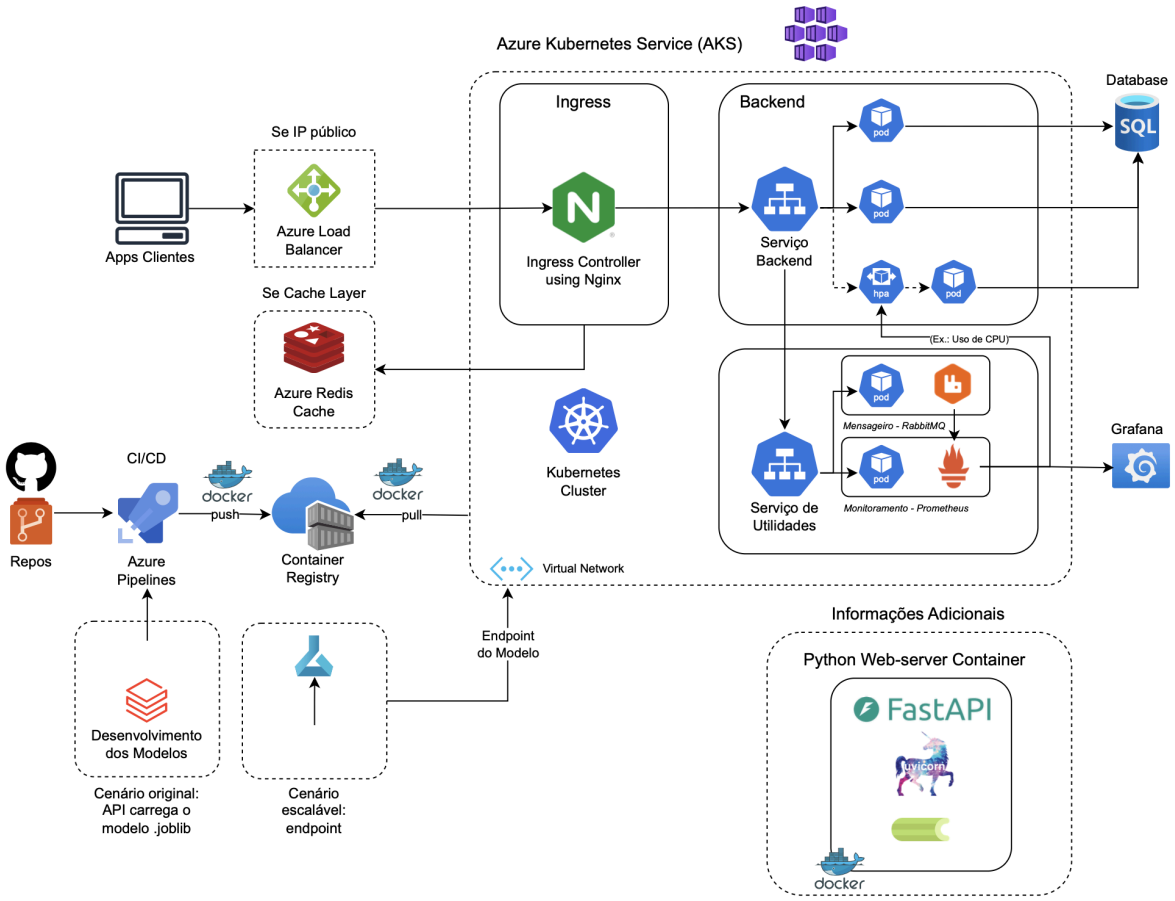
- URL (Docs): https://fiap-ml-tech-challenge-1-nine.vercel.app/docs

Consumo: A partir da URL pública fornecida pela Vercel, desenvolvedores podem consumir a API para construir aplicações web e mobile, como um catálogo de livraria online. Da mesma forma, cientistas de dados podem carregar os dados para análise, visualização e treinamento de modelos de Machine Learning, demonstrando a grande versatilidade do pipeline.

2. ARQUITETURA PARA ESCALABILIDADE FUTURA

Pensando em escalabilidade futura, a arquitetura atual pode evoluir para um sistema de microsserviços em ambiente nuvem. A FIGURA 2 abaixo retrata essa possibilidade utilizando os serviços da Microsoft Azure:

FIGURA 2 - PIPELINE DE DADOS



FONTE: OS AUTORES

A seguir, são descritos os principais componentes dessa arquitetura bem como suas funções para viabilizar escalabilidade:

Orquestração de Contêineres com AKS: O coração da arquitetura é o Azure Kubernetes Service (AKS). A aplicação FastAPI é empacotada em um container Docker e implantada no

cluster AKS. O Kubernetes orquestra a execução desses contêineres permitindo o autoescalonamento. O tráfego interno é gerenciado por um Ingress Controller (Nginx), que roteia as requisições para os serviços corretos dentro do cluster.

Entrada e Balanceamento de Carga: As requisições dos "Apps Clientes" são direcionadas para um IP Público que distribui o tráfego de entrada de forma eficiente entre as instâncias da aplicação.

CI/CD e Automação de Deploy: Um pipeline de CI/CD com Azure Pipelines automatiza todo o processo de deploy. Qualquer alteração no código-fonte no repositório dispara o pipeline, que automaticamente constrói a nova imagem Docker tornando as entregas mais rápidas e seguras.

Banco de Dados e Cache: O banco de dados SQLite local é substituído por uma solução de banco de dados gerenciado e escalável, como o Azure SQL Database ou Azure Database for PostgreSQL. Para otimizar a performance e reduzir a carga no banco, uma camada de cache é implementada com Azure Redis Cache.

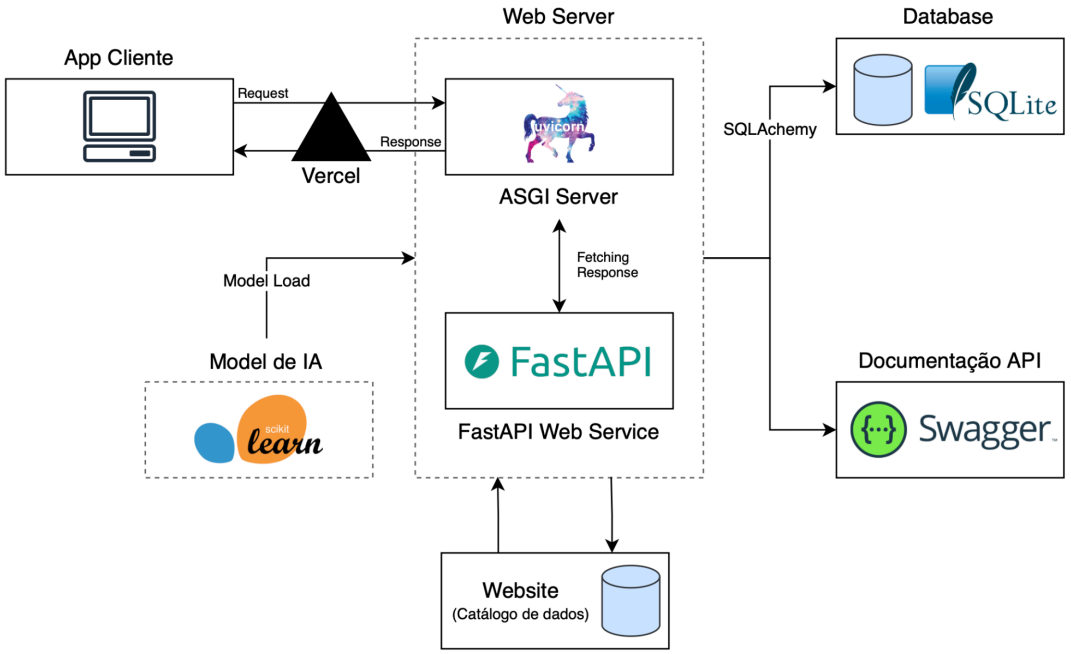
Monitoramento e Serviços de Utilidades: Dentro do AKS, um conjunto de "Serviços de Utilidades" é implantado para garantir a observância e a resiliência do sistema. O Prometheus coleta métricas de performance de todos os serviços, que são visualizadas em dashboards no Grafana. Um mensageiro como o RabbitMQ pode ser usado para processar tarefas de forma assíncrona, desacoplando componentes e melhorando a resposta do sistema.

Escalabilidade de Modelos de ML com MLflow: Uma solução mais próxima do cenário original do Tech Challenge é o de carregar um arquivo `.joblib` diretamente na API. Como um avanço de eficiência, pode-se construir um cenário em que o modelo é servido através do MLflow, ferramenta que expõe um Endpoint do Modelo dedicado. Esse endpoint é conteinerizado e implantado no AKS. Essa abordagem desacopla o ciclo de vida do modelo do ciclo de vida da API, permitindo que os modelos sejam atualizados, versionados e escalados de forma independente.

3. CENÁRIO DE USO PARA CIENTISTAS DE DADOS/ML

Para ilustrar possíveis cenários de uso da solução por um Cientista de Dados, adotou-se uma visão simplificada da arquitetura de referência conforme FIGURA 3 abaixo:

FIGURA 3 - CENÁRIO DE USO



FONTE: OS AUTORES

Essa abordagem permite um fluxo de trabalho direto e eficiente, permitindo dois cenários claros de utilização para um cientista de dados:

Aquisição de Dados para Análise: O cientista, a partir de seu "App Cliente" (como um Jupyter Notebook), envia uma `JSON Request` para um dos endpoints da API (ex: `GET /api/v1/ml/training-data`). A aplicação FastAPI retorna uma `JSON Response` com os dados solicitados. Esses dados podem ser facilmente utilizados para análise exploratória, visualização e preparação para o treinamento de modelos.

Treinamento e Integração de Modelos: Após adquirir e processar os dados, o cientista treina um modelo para uma tarefa específica, como prever o preço de um livro. Em um cenário de uso simplificado, a própria aplicação FastAPI pode ser configurada para carregar o modelo treinado (ex: um arquivo `.joblib`) diretamente na memória durante sua inicialização. Isso permite a real utilização dos novos endpoints preditivos (ex: `POST /api/v1/ml/predictions`) que utilizam o modelo para fazer inferências em tempo real. No cenário atual de desenvolvimento da API esse endpoint retorna dados de simulação baseados apenas em alguns condicionais base.

4. PLANO DE INTEGRAÇÃO COM MODELOS DE ML

O plano de integração de modelos de ML à API prevê uma evolução em duas abordagens principais, permitindo desde uma implementação simples até uma arquitetura escalável para um momento futuro:

Fase 1: Integração Direta via Arquivo Serializado

- **Arquitetura:** Cenário mais simples como descrito acima no tópico 3, onde após o treinamento o modelo de IA é salvo em um arquivo serializado, como `model.joblib`. A aplicação FastAPI é então configurada para carregar este arquivo em memória durante o seu processo de inicialização.
- **Implementação:** Implementação é baseado no uso em produção do endpoint `POST api/v1/ml/predictions`. Este endpoint receberia os dados de entrada necessários para a predição, os processaria e os passaria para o modelo carregado, retornando o resultado da inferência.
- **Observações:** Simplicidade de implementação para projetos de menor escala. Porém, qualquer atualização no modelo exige um novo deploy da API inteira.

Fase 2: Servindo o Modelo como um Endpoint Dedicado com MLflow

- **Arquitetura:** Esta é a abordagem escalável e recomendada para ambientes de produção. O ciclo de vida do modelo é gerenciado pela plataforma MLflow. Após o treinamento, o modelo é registrado no MLflow, que por sua vez o serve como uma API independente e dedicada.
- **Implementação:** A API principal de livros não carrega mais o modelo, em vez disso atua como um orquestrador. A API recebe a requisição, formata adequadamente e faz uma chamada HTTP para o endpoint do modelo servido pelo MLflow.
- **Observações:** Essa abordagem exige maior complexidade inicial de configuração, porém ao tratar a API de livros e o modelo de ML como serviços independentes facilita o versionamento, registro e monitoramento dos modelos, facilitando a governança.