Documentação Técnica - Sistema de Processamento de Imagens

Visão Geral do Sistema

O sistema foi desenvolvido para processar imagens de forma escalável , como estudado na matéria de computação gráfica, a ideia inicial era conseguir aplicar filtros ou alterações de cores especificas, mas priorizei a parte de balanceamento funcional, utilizando processamento paralelo através de threads, containerização com Docker, e orquestração com Kubernetes. O objetivo principal é permitir o upload de imagens, processá-las de forma paralela e disponibilizá-las para download após o processamento.

Processamento Paralelo com Threads

O sistema utiliza threads para processar múltiplas imagens simultaneamente. A implementação está centralizada na classe ImageProcessor.

Threads

O sistema mantém um conjunto fixo de threads (por padrão, 4 threads) que ficam aguardando por trabalho. Cada thread é responsável por:

1. Pegar uma imagem da fila de processamento
2. Processar a imagem usando a biblioteca Pillow
3. Salvar o resultado no banco de dados
4. Retornar à fila para pegar mais trabalho

Fila de Processamento

Foi criada uma fila thread-safe usando queue.Queue() do Python. A fila:

1. Armazena as imagens aguardando processamento
2. Garante que cada imagem seja processada apenas uma vez
3. Permite que múltiplas threads consumam imagens de forma segura
4. Mantém o balanceamento de carga entre as threads

Containerização com Docker

Containers do Sistema

O sistema utiliza dois containers principais:

1 - Container da Aplicação (image-processor)

- Base: Python 3.9-slim
- Responsabilidades:
  - Executar a API Flask
  - Processar imagens
  - Gerenciar threads
  - Servir a interface web
- Dependências principais:
  - Flask para API web
  - Pillow para processamento de imagens
  - psycopg2 para conexão com PostgreSQL

2 - Container do Banco de Dados (postgres-db)

- Base: PostgreSQL 13
- Responsabilidades:
  - Armazenar metadados das imagens

- ▼ Armazenar imagens originais e processadas
- ▼ Manter estado do sistema

Arquitetura e Comunicação

Nossa arquitetura é baseada em microserviços, onde cada componente opera de forma isolada mas integrada. Temos dois serviços principais: o processador de imagens e o banco de dados PostgreSQL.
A comunicação entre estes serviços é gerenciada pelo Kubernetes, que estabelece uma rede interna isolada. O processador de imagens se comunica com o banco de dados através de um serviço denominado "postgres", que atua como um proxy interno. Esta comunicação é gerenciada através de credenciais armazenadas em Secrets do Kubernetes, garantindo que informações “sensíveis” nunca sejam expostas no código ou nos containers.

Orquestração e Gerenciamento

O Kubernetes atua como nosso orquestrador, gerenciando todos os aspectos operacionais do sistema. Utilizamos três componentes principais:

Deployments

O processador de imagens é gerenciado através de Deployments, que nos permite controlar precisamente como a aplicação é executada e escalada. Este mecanismo garante que sempre tenhamos o número desejado de réplicas em execução, mesmo em caso de falhas. Quando uma atualização é necessária, o Deployment gerencia o processo de forma gradual, garantindo zero downtime.

StatefulSet

Para o PostgreSQL, temos a opção por um StatefulSet devido à natureza stateful do banco de dados. Este componente garante que os dados persistam mesmo após reinicializações e que cada instância do banco mantenha sua identidade única. O StatefulSet gerencia volumes persistentes automaticamente, garantindo que os dados sejam preservados de forma segura.

Serviços

O sistema utiliza dois tipos de serviços: um LoadBalancer para a aplicação web, que permite acesso externo de forma balanceada, e um ClusterIP para o PostgreSQL, que mantém o banco de dados acessível apenas internamente no cluster.

Banco de Dados PostgreSQL

Estrutura do Banco

Tabela principal: processed_images

```
CREATE TABLE processed_images (
    id UUID PRIMARY KEY,
    original_data BYTEA,
    processed_data BYTEA,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed_at TIMESTAMP,
    status VARCHAR(20)
);
```

Campos:

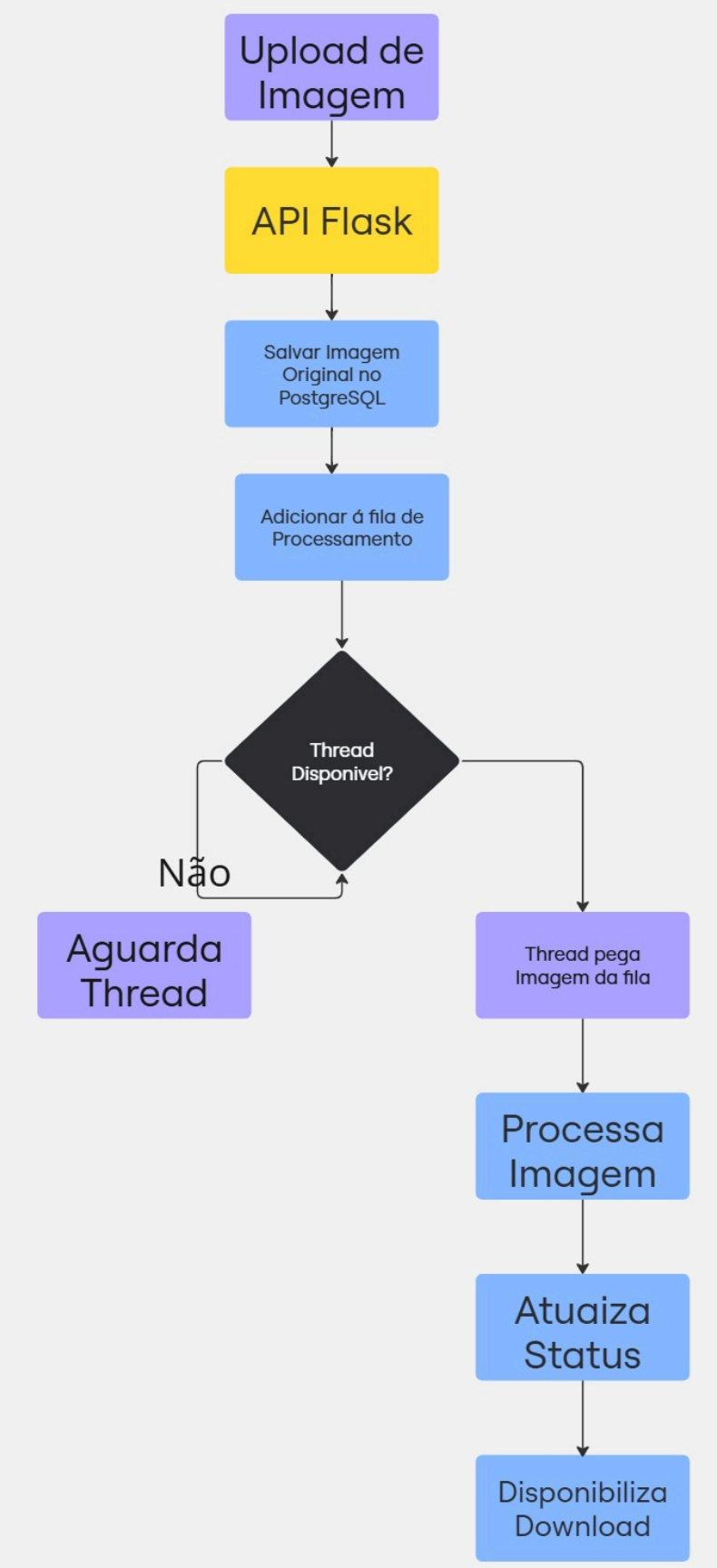
- id: Identificador único da imagem
- original_data: Dados binários da imagem original
- processed_data: Dados binários da imagem processada
- uploaded_at: Timestamp do upload
- processed_at: Timestamp do processamento
- status: Estado atual (processing/completed)

Interface Web

Componentes da Interface

Para a criação a interface foi utilizado icones e tabelas prontas, extraindo o template html do site do MUI (MUI: The React component library you always wanted) pacotes prontos e fácil de usar para deixar o sistemais mais interativo possível.

Fluxograma do Sistema:


Upload de Imagem
API Flask
Salvar Imagem Original no PostgreSQL
Adicionar á fila de Processamento
Thread Disponivel?
Não
Aguarda Thread
Thread pega Imagem da fila
Processa Imagem
Atuaiza Status
Disponibiliza Download

Limitações e Possíveis Melhorias

Processamento de Imagens

- O sistema está limitado apenas ao redimensionamento básico de imagens
- Não há suporte para diferentes formatos de imagem além de JPEG
- Não há validação do tipo ou tamanho de arquivo durante o upload

Para realização de testes, utilizei o Postman para testar todos os endpoints (upload, status, download).