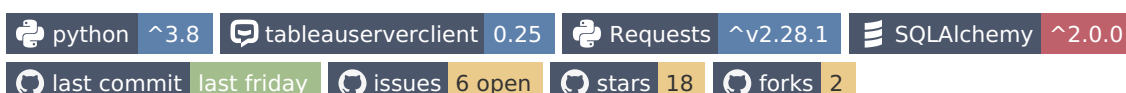


# Painel Streamlit para visualizar, analisar e prever dados de vendas de um restaurante

## Projeto Pedacinho do Céu



Este painel foi desenvolvido usando Streamlit. Vários pacotes de código aberto são usados para processar os dados e gerar as visualizações, por exemplo. [pandas](#), [geopandas](#), [leafmap](#), [matplotlib](#) e [pydeck](#).

Este é um projeto de análise de dados para um bar e restaurante localizado no sul da ilha de Florianópolis. O objetivo do projeto é utilizar as informações disponíveis para realizar análises e obter insights que possam ser utilizados para tomar decisões estratégicas.

**Nota:** Resultados de gráficos e tabelas no power bi, estão disponíveis na pasta data/processed.

Power BI

## Índice

1. [🚀 Introdução](#)
2. [🌳 Arvore de Diretorios](#)
3. [🏠 Arquitetura do projeto](#)
4. [🎆 Tecnologias Utilizadas](#)
5. [💎 Pré-requisitos](#)
6. [✨ Instalando o Projeto](#)
7. [🎉 Executando o Projeto](#)
8. [🛠 Configurando o ambiente virtual](#)
9. [🔄 Continuando a instalação](#)
10. [🧪 Testes](#)
11. [🔥 Utilizando a API com Insomnia](#)
12. [🧠 Versão atual](#)
13. [🦊 Coletando Dados](#)
14. [🤖 Processando Dados](#)
15. [🖥 Visualizando os Dados](#)

16.  [Futuras Atualizações](#)
17.  [Tecnologias e conceitos utilizados](#)
  - [Hadoop](#)
  - [Kafka](#)
  - [Docker](#)
  - [SOLID](#)
  - [Padrões de commit](#)
18.  [Roadmap v1](#)
  - [Travis CI](#)
  - [Princípios SOLID](#)
  - [SQLAlchemy com PostgreSQL ou MySQL](#)
  - [Autenticação e autorização](#)
  - [Interface de usuário](#)
19.  [Roadmap v2](#)
  - [Roadmap v1.1](#)
  - [Roadmap v1.2](#)
  - [Roadmap v1.3](#)
  - [Roadmap v2.0](#)
20.  [O que é o Apache Spark?](#)
21.  [Critérios de aceitação do projeto](#)
22.  [Contribuindo](#)
23.  [Contribuidores](#)
24.  [Licença](#)

## Introdução

Este projeto é um estudo de caso de Big Data e Power BI. O objetivo é demonstrar como coletar, processar, analisar e visualizar grandes volumes de dados em tempo real usando ferramentas e tecnologias como Python, SQLite e Power BI.

O projeto é baseado em um restaurante chamado "Pedacinho do Céu". O restaurante está localizado em uma cidade turística e serve comida tradicional da região. O restaurante está interessado em coletar dados de várias fontes para analisar e obter insights sobre o negócio. O restaurante também deseja criar visualizações e relatórios interativos para ajudar na tomada de decisões e na compreensão de tendências e padrões nos dados.

O mesmo utiliza a linguagem Python e diversas bibliotecas para análise de dados, como Pandas, NumPy e Plotly. Além disso, é utilizado o framework Streamlit para a criação de uma interface interativa para o usuário, permitindo a visualização dos dados e a interação com as funcionalidades desenvolvidas.

Entre as funcionalidades desenvolvidas, estão a análise de vendas por mês, a previsão de clientes para o próximo mês, a análise de dados de clientes

cadastrados, a exibição de um mapa de localização do estabelecimento e a criação de uma história do bar e restaurante.

Para utilizar o projeto, basta clonar o repositório e instalar as dependências listadas no arquivo requirements.txt. Em seguida, execute o comando "streamlit run app.py" para iniciar a aplicação.

1. Coleta de dados: O projeto utiliza diversas fontes de dados, como arquivos CSV, APIs e bancos de dados. Os dados são coletados e processados em uma variedade de formatos para atender às necessidades do projeto. A coleta de dados é um processo crucial para garantir que as informações corretas estejam disponíveis para análise e visualização.
2. Processamento de dados: O projeto utiliza uma variedade de técnicas para processar e limpar os dados coletados, incluindo o uso de bibliotecas Python para análise e transformação de dados. Os dados são organizados e limpos para garantir que estejam prontos para análise e visualização. O processamento de dados é uma etapa importante no processo de garantir que as informações corretas estejam disponíveis para o usuário final.
3. Armazenamento de dados: O projeto utiliza uma variedade de tecnologias de armazenamento de dados, incluindo bancos de dados relacionais e não relacionais, armazenamento em nuvem e arquivos CSV. Os dados são armazenados de forma a garantir que estejam seguros e disponíveis para análise e visualização. O armazenamento de dados é um componente crítico do projeto, garantindo que os dados estejam disponíveis quando necessários.
4. Análise e visualização de dados: O projeto utiliza ferramentas de análise e visualização de dados, como Power BI e bibliotecas Python, para extrair informações significativas dos dados coletados. As informações são apresentadas de forma clara e intuitiva, permitindo que o usuário final compreenda facilmente os insights obtidos a partir dos dados. A análise e visualização de dados são etapas críticas no processo de transformação de dados em informações úteis.
5. Atualização e manutenção: O projeto requer atualizações regulares para garantir que os dados estejam atualizados e precisos. Além disso, a manutenção do sistema é importante para garantir que as tecnologias utilizadas estejam atualizadas e seguras. A atualização e manutenção do sistema são etapas críticas para garantir que o projeto continue funcionando de forma eficiente e eficaz ao longo do tempo.

O projeto utiliza conceitos de Big Data e Power BI. Vamos revisar como cada tecnologia é aplicada no projeto:

1. Linguagem de programação Python para desenvolvimento do back-end e front-end da aplicação web, utilizando o framework Streamlit para criação da interface do usuário.
2. Pandas e Numpy para manipulação e análise de dados, realizando operações como seleção, filtragem, agrupamento e junção de dados.
3. Matplotlib e Plotly para criação de gráficos e visualizações de dados interativas.
4. Scikit-Learn para modelagem de dados, com algoritmos de aprendizado de máquina para previsão e classificação.
5. Power BI para criação de dashboards e relatórios interativos e visuais a partir dos dados gerados e analisados na aplicação web.

Combinando Big Data e Power BI, este projeto oferece uma solução completa para coletar, processar, analisar e visualizar grandes volumes de dados em tempo real, ajudando na tomada de decisões informadas e oferecendo insights valiosos para o negócio do restaurante "Pedacinho do Céu".

## Arvore de Diretorios

Abaixo está a estrutura de diretórios do projeto:

```
bash . |─ 📁 .github |─ 📁 .husky |─ 📁 .vscode |─ 📁 assets
|─ 📁 backend |   |─ 📁 admin |   |─ 📁 conf |   |─ 📁
data_crawlers |   |─ 📁 target_url_crawlers |   |─ .gitignore |   |─
docker-compose.yml |   |─ Dockerfile |   |─ go_spider.py |   |─
README.md |   |─ requirements.txt |   |─ 📁 build |   |─ 📁 client |   |─
📁 docs |   |─ 📁 myenv |   |─ 📁 docs |   |─ 📁 src |   |─ 📁 api |   |─
📁 data |   |─ 📁 error |   |─ 📁 log |   |─ 📁 public |   |─ 📁
scripts |   |─ main.py |   |─ .dockerignore |   |─ .editorconfig |   |─
.env |   |─ .gitignore |   |─ .npmrc |   |─ .travis.yml |   |─ app.py |   |─
architecture.md |   |─ AUTHORS.md |   |─ CHANGELOG.md |   |─ docker-
compose.yml |   |─ Dockerfile |   |─ LICENSE |   |─ Makefile |   |─ package-
lock.json |   |─ package.json |   |─ Procfile |   |─ pyproject.toml |   |─
README.md |   |─ requirements.txt |   |─ runtime.txt |   |─ SECURITY.md
|   |─ setup.sh
```

## Arquitetura do projeto

A arquitetura do projeto é dividida em várias partes:

- **.github:** diretório que contém arquivos relacionados à integração contínua com o GitHub.

- **.husky**: diretório que contém arquivos relacionados à configuração do Husky, ferramenta que permite a execução de scripts no Git Hooks.
- **admin**: diretório que contém arquivos relacionados à administração do projeto, como scripts para extração de dados (data\_crawlers) e URLs alvo (target\_url\_crawlers).
- **docs**: diretório que contém arquivos relacionados à documentação do projeto.
- **src**: diretório que contém o código fonte do projeto, organizado em diferentes subdiretórios, como api (que contém as rotas da aplicação), data (que contém os arquivos de dados), error (que contém o tratamento de erros), log (que contém os arquivos de logs) e public (que contém arquivos estáticos, como imagens).
- **app.py**: arquivo que contém a configuração e inicialização da aplicação Flask.
- **docker-compose.yml**: arquivo que contém a configuração do Docker Compose para a execução da aplicação e do banco de dados.
- **Dockerfile**: arquivo que contém a configuração do Docker para a construção da imagem da aplicação.
- **Makefile**: arquivo que contém os comandos de automatização de tarefas do projeto.
- **README.md**: arquivo que contém a descrição do projeto e sua documentação.
- **requirements.txt**: arquivo que contém as dependências do projeto.
- **LICENSE**: arquivo que contém as informações sobre a licença do projeto.

## Tecnologias Utilizadas

Neste projeto "Pedacinho do Céu", diversas tecnologias são utilizadas para coletar, processar, armazenar e visualizar dados. Abaixo está uma lista dessas tecnologias e como elas se encaixam no projeto:

1. **Docker Compose**: Ferramenta para definir e gerenciar aplicações multi-container usando arquivos de configuração (docker-compose.yml). É usado para simplificar o processo de inicialização e gerenciamento de todos os serviços envolvidos no projeto.
2. **Power BI**: Ferramenta de Business Intelligence (BI) da Microsoft para criar relatórios e visualizações de dados. É usado para analisar e visualizar os dados coletados e processados pelo projeto.
3. **Flask** (opcional): Microframework Python para desenvolvimento de aplicações web. Pode ser usado para criar uma API RESTful que expõe os dados processados e armazenados para outras aplicações ou serviços.

4. **Matplotlib** (opcional): Biblioteca Python para criação de gráficos e visualizações de dados. Pode ser usada em conjunto com o Apache Spark para criar gráficos e visualizações a partir dos dados processados.
5. **Pandas** (opcional): Biblioteca Python para manipulação e análise de dados. Pode ser usada em conjunto com o Apache Spark para realizar análises e limpeza de dados em pequena escala antes de processá-los no Spark.
6. **Python**: a linguagem de programação utilizada em todas as etapas do projeto, desde a coleta de dados até a análise e visualização. O Python é uma linguagem de programação interpretada, orientada a objetos e de alto nível, que possui uma vasta biblioteca padrão e diversas bibliotecas de terceiros para processamento de dados.
7. **Streamlit**: uma biblioteca de código aberto para criação de aplicativos web de dados em Python. O Streamlit é utilizado no projeto para criar uma interface amigável e interativa para visualização dos dados.
8. **Plotly**: uma biblioteca de visualização de dados interativa de código aberto para Python. O Plotly é utilizado no projeto para criar gráficos e visualizações interativas a partir dos dados processados com o Pandas.
9. **Apache Airflow**: uma plataforma de orquestração de fluxo de trabalho para gerenciamento de tarefas de processamento de dados. O Apache Airflow é utilizado no projeto para agendar e executar tarefas de coleta, processamento e análise de dados de forma automática.
10. **Docker**: uma plataforma de código aberto para criação, implantação e execução de aplicativos em contêineres. O Docker é utilizado no projeto para criar e executar contêineres para cada serviço envolvido no projeto.
11. **Apache Spark**: um framework de computação distribuída de código aberto para processamento de dados em larga escala. O Apache Spark é utilizado no projeto para processar e analisar grandes quantidades de dados.
12. **Apache Hadoop**: um framework de computação distribuída de código aberto para armazenamento de dados em larga escala. O Apache Hadoop é utilizado no projeto para armazenar os dados coletados e processados.

## Pré-requisitos

- Python 3.6+
- Apache Spark 3.0+
- Power BI Desktop
- Docker
- Docker Compose

- Node.js e npm
- Git
- Insomnia
- IDE de sua preferência (PyCharm, VS Code, etc.)
- Terminal de sua preferência (Git Bash, PowerShell, etc.)
- Sistema operacional Linux, macOS ou Windows
- Conhecimentos básicos de Python e SQL

## REQUISITOS DE SOFTWARE

- Sistema Operacional: Windows/Linux/Mac
- IDE: Notebook Jupyter
- Bibliotecas Python: Pandas, Numpy, spacy, gensim, NLTK, langdetect e textblob

## REQUISITOS DE HARDWARE

- Ram 6GB+
- Rom 128GB+
- Processador Intel i3 ou superior

## Arquivo de configuração package.json

O arquivo `package.json` contém as dependências do projeto. Para instalar as dependências, execute o seguinte comando:

```
bash { "name": "analise-de-dados", "author": "grupo-estacio",
"version": "1.0.3", "main": "./build/electron/main.js",
"keywords": [ "restaurante", "python", "flask" ], "scripts": {
"dump": "dump-stlite-desktop-artifacts", "dev:app": "streamlit
run app.py", "production": "NODE_ENV=\"production\" electron .",
"start": "node ./dist/server.js", "servewindows": "electron .",
"pack": "electron-builder --dir", "prisma": "npx prisma", "dist":
"electron-builder", "postinstall": "electron-builder install-app-
deps", "server": "nodemon --exec npx babel-node client/src/api/
router/router.js" }, "build": { "files": ["build/**/*"],
"directories": { "buildResources": "assets" }, "win": { "target":
"portable", "icon": "assets/icon.ico" } }, "dependencies": { "-":
"^0.0.1", "@prisma/client": "^4.13.0", "@types/multer": "^1.4.7",
"D": "^1.0.0", "cors": "^2.8.5", "csv-parser": "^3.0.0", "csv-
writer": "^1.6.0", "express": "^4.18.2", "multer": "^1.4.5-lts.
1", "node-fetch": "^3.3.1", "typescript": "^5.0.4" },
"devDependencies": { "@stlite/desktop": "^0.22.2", "electron":
```

```
"22.0.0", "electron-builder": "^23.6.0", "@babel/core":  
"7.21.5", "@babel/node": "7.20.7", "@types/cors": "2.8.13",  
"@types/express": "4.17.17", "prisma": "4.13.0", "nodemon":  
"2.0.22" } }
```

## Instalando o Projeto

1. Clone o repositório:

```
bash git clone https://github.com/big-data-estacio/data.git cd  
data
```

## Executando o Projeto

### Configurando o ambiente virtual

É recomendado utilizar um ambiente virtual para isolar as dependências do projeto. Siga os passos abaixo para configurar e ativar o ambiente virtual usando o venv:

1. Instale o módulo venv, caso ainda não tenha, com o seguinte comando:

```
bash python3 -m pip install --user virtualenv
```

1. Navegue até a pasta do projeto e crie um ambiente virtual:

```
```bash virtualenv myenv
```

## ou, se você estiver no Windows

```
py -m venv myenv ```
```

1. Ative o ambiente virtual:

2. No Windows:

```
```bash .\myenv\Scripts\activate
```

## ou, se você estiver usando o Git Bash

```
source myenv/Scripts/activate ```
```

- No macOS e Linux:



```
bash set -x VIRTUAL_ENV /mnt/c/Users/estev/OneDrive/Área de Trabalho/johanEstevam/myenv source myenv/bin/activate.fish
```

1. Após a ativação do ambiente virtual, seu terminal deve mostrar o prefixo (venv).

Agora você pode executar o projeto com as dependências instaladas no ambiente virtual. Lembre-se de ativar o ambiente virtual sempre que for trabalhar no projeto.

1. Instale todas as bibliotecas necessárias usando o comando `pip install`:

```
bash pip install biblioteca1 biblioteca2 biblioteca3
```

Substitua `biblioteca1`, `biblioteca2` e `biblioteca3` pelos nomes das bibliotecas que você deseja instalar.

1. Agora que todas as bibliotecas estão instaladas, execute o seguinte comando para gerar o arquivo `requirements.txt`:

```
bash pip freeze > requirements.txt
```

O comando `pip freeze` listará todas as bibliotecas instaladas no ambiente virtual e suas versões específicas. O operador `>` redirecionará a saída para o arquivo `requirements.txt`, criando-o ou atualizando-o, se já existir.

1. Instale as dependências do projeto utilizando o arquivo `requirements.txt`:

```
bash pip install -r requirements.txt
```

arquivo `requirements.txt` para esse projeto com as dependências necessárias.

```
bash aiohttp==3.8.4 aiosignal==1.3.1 altair==4.2.2 async-timeout==4.0.2 attrs==23.1.0 beautifulsoup4==4.12.2 blinker==1.6.2 Brotli==1.0.9 cachetools==5.3.0 certifi==2023.5.7 cffi==1.15.1 charset-normalizer==3.1.0 click==8.1.3 contourpy==1.0.7 cssselect2==0.7.0 cycler==0.11.0 decorator==5.1.1 deta==1.1.0 entrypoints==0.4 Faker==18.9.0 favicon==0.7.0 fonttools==4.39.4 frozenlist==1.3.3 gitdb==4.0.10 GitPython==3.1.31 htbuilder==0.6.1 html5lib==1.1 idna==3.4 importlib-metadata==6.6.0 Jinja2==3.1.2 jsonschema==4.17.3 kiwisolver==1.4.4 lxml==4.9.2 Markdown==3.4.3 markdown-it-py==2.2.0 markdownlit==0.0.7 MarkupSafe==2.1.2 matplotlib==3.7.1 mdurl==0.1.2 more-itertools==9.1.0 multidict==6.0.4 numpy==1.24.3 openai==0.27.7 packaging==23.1 pandas==2.0.1 Pillow==9.5.0 plotly==5.14.1 protobuf==3.20.3 pyarrow==12.0.0 pycparser==2.21
```

```
pydeck==0.8.1b0 pydyf==0.6.0 Pygments==2.15.1 pymdown-  
extensions==10.0.1 Pympler==1.0.1 pyparsing==3.0.9 pyphen==0.14.0  
pypng==0.20220715.0 pyrsistent==0.19.3 python-dateutil==2.8.2  
pytz==2023.3 PyYAML==6.0 qrcode==7.4.2 requests==2.31.0  
rich==13.3.5 six==1.16.0 smmap==5.0.0 soupsieve==2.4.1 st-  
annotated-text==4.0.0 tqdm==0.0.5 streamlit==1.22.0 streamlit-  
camera-input-live==0.2.0 streamlit-card==0.0.4 streamlit-  
embedcode==0.1.2 streamlit-extras==0.2.7 streamlit-faker==0.0.2  
streamlit-image-coordinates==0.1.3 streamlit-keyup==0.2.0  
streamlit-lottie==0.0.3 streamlit-toggle-switch==1.0.2 streamlit-  
vertical-slider==1.0.2 tenacity==8.2.2 tinycss2==1.2.1  
toml==0.10.2 toolz==0.12.0 tornado==6.3.2 tqdm==4.65.0  
typing_extensions==4.6.0 tzdata==2023.3 tzlocal==5.0.1  
urllib3==2.0.2 validators==0.20.0 watchdog==3.0.0  
weasyprint==59.0 webencodings==0.5.1 yarl==1.9.2 zipp==3.15.0  
zopfli==0.2.2
```

1. Quando terminar de trabalhar no projeto, você pode desativar o ambiente virtual com o seguinte comando:

```
bash deactivate
```

## Continuando a instalação

1. Crie um arquivo `.env` na raiz do projeto com as variáveis de ambiente necessárias. Você pode usar o arquivo `.env.example` como modelo.

```
bash cp .env.example .env
```

O código acima copiará o arquivo `.env.example` e o renomeará para `.env` e você poderá preencher as variáveis de ambiente necessárias. *Nota: O arquivo `.env` é ignorado pelo Git, portanto, não será enviado para o repositório.*

1. Instale as dependências do projeto com o yarn na raiz do projeto:

```
bash yarn add
```

## Para iniciar o projeto "Pedacinho do Céu" com o Docker, siga estas etapas:

1. Certifique-se de que o Docker e o Docker Compose estejam instalados em seu sistema. Se você ainda não os instalou, siga as instruções de instalação no site oficial do Docker: <https://docs.docker.com/get-docker/> e <https://docs.docker.com/compose/install/>

2. Abra um terminal e navegue até o diretório raiz do projeto data.
3. Execute o seguinte comando para criar e iniciar os containers do projeto, conforme definido no arquivo `docker-compose.yml`:

- `docker build -t big_data_app_estacio .`
- `docker run -p 8501:8501 -e MYSQL_USER=user -e MYSQL_PASSWORD=user -e MYSQL_DATABASE=big_data_app_estacio -e MYSQL_HOST=192.168.1.100 big_data_app_estacio`
- Open `http://localhost:8501/`
- ou
- acesse o app do docker e execute o container docker

Isso iniciará os serviços do Docker, a API, como o MySQL e o serviço de processamento de dados se necessário.

1. Abra os relatórios do Power BI na pasta `data_visualization/power_bi_reports/` para visualizar e analisar os dados processados. Se você não possui o Power BI Desktop, faça o download e instale-o a partir do site oficial: <https://powerbi.microsoft.com/en-us/desktop/>
2. Ao longo do desenvolvimento do projeto, você pode modificar e ajustar os arquivos na pasta `src/` conforme necessário para aprimorar a análise e o processamento dos dados.

Lembre-se de que, dependendo da configuração do projeto, algumas etapas podem variar ou exigir ações adicionais. Ajuste as etapas conforme necessário para se adequar às necessidades específicas do seu projeto.

1. Quando terminar de usar o projeto, pare os serviços do Docker Compose pressionando `Ctrl+C` no terminal onde o Docker Compose está sendo executado. Para remover os contêineres e os volumes, execute:

```
bash docker-compose down --remove-orphans --volumes
```

**Para iniciar o projeto "Pedacinho do Céu" sem o Docker, siga estas etapas:**

1. Abra um terminal e navegue até o diretório raiz do projeto data.
2. Execute o seguinte comando para executar o projeto na pasta `client`:

```
bash cd client yarn run server:app
```

**ou**

```
``bash cd [client] streamlit run app.py --server.address 0.0.0.0 --server.port
```

**http://0.0.0.0:[your port]**

...

## Iniciando a API

1. Inicie o servidor da API:

```
bash node client/src/api/index.js
```

Já o arquivo `api/app.py`, é o arquivo principal da API, onde é feita a conexão com o banco de dados e a definição dos endpoints.

Ele é executado na porta 5000, e pode ser acessado em `http://localhost:5000/`

## Demo

A demonstração do projeto pode ser acessada em `https://pedacinho-do-ceu.herokuapp.com/`

Seu navegador não suporta a tag de vídeo.

## Acessando as funcionalidades do projeto

- **/**: Página inicial do projeto, onde é possível visualizar os dados processados e analisados.
- **/about**: Página com informações sobre o projeto e os desenvolvedores.
- **/data**: Página com informações sobre os dados utilizados no projeto.
- **/data-visualization**: Página com informações sobre a visualização dos dados.
- **/data-processing**: Página com informações sobre o processamento dos dados.

## Utilizando a API

A API RESTful permite gerenciar os dados armazenados no banco de dados PostgreSQL. Para utilizar a API, você pode fazer requisições HTTP para os seguintes endpoints:

Nesse arquivo, se encontram os endpoints da API, que são:

- **GET** `/`: Lista todos os itens armazenados no banco de dados.
- **GET** `/<id>`: Retorna os detalhes de um item específico com base no ID.
- **POST** `/`: Adiciona um novo item ao banco de dados.
- **PUT** `/<id>`: Atualiza os detalhes de um item específico com base no ID.
- **DELETE** `/<id>`: Remove um item específico com base no ID.

@TODO: Adicionar mais detalhes sobre a API

@Note: A API foi desenvolvida com o framework FastAPI, que é um framework web assíncrono de alto desempenho, fácil de aprender, rápido para codificar, pronto para produção. Ele é construído com base no Starlette e Pydantic.

o arquivo `api/app.py` está configurado com os scripts de coleta de dados e processamento de dados, para que os dados sejam coletados e processados automaticamente quando a API for iniciada:

```
python from flask import Flask from routes.bebidas import bebidas_bp from routes.clientes import clientes_bp
```

```
app = Flask(name) app.register_blueprint(bebidas_bp) app.register_blueprint(clientes_bp)
```

```
if name == 'main': app.run(debug=True)
```

A API estará disponível na porta 5000.

- Abra o Power BI Desktop e carregue os relatórios na pasta `data_visualization/power_bi_reports`.

## Testes

Execute os testes unitários e de integração usando o seguinte comando:

```
bash python -m unittest discover tests
```

## Debugging e Logging

Para depurar o projeto, você pode usar o depurador interativo do Python. Para isso, basta adicionar o seguinte código ao arquivo que você deseja depurar:

```
python import pdb; pdb.set_trace()
```

Em seguida, execute o arquivo normalmente e o depurador interativo será iniciado quando o código atingir o ponto de interrupção.

Para adicionar logs ao projeto, você pode usar o módulo logging do Python. Para isso, basta adicionar o seguinte código ao arquivo que você deseja depurar:

```
```python import logging
```

```
logging.basicConfig(level=logging.DEBUG) ```
```

como que fica o arquivo de log logs/app.log

Exemplo de arquivo de log

```
[IPv4 ('127.0.0.1', 9092)]>: connecting to localhost:9092 [('127.0.0.1', 9092) IPv4]
2023-04-25 09:39:44,780 - INFO - Probing node bootstrap-0 broker version
2023-04-25 09:39:44,781 - ERROR - Connect attempt to [IPv4 ('127.0.0.1', 9092)]>
returned error 111. Disconnecting. 2023-04-25 09:39:44,782 - INFO - [IPv4
('127.0.0.1', 9092)]>: Closing connection. KafkaConnectionError: 111
ECONNREFUSED 2023-04-25 09:39:44,827 - INFO - [IPv4 ('127.0.0.1', 9092)]>:
connecting to localhost:9092 [('127.0.0.1', 9092) IPv4] 2023-04-25 09:39:44,828 -
ERROR - Connect attempt to [IPv4 ('127.0.0.1', 9092)]> returned error 111.
Disconnecting. 2023-04-25 09:39:44,828 - INFO - [IPv4 ('127.0.0.1', 9092)]>: Closing
connection. KafkaConnectionError: 111 ECONNREFUSED 2023-04-25 09:41:52,885 -
INFO - A aplicação foi encerrada com sucesso. 2023-04-25 09:41:53,647 - INFO -
[IPv4 ('127.0.0.1', 9092)]>: connecting to localhost:9092 [('127.0.0.1', 9092) IPv4]
2023-04-25 09:41:53,648 - INFO - Probing node bootstrap-0 broker version
2023-04-25 09:41:53,649 - ERROR - Connect attempt to [IPv4 ('127.0.0.1', 9092)]>
returned error 111. Disconnecting. 2023-04-25 09:41:53,649 - INFO - [IPv4
('127.0.0.1', 9092)]>: Closing connection. KafkaConnectionError: 111
ECONNREFUSED 2023-04-25 09:41:53,693 - INFO - [IPv4 ('127.0.0.1', 9092)]>:
connecting to localhost:9092 [('127.0.0.1', 9092) IPv4] 2023-04-25 09:41:53,694 -
ERROR - Connect attempt to [IPv4 ('127.0.0.1', 9092)]> returned error 111.
Disconnecting. 2023-04-25 09:41:53,694 - INFO - [IPv4 ('127.0.0.1', 9092)]>: Closing
connection. KafkaConnectionError: 111 ECONNREFUSED 2023-04-25 09:42:12,859 -
INFO - A aplicação foi encerrada com sucesso. ```
```

O arquivo de debug:

```
[[debug](client/src/public/debug.png)](client/src/public/debug.png)
```

## Utilizando a API com Insomnia

Para testar e interagir com a API, você pode usar o Insomnia REST Client, que é uma aplicação de desktop para gerenciar requisições HTTP e API.

1. Instale o Insomnia em seu computador: Insomnia Download
2. Abra o Insomnia e crie um novo workspace ou use um existente.
3. Para adicionar as rotas da API ao seu workspace, clique em Create Request. Escolha o método HTTP (GET, POST, PUT, DELETE) e insira a URL da API (por exemplo, `http://localhost:5000/` para listar todos os itens).

Adicione os parâmetros, cabeçalhos ou corpo da requisição conforme necessário e clique em Send para executar a requisição.

4. Você pode criar várias requisições para diferentes endpoints da API e organizá-las em pastas para facilitar a navegação.

Agora você pode usar o Insomnia para testar e explorar a API RESTful do projeto Pedacinho do Céu.

## Versão atual

A versão atual do projeto inclui as seguintes funcionalidades:

- Coleta e armazenamento de dados em tempo real com Hadoop e Kafka.
- API RESTful para interagir com os dados e serviços.
- Docker Compose para facilitar a instalação e configuração do projeto.

## Coletando Dados

Os dados podem ser coletados usando o script `data_collection/data_collector.py`. Para coletar os dados, você pode executar o seguinte comando:

```
bash python data_collection/data_collector.py
```

Os dados coletados serão armazenados em arquivos no diretório `data_collection/data`.

## Processando Dados

Os dados podem ser processados usando o script `data_processing/data_processor.py`. Para processar os dados, você pode executar o seguinte comando:

```
bash python data_processing/data_processor.py
```

Os dados processados serão armazenados no banco de dados PostgreSQL.

## Visualizando os Dados

Os dados podem ser visualizados usando relatórios do Power BI. Para visualizar os relatórios, você pode fazer o download do Power BI Desktop [aqui](#). Após instalar o Power BI Desktop, você pode abrir o arquivo `data.pbix` para visualizar os relatórios.

# Futuras Atualizações

- Melhoria na coleta de dados
- Implementação de novas análises e visualizações
- Adição de mais testes
- Melhoria na documentação

O projeto "Pedacinho do Céu" está em constante evolução e melhorias. Abaixo estão algumas das atualizações futuras planejadas para o projeto:

1. **Expansão das fontes de dados:** Atualmente, o projeto utiliza um conjunto limitado de fontes de dados. No futuro, pretendemos expandir a lista de fontes de dados para enriquecer ainda mais nossas análises e visualizações.
2. **Otimização do processamento de Big Data:** Continuaremos a melhorar a eficiência e a velocidade do processamento de Big Data usando técnicas de otimização avançadas e atualizações nas ferramentas e tecnologias utilizadas.
3. **Visualizações interativas:** Atualmente, o projeto utiliza relatórios estáticos do Power BI. Planejamos adicionar visualizações interativas e em tempo real para tornar os dados mais acessíveis e fáceis de entender.
4. **Machine Learning:** Exploraremos a implementação de algoritmos de aprendizado de máquina para identificar padrões e tendências nos dados e fornecer insights adicionais.
5. **Integração com outras plataformas:** Planejamos adicionar suporte para integração com outras plataformas de visualização de dados e análise, como Tableau e Looker.
6. **APIs e microserviços:** Atualmente, o projeto possui uma API simples para acessar os dados. No futuro, pretendemos expandir a API e usar microserviços para fornecer uma solução mais modular e escalonável.
7. **Segurança e privacidade:** Continuaremos a aprimorar a segurança e a privacidade dos dados, garantindo a conformidade com os regulamentos de proteção de dados e as melhores práticas do setor.
8. **Documentação e tutoriais:** A documentação e os tutoriais serão atualizados regularmente para garantir que sejam claros, abrangentes e atualizados com as últimas informações do projeto.



# Tecnologias e conceitos utilizados

## Docker

O projeto é executado em contêineres Docker para facilitar a implantação e a escalabilidade.

## SOLID

Os princípios SOLID foram aplicados neste projeto para garantir um código limpo, modular e fácil de manter.

## Padrões de commit

Ao fazer um commit, siga as diretrizes de mensagens de commit personalizadas e com emojis, utilizando até 200 caracteres.

## Roadmap v1

- Adicionar suporte ao Travis CI para integração contínua.
- Aplicar os princípios SOLID ao projeto para melhorar a manutenibilidade e escalabilidade.
- Atualizar o SQLAlchemy para usar um banco de dados mais robusto, como PostgreSQL ou MySQL, em vez de SQLite.
- Implementar autenticação e autorização para proteger as rotas da API.
- Adicionar testes automatizados para garantir a qualidade do código.
- Implementar uma interface de usuário para interagir com a API.

## Travis CI

1. Para adicionar suporte ao Travis CI, siga estas etapas:
2. Vá para `travis-ci.com` e faça login com sua conta do GitHub.
3. Ative o repositório do projeto nas configurações do Travis CI.
4. Crie um arquivo `.travis.yml` na raiz do projeto com o seguinte conteúdo:

```
``yaml language: python python: - "3.8"
```

```
install: - pip install -r api/requirements.txt
```

```
script: - pytest ``
```

1. Vincule seu repositório do GitHub ao Travis CI e habilite a integração contínua para o projeto.

## Princípios SOLID

Para aplicar os princípios SOLID ao projeto, siga estas diretrizes:

1. Single Responsibility Principle (SRP): Separe as responsabilidades do código em classes e funções distintas. Por exemplo, separe a lógica de negócio da lógica de banco de dados.
2. Open/Closed Principle (OCP): Crie classes e funções extensíveis sem modificar seu código-fonte. Por exemplo, use a herança e a composição para estender a funcionalidade de uma classe.
3. Liskov Substitution Principle (LSP): As subclasses devem ser substituíveis por suas classes base sem alterar a correção do programa. Por exemplo, verifique se as subclasses seguem o contrato estabelecido pelas classes base.
4. Interface Segregation Principle (ISP): Divida as interfaces grandes em interfaces menores e mais específicas. Por exemplo, crie interfaces separadas para operações de leitura e gravação em vez de uma interface única.
5. Dependency Inversion Principle (DIP): Dependências de alto nível não devem depender de dependências de baixo nível; em vez disso, ambas devem depender de abstrações. Por exemplo, use a injeção de dependência para inverter as dependências entre os módulos.

## SQLAlchemy com PostgreSQL ou MySQL

Para usar o PostgreSQL ou MySQL com SQLAlchemy, siga estas etapas:

1. Instale os pacotes necessários:

```
bash pip install psycopg2 sqlalchemy psycopg2-binary
```

OU

```
bash pip install pymysql sqlalchemy
```

1. Atualize a string de conexão do banco de dados no arquivo `api/database.py` para usar PostgreSQL ou MySQL:

Para PostgreSQL:

```
``python from sqlalchemy import create_engine from sqlalchemy.ext.declarative
import declarative_base from sqlalchemy.orm import sessionmaker
```

```
DATABASE_URL = "postgresql://user:password@localhost/dbname"
```

```
engine = create_engine(DATABASE_URL) SessionLocal =  
sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

```
Base = declarative_base()
```

```
def get_db(): db = SessionLocal() try: yield db finally: db.close() ``
```

Para MySQL:

```
``python from sqlalchemy import create_engine from sqlalchemy.ext.declarative  
import declarative_base from sqlalchemy.orm import sessionmaker
```

```
DATABASE_URL = "mysql+pymysql://user:password@localhost/dbname"
```

```
engine = create_engine(DATABASE_URL) SessionLocal =  
sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

```
Base = declarative_base()
```

```
def get_db(): db = SessionLocal() try: yield db finally: db.close() ``
```

1. Atualize as configurações do banco de dados em `api/__init__.py` para usar a nova string de conexão.

## Autenticação e autorização

Para implementar a autenticação e autorização, você pode usar a extensão Flask-Login ou Flask-Security. Siga a documentação oficial de cada extensão para configurar a autenticação e autorização no projeto.

## Testes automatizados

Para adicionar testes automatizados, siga estas etapas:

1. Instale o pacote pytest:

```
bash pip install pytest
```

1. Crie um diretório `tests` na raiz do projeto.
2. Dentro do diretório `tests`, crie arquivos de teste para cada módulo do projeto. Por exemplo, você pode criar um arquivo `test_restaurant_service.py` para testar a classe `RestaurantService`.
3. Escreva testes usando o `pytest` para cada função e classe do projeto.
4. Execute os testes usando o comando `pytest` no terminal.

## **Interface de usuário**

Para implementar uma interface de usuário, você pode criar uma aplicação web usando HTML, CSS e JavaScript para interagir com a API. Você pode usar bibliotecas como React, Angular ou Vue.js para criar a interface do usuário.

## **Roadmap v2**

### **Versão 1.1**

- Implementar autenticação e autorização na API.
- Adicionar suporte para diferentes formatos de dados (JSON, XML, etc.).
- Desenvolver uma interface gráfica para visualizar os dados e interagir com a API.

### **Versão 1.2**

- Adicionar suporte para processamento de dados em tempo real usando Apache Spark ou Flink.
- Implementar algoritmos de machine learning para análise de dados e geração de insights.
- Integrar com outras fontes de dados e APIs de terceiros.

### **Versão 1.3**

- Adicionar suporte para escalabilidade horizontal e balanceamento de carga.
- Implementar monitoramento e alertas para garantir a saúde e o desempenho do sistema.
- Desenvolver testes de carga e estresse para garantir a estabilidade do projeto em ambientes de produção.

### **Versão 2.0**

- Redesenhar a arquitetura do projeto para melhorar a modularidade e a manutenibilidade.
- Adicionar suporte para outras linguagens de programação e frameworks.
- Implementar uma API GraphQL para maior flexibilidade na consulta e manipulação de dados.

## **O que é o Apache Spark?**

# Apache\_Spark

:sparkles::tada: Este repositório explica sobre o Apache Spark com práticas :tada::sparkles:

- É uma ferramenta de processamento em tempo real desenvolvida para resolver o problema de trabalhar com dados em tempo real.
- É tratado pelo Apache.
- É muito rápido, dá resultado no clique.
- Usa avaliação preguiçosa, ou seja, processa sempre que necessário.
- Como o MapReduce não conseguiu lidar com dados em tempo real, o Spark entrou em cena para ajudar.
- Agora é usado por muitos grandes gigantes da tecnologia como Oracle, Amazon, Microsoft, Visa, Cisco, Verizon, Hortonworks.
- Como acima, temos 3.000 empresas usando o Apache Spark.

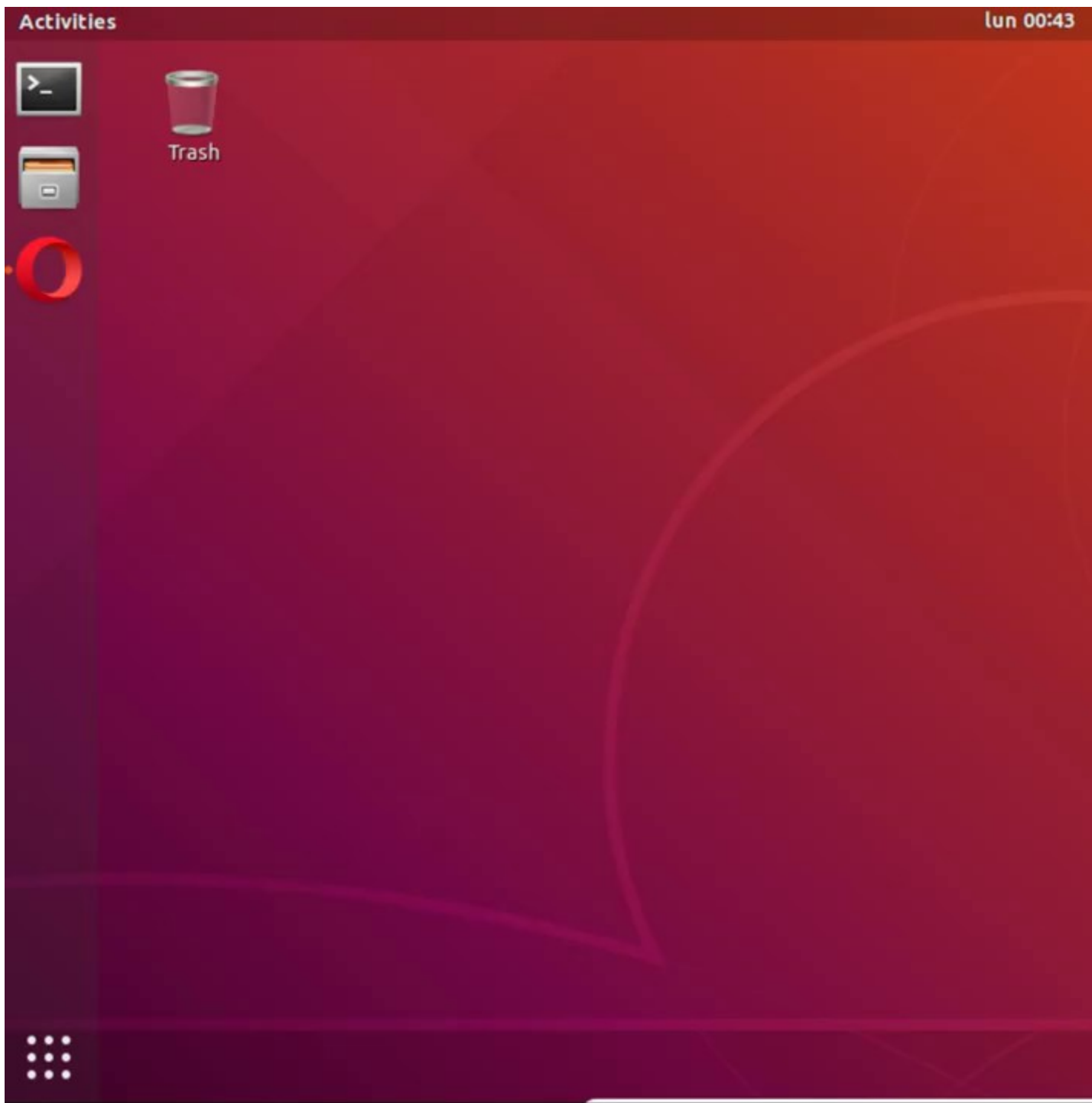
**DICIONÁRIO DO PROGRAMADOR**



**Para iniciar o cluster Spark a partir da máquina local com o Linux, você precisa seguir os seguintes passos:**

**Instalação e Execução do Apache Spark no Ubuntu/Debian/Linux Mint/Pop!\_OS/Elementary OS/Linux Lite/Peppermint OS/Windows 10 com WSL**

Este guia tem como objetivo fornecer os passos necessários para a instalação e execução do Apache Spark em um ambiente Linux. O Apache Spark é um sistema de computação distribuída que permite processar grandes volumes de dados em um cluster de computadores. Ele é amplamente utilizado para análise de big data, machine learning, processamento de stream e outras aplicações de processamento paralelo.



## Pré-requisitos

---

Antes de começar a instalar o Apache Spark, certifique-se de que o seu sistema atenda aos seguintes requisitos:

- Linux (Ubuntu ou outra distribuição)
- Java 8 ou superior
- Python 2.7 ou 3.x
- Acesso à Internet



- Um sistema operacional Debian ou seus derivados.
- Acesso à linha ou terminal de comando.
- Um usuário com permissões sudo ou **\*\* root\*\***.

Antes de baixar e configurar o Spark, precisamos instalar dependências. Essas etapas incluem a instalação da seguinte parcela.

- JDK
- Scala
- Git

Certifique-se de que o Java esteja instalado em sua máquina.

- Instale o Java 8 caso não esteja instalado:

Abra o terminal e execute o seguinte comando: `sudo apt update && sudo apt install openjdk-8-jdk`

- Verifique a versão do Python instalada:

```
bash python --version
```

- Instale o Python 3.7 ou superior caso não esteja instalado:

```
bash sudo apt update sudo apt install python3
```

## 1. Verifique se o Java está instalado

Antes de começar a instalar o Apache Spark, verifique se o Java está instalado em sua máquina. O Apache Spark requer que o Java esteja instalado.

Para verificar a versão do Java em sua máquina, execute o seguinte comando em seu terminal:

```
java -version
```

Se o Java não estiver instalado, siga as instruções no site oficial da Oracle para baixar e instalar o Java.

## Instalação por gerenciador de pacotes

1. Abra o terminal e execute o seguinte comando para instalar o Apache Spark:

```
sql
```

```
sql sudo apt update sudo apt install spark
```

## Abra o navegador e digite o seguinte URL na barra de endereço para instalar o Apache Spark:

Agora, precisamos baixar a versão do Spark que você deseja e ela está disponível no site oficial. No momento da edição deste texto, a versão mais atualizada é *Spark 3.2.1* ( Enero-26-2022 ) junto com o pacote *Hadoop 3.2*.

Usamos o comando **\*\* wget \*\*** junto com o link do site para baixar nosso arquivo Spark:

```
bash https://www.apache.org/dyn/closer.lua/spark/spark-3.4.0/spark-3.4.0-bin-hadoop3.tgz
```

---

**\*\*Nota: \*\*** Se o URL não for executado, vá para a página oficial [ Apache Spark ] ( <https://spark.apache.org/>) e procure a versão mais recente no menu de download ( ). Em outras palavras, você também pode tentar substituir as versões da liga que estou compartilhando com você.

---

- logo em seguida baixe na raiz do projeto o arquivo spark-3.4.0-bin-hadoop3.tgz
- mova o arquivo para a pasta /opt:

```
bash sudo mv spark-3.4.0-bin-hadoop3.tgz /opt
```

- acessar a pasta /opt:

```
bash cd /opt
```

- descompactar o arquivo:

```
bash sudo tar -xvzf spark-3.4.0-bin-hadoop3.tgz
```

- acessar a pasta spark-3.4.0-bin-hadoop3:

```
bash cd spark-3.4.0-bin-hadoop3
```

- configurar as variáveis de ambiente:

```
bash sudo nano /etc/profile
```

## Configure o ambiente Spark

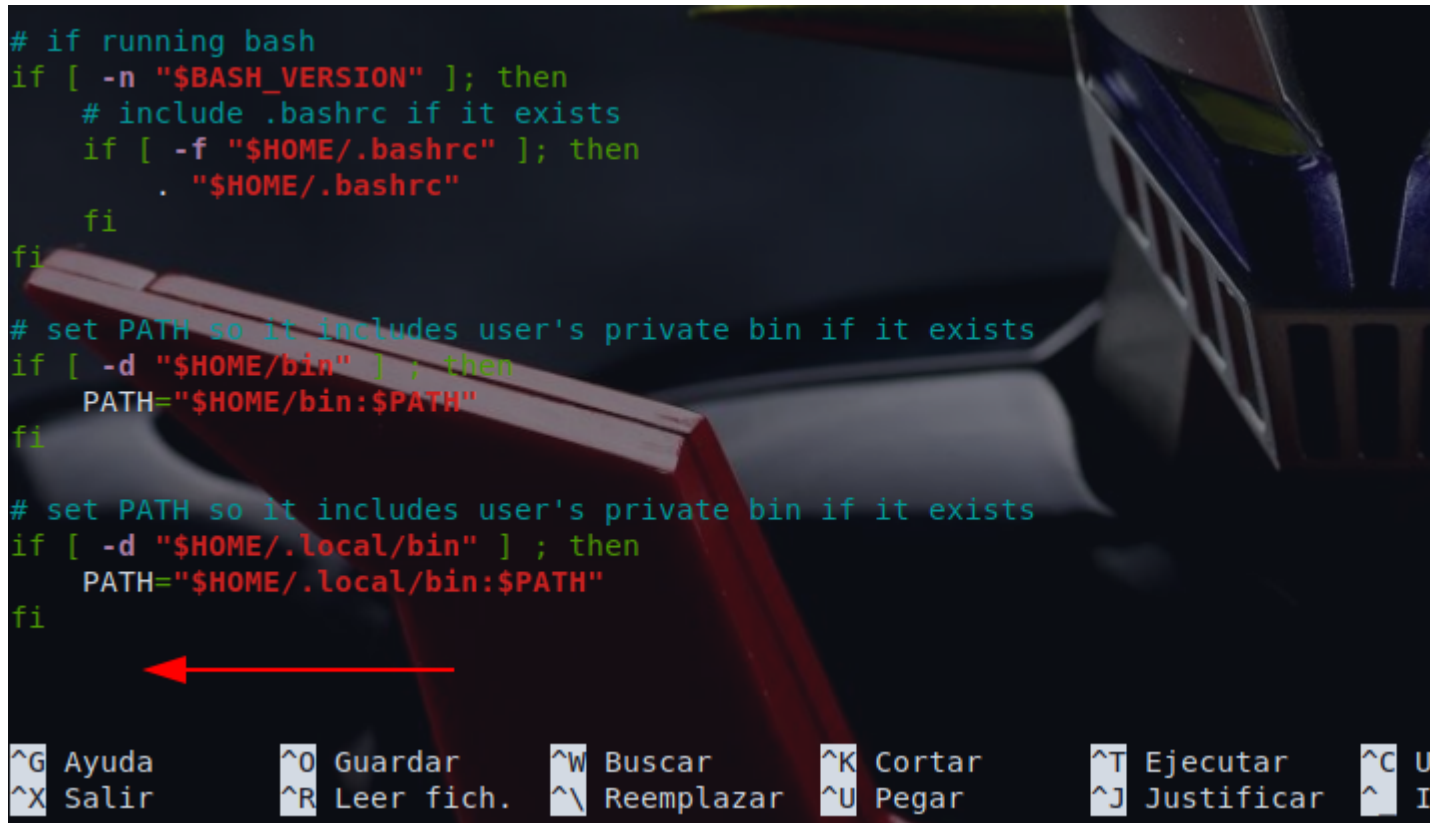
Antes de inicializar o servidor mestre, precisamos configurar as variáveis de ambiente. Esses são frequentemente caminhos ( ) no Spark que precisamos adicionar ao perfil do usuário.

Também podemos adicionar as rotas de exportação editando o arquivo *.profile* no editor escolhido, como nano ou vim.

Por exemplo, para o nano editor, inserimos:

```
javascript nano ~ / .profile
```

Ao carregar o perfil, colocamos no final do arquivo:



```
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
```

Então, adicionamos as duas linhas a seguir no final do arquivo:

```
bash export SPARK_HOME=/opt/spark-3.4.0-bin-hadoop3 export
PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

- salvar o arquivo e sair do editor de texto com os seguintes comandos:

```
bash CTRL + s CTRL + x
```

1. Execute o comando abaixo para atualizar as variáveis de ambiente:

```
bash bash -n ~/.bashrc
```

1. Verifique se o Apache Spark foi instalado corretamente:

```
bash spark-submit --version
```

Caso ocorra esse erro:

```
log Traceback (most recent call last): File "/home/estevam/
micromamba/bin/find_spark_home.py", line 95, in <module>
print(_find_spark_home()) File "/home/estevam/micromamba/bin/
find_spark_home.py", line 59, in _find_spark_home module_home =
os.path.dirname(find_spec("pyspark").origin) AttributeError:
'NoneType' object has no attribute 'origin' /home/estevam/
micromamba/bin/spark-submit: line 27: /bin/spark-class: No such
file or directory
```

Esse erro indica que o PySpark não foi instalado corretamente ou não foi encontrado no sistema.

Verifique se você seguiu todas as etapas corretamente durante a instalação e se adicionou as variáveis de ambiente corretamente.

Você pode tentar reinstalar o PySpark usando o comando `pip install pyspark` ou verifique se o caminho para o Spark Home está correto e atualizado nas variáveis de ambiente.

Agora poderá usar novamente o comando `spark-submit --version` e logo aparecerá isso no terminal:

```

`log 23/05/02 18:52:38 WARN Utils: Your hostname, estevam
resolves to a loopback address: 127.0.1.1; using 172.20.231.90
instead (on interface eth0) 23/05/02 18:52:38 WARN Utils: Set
SPARK_LOCAL_IP if you need to bind to another address Welcome to
____ _ / __/_ __ ____/ /__ _ \ \ / _ \ / _ \|'//. _ _// _ \| version 3.4.0
//

```

```
Using Scala version 2.12.17, OpenJDK 64-Bit Server VM, 11.0.18 Branch HEAD
Compiled by user xinrong.meng on 2023-04-07T02:18:01Z Revision
87a5442f7ed96b11051d8a9333476d080054e5a0 Url https://github.com/apache/
spark Type --help for more information. ``
```

rode agora o pyspark no terminal e verá essa informação:

```

`log Python 3.9.16 | packaged by conda-forge | (main, Feb 1
2023, 21:39:03) [GCC 11.3.0] on linux Type "help", "copyright",
"credits" or "license" for more information. 23/05/02 18:50:48
WARN Utils: Your hostname, estevam resolves to a loopback
address: 127.0.1.1; using 172.20.231.90 instead (on interface
eth0) 23/05/02 18:50:48 WARN Utils: Set SPARK_LOCAL_IP if you
need to bind to another address Setting default log level to
"WARN". To adjust logging level use sc.setLogLevel(newLevel). For
SparkR, use setLogLevel(newLevel). 23/05/02 18:50:51 WARN
NativeCodeLoader: Unable to load native-hadoop library for your

```

```
platform... using builtin-java classes where applicable Welcome
to ____ _ / __/_ ____ _/ /__ _\ \ / _ \ /'|'_|/._|_/||/_|version
3.4.0 //
```

Using Python version 3.9.16 (main, Feb 1 2023 21:39:03) Spark context Web UI available at <http://172.20.231.90:4040> Spark context available as 'sc' (master = local[\*], app id = local-1683064253636). SparkSession available as 'spark'.

///

Agora que o PySpark está rodando corretamente no seu ambiente, você pode seguir as etapas do roadmap que você criou anteriormente para executar o seu projeto.

Você pode, por exemplo, utilizar as funções que você criou anteriormente para ler os arquivos `estoque_mercadorias.csv` e `total_clientes.csv`, realizar transformações e análises nos dados utilizando o PySpark e gerar os gráficos utilizando o Altair.

Lembre-se de ajustar as funções para que elas utilizem o PySpark ao invés do Pandas para processar os dados. Isso pode envolver a utilização de operações específicas do PySpark como o `groupBy` e o `agg`.

## Inicializando o Servidor Mestre de spark Independente

Agora que concluímos as configurações do ambiente Spark, podemos inicializar o servidor mestre.

No terminal, escrevemos:

```
bash sudo /opt/spark-3.4.0-bin-hadoop3/sbin/start-master.sh
```

- acessar o endereço <http://localhost:8080/> para verificar se o servidor está funcionando

Para visualizar a interface da web Spark, abrimos um navegador da web e inserimos o endereço IP do host local na porta 8080.

http://127.0.0.1:8080/ ou http://localhost:8080/

- **OBS:** caso o servidor não esteja funcionando, verifique se a porta 8080 está liberada no firewall

A página mostra seu **\*\* URL Spark \*\***, informações sobre o status do trabalhador (trabalhadores ), recursos de hardware usados etc.

O URL do Master Spark é o nome do seu dispositivo na porta 8080. No meu caso, é `_** debian.gabi: 8080 **_`. Então, aqui temos três maneiras possíveis de carregar a UI Spark Master Web:

1. `127.0.0.1: 8080`
2. `localhost: 8080`
3. `nome do dispositivo: 8080`

## Inicialize o servidor do trabalhador Spark ( inicie um processo do trabalhador )

Nesta configuração autônoma de um único servidor, inicializaremos um servidor em funcionamento junto com o servidor mestre.

Para isso, executamos o seguinte comando neste formato:

```
javascript start-slave.sh spark: // master: port
```

O `** master **` neste comando pode ser um IP ou um nome de host.

No meu caso, é `** debian.gabi:**`

```
javascript start-slave.sh spark://estevam.localdomain:7077
```

Agora que o trabalhador ou slave está carregado e funcionando, se recarregarmos a interface do usuário da Web do Spark Master, você deverá vê-la na lista:

## Especifique a alocação de recursos para trabalhadores

A configuração padrão quando inicializamos um trabalhador em uma máquina é a disponível por todos os núcleos da CPU. Você pode especificar o número de núcleos que passam pelos sinalizadores `** -c **` para o comando `** start-slave**`.

```
javascript start-slave.sh -c 1 spark: //estevam.localdomain:7077
```

Recarregamos a interface do usuário da Web do Spark Master para confirmar as configurações do trabalhador.

### ▼ Workers (1)

Worker Id	Address
<code>worker-20220404090438-192.168.0.6-44391</code>	<code>192.168.0.6:44391</code>

Da mesma forma, podemos alocar a quantidade específica de memória quando inicializamos o trabalhador. A configuração padrão é a quantidade de memória RAM usada pela máquina menos 1 GB.

Ao iniciar um trabalhador e atribuir a ele uma quantidade específica de memória, adicionamos a opção `** -m **` e o número. Para gigabytes, usamos `** G **` e para megabytes, usamos `** M.**`

Por exemplo, para iniciar um trabalhador com 512 MB de memória, inserimos o seguinte comando:

```
javascript start-slave.sh -m 512MB spark://estevam.localdomain:7077
```

Recarregue a interface do usuário da Web do Spark Master para visualizar o status do trabalhador e confirmar as configurações

## Tente Spark Shell

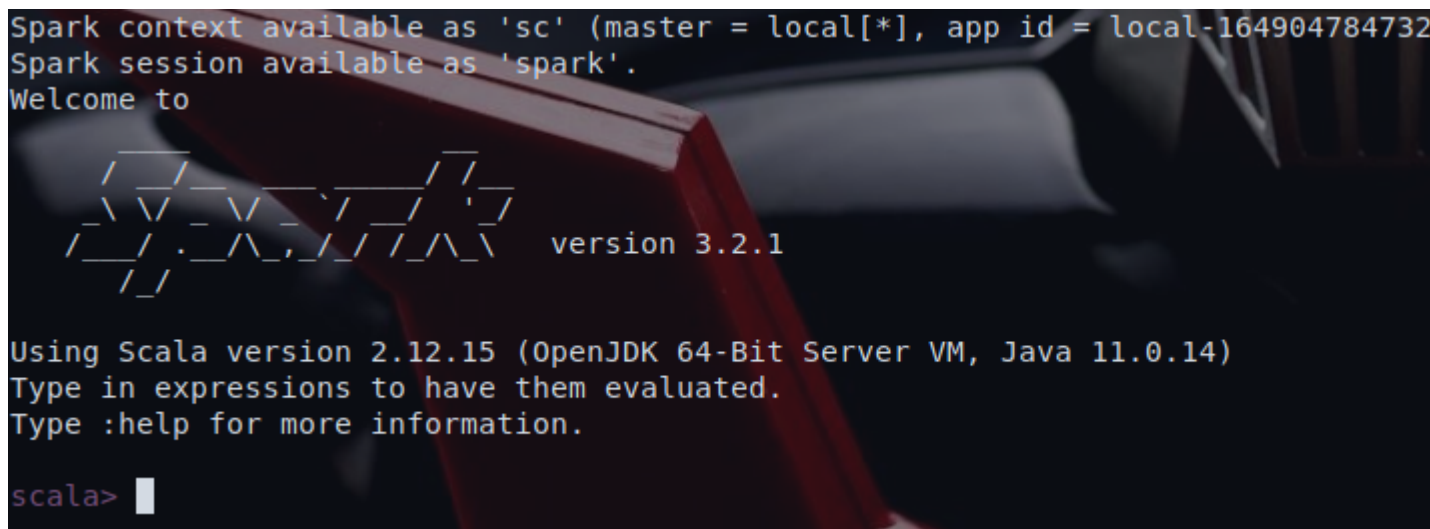
Depois de concluirmos a configuração e inicialização do servidor mestre e slave, testamos se o shell Spark funciona.

Carregamos o shell digitando:

```
javascript spark-shell
```

Você precisará obter uma notificação na tela e a descrição do Spark. Por padrão, a interface é Scala e carrega o shell quando você executa *spark-shell*.

No final da saída, a imagem aparecerá com a versão usada ao escrever este guia:



```
Spark context available as 'sc' (master = local[*], app id = local-164904784732)
Spark session available as 'spark'.
Welcome to

  ____ _
 / ___ \| | | |
 \___ \| |_| |
  ___) | __| |
 / ___ \| | | |
 \___ \| |_| |
  ___) | __| |
  ____|_|_|_|

version 3.2.1

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.14)
Type in expressions to have them evaluated.
Type :help for more information.

scala> █
```

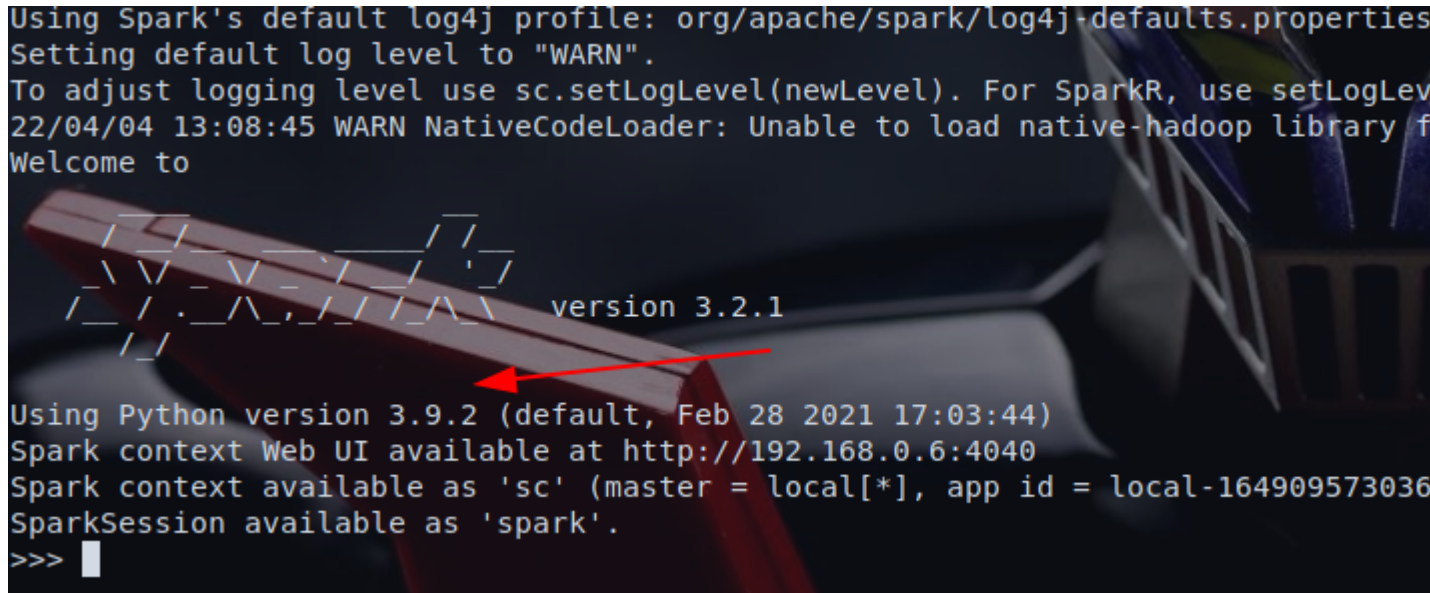
## Testando Python no Spark

Se você não quiser usar a interface Scala por padrão, poderá usar o Python.

Certifique-se de sair do Scala e, em seguida, inserimos o seguinte comando:

```
javascript pyspark
```

A saída resultante é semelhante à anterior. Na parte inferior, você verá a versão Python.



```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel
22/04/04 13:08:45 WARN NativeCodeLoader: Unable to load native-hadoop library f
Welcome to

version 3.2.1

Using Python version 3.9.2 (default, Feb 28 2021 17:03:44)
Spark context Web UI available at http://192.168.0.6:4040
Spark context available as 'sc' (master = local[*], app id = local-164909573036
SparkSession available as 'spark'.
>>> █
```

Para sair deste shell, digite ***exit()*** e pressione Enter.

## Comandos básicos para iniciar e parar o servidor Master e Escalavos ( Servidor mestre e de trabalhadores )

Você encontrará imediatamente os comandos básicos para iniciar e parar o servidor mestre e slave do Apache Spark. Essa configuração é para uma única máquina, os scripts são executados por padrão no host local.

***Para iniciar uma instância do servidor mestre*** na máquina atual, executamos o comando que já tínhamos usado:

```
javascript start-master.sh
```

***Para interromper a instância mestre***, começamos executando o próximo script, executamos:

```
javascript stop-master.sh
```



**Para parar um slave** que está em execução, inserimos o seguinte comando:

```
javascript stop-slave.sh
```

Na UI Spark Master Web, você será exibido no campo 'status' do Worker Id como MORTO.

Você pode **iniciar instâncias mestre e slave** usando o comando start-all:

```
start-all.sh
```

Agora que você iniciou o serviço do Apache Spark, você pode utilizar suas funcionalidades em seu projeto. Para fazer isso, você precisa primeiro iniciar o SparkSession em seu código Python. O SparkSession é o ponto de entrada para o Spark e fornece a interface para trabalhar com os dados no Spark.

Para iniciar o SparkSession em seu código Python, você pode usar o seguinte código:

```
```python import findspark findspark.init()

from pyspark.sql import SparkSession
```

**Aqui, substitua "local" pelo endereço do seu gerenciador de cluster.**

**Por exemplo, se você estiver usando o gerenciador de cluster autônomo do Spark, poderia ser algo como "spark://master:7077"**

```
spark = SparkSession.builder \ .master("spark://estevam.localdomain:7077") \
.appName("big-data") \ .getOrCreate() ```
```

O appName define o nome da sua aplicação e o master define o modo de execução do Spark. No exemplo acima, estamos usando local[] *para executar o Spark no modo local*. Você pode substituir local[] pelo endereço do servidor do Spark se você estiver executando o Spark em um cluster.

Com o SparkSession criado, você pode utilizar todas as funcionalidades do Spark, como ler e escrever dados de e para diferentes fontes de dados, realizar

transformações e operações em larga escala em seus dados, etc. Por exemplo, para ler um arquivo CSV e criar um DataFrame com os dados, você pode usar o seguinte código:

```
``python df = spark.read \ .option("header", True) \ .option("inferSchema", True) \
.csv("/caminho/para/o/arquivo.csv")

df.show() ``
```

Neste exemplo, estamos lendo um arquivo CSV que tem um cabeçalho e o Spark vai inferir o esquema dos dados. Você pode modificar as opções de leitura de acordo com suas necessidades.

- iniciar um ou mais workers Spark executando o seguinte comando:

A saída sugere que o comando `sudo /opt/spark-3.4.0-bin-hadoop3/sbin/start-slave.sh spark://estevam.localdomain:7077` está obsoleto e você deve usar `start-worker.sh` em seu lugar.

Além disso, a saída também sugere que já existe um trabalhador em execução como processo 2110 e você precisa interrompê-lo antes de iniciar um novo. Você pode interromper o worker executando o seguinte comando:

```
sudo /opt/spark-3.4.0-bin-hadoop3/sbin/stop-worker.sh
```

Depois que o worker anterior for interrompido, você poderá iniciar um novo worker usando o seguinte comando:

```
sudo /opt/spark-3.4.0-bin-hadoop3/sbin/start-worker.sh spark://
estevam.localdomain:7077
```

Observe que pode ser necessário inserir sua senha após executar o comando `ssudo`.

Isso inicia um worker Spark que se conecta ao mestre Spark no endereço `spark://estevam.localdomain:7077`. Você pode substituir `spark://estevam.localdomain:7077` pelo endereço do servidor Spark se você estiver executando o Spark em um cluster. Você pode iniciar vários workers Spark executando o comando acima várias vezes.

- No seu projeto Python, importe as bibliotecas do PySpark e crie uma sessão Spark:

```
``python from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("myApp").getOrCreate() ``
```

Em seguida, crie as funções que usam os serviços do Spark para realizar as tarefas desejadas. Por exemplo, se você quiser ler um arquivo CSV em um DataFrame do Spark, pode usar a seguinte função:

```
```python def read_csv_file(file_path): df =  
spark.read.format("csv").option("header", "true").load(file_path) return df  
  
df = read_csv_file("/path/to/file.csv")  
  
df.show() ```
```

Por fim, execute seu aplicativo Streamlit a partir do terminal com o comando:

```
bash streamlit run app.py
```

Depois de executar o comando `streamlit run app.py` e ter iniciado a aplicação Streamlit, você pode interagir com a interface do seu aplicativo no navegador. A partir daí, é possível utilizar as funcionalidades e algoritmos implementados no seu projeto que utilizam o Apache Spark como processador de dados. Por exemplo, se o seu aplicativo utiliza o Spark para processar dados de um conjunto de dados, você pode carregar esse conjunto de dados na interface e executar as operações definidas pelo seu algoritmo para processá-los. É importante lembrar que o Apache Spark funciona como um motor de processamento de dados distribuído, o que significa que ele é capaz de processar grandes conjuntos de dados de forma eficiente, o que pode ser particularmente útil em projetos que lidam com dados em escala.

- **Ou** pode optar por seguir as seguintes instruções:

- iniciar o worker:

```
bash sudo /opt/spark-3.4.0-bin-hadoop3/sbin/start-slave.sh  
spark://localhost:7077
```

- atualizar as variáveis de ambiente:

```
bash source /etc/profile
```

- acessar a pasta `spark-3.4.0-bin-hadoop3`:

```
bash cd spark-3.4.0-bin-hadoop3
```

- iniciar o master Spark no seu computador local:

```
bash ./sbin/start-master.sh
```

- pronto, o servidor apache spark está funcionando

- para parar o master Spark e todos os workers, execute o seguinte comando:

```
bash ./sbin/stop-all.sh
```

## Como iniciar o Apache Spark em um cluster com o Docker

- Para iniciar o Apache Spark em um cluster com o Docker, você precisa ter o Docker instalado em seu sistema. Você pode verificar se o Docker está instalado em seu sistema executando o seguinte comando:

```
bash docker --version
```

- Se o Docker estiver instalado em seu sistema, você verá a versão do Docker instalada em seu sistema. Caso contrário, você pode instalar o Docker em seu sistema seguindo as instruções em <https://docs.docker.com/engine/install/>.

**Agora o próximo passo é entrar na pasta onde o arquivo docker-compose.yml está localizado e executar o seguinte comando:**

```
bash cd backend/admin/apache/pyspark_docker/
```

Logo depois é seguir os passos abaixo

### Ambiente de encaixe do Pyspark

Se você simplesmente deseja testar um cluster de spark de nó único, basta executar `docker run -it wlongxiang/pyspark-2.4.4 pyspark`, isso trará seu shell de spark.

Ou você também pode criar um cluster multi-worker com um arquivo de composição simples, o exemplo montará os diretórios locais data e code para o cluster worker e master de forma que você possa alterar facilmente seu código e dados localmente e testá-los dentro do cluster do docker.

- Check-out do branch master
- Execute `docker-compose up` para ativar 2 workers e 1 master (ou você também pode definir `COMPOSE_FILE` env para um arquivo diff. compose)
- Acesse `http://localhost:8080` para ver a interface do usuário do Spark Master
- Execute `docker exec -it pyspark_docker_master_1 bash` para shell no contêiner de spark
- contagem de palavras: `spark-submit /code/wordcount.py /data/logs.txt`

Versões de spark suportadas: 2.4.4.

Alguns exemplos para testar, veja o diretório code.

## Como publicar uma nova imagem (manualmente)

- primeiro você faz `docker login` com suas credenciais para docker-hub
- então `docker build -t wlongxiang/pyspark-2.4.4:<version_tag> .`, este comando irá construir seu e-mail com o nome pyspark e marcar com 2.4.4
- verifique se sua imagem foi criada com sucesso por `docker images`
- finalmente `docker push wlongxiang/pyspark-2.4.4:<version_tag>`, no final isso estará disponível em seu repositório docker
- agora, tudo deve ser capaz de executar sua imagem ou usá-la no arquivo `docker-compose`, como `docker run -it pyspark-2.4.4:<version_tag> bash`

## Conclusão

Com isso, você já está pronto para instalar e executar o Apache Spark em seu sistema. Lembre-se de sempre verificar a documentação oficial do Apache Spark em <https://spark.apache.org/docs/latest/> para informações mais detalhadas.

Caso for enviar as alterações feitas com o arquivo `spark-3.2.1-bin-hadoop2.7.tgz`, não esqueça de deletar o arquivo do diretório antes de enviar, ou posso usar os comandos:

```
git lfs install
git lfs track "venv/lib/python3.9/site-packages/pyspark/jars/rocksdbjni-7.9.2.jar"
git lfs track "backend/spark-3.2.4-bin-hadoop3.2.tgz"
```

- caso ocorra esse erro:

```
log git lfs install
git lfs track "venv/lib/python3.9/site-packages/pyspark/jars/rocksdbjni-7.9.2.jar"
git lfs track "backend/spark-3.2.4-bin-hadoop3.2.tgz"
git: 'lfs' is not a git command. See 'git --help'.
```

The most similar command is `lsfn` git: 'lfs' is not a git command. See 'git --help'.

The most similar command is `lsfn` git: 'lfs' is not a git command. See 'git --help'.

The most similar command is `lsfn` ``

- execute o comando no linux:

```
bash sudo apt-get install git-lfs
```

Em seguida, execute novamente os comandos para rastrear os arquivos grandes e fazer o commit e push novamente.

```
bash git lfs install git lfs track "venv/lib/python3.9/site-packages/pyspark/jars/rocksdbjni-7.9.2.jar" git lfs track "backend/spark-3.2.4-bin-hadoop3.2.tgz" git add .gitattributes git add . git commit -m "Commit message" git push
```

Certifique-se de também confirmar o seu email e o nome de usuário do git com o comando `git config --global user.email "you@example.com"` e `git config --global user.name "Your Name"`.

## Linguagens usadas para programação no Spark:

- Python
- Escala

## Instruções para executar scripts do Apache Spark

- Copie o código do respectivo script e cole-o no shell Spark correspondente para executar o código.

## Instruções para executar scripts python

- Abra o terminal (no ambiente Linux)/Command Prompt (Windows).
- Execute "python .py"

## CrITÉRIOS de aceitação do projeto

Com base nas informações fornecidas até agora, o projeto está usando as seguintes tecnologias e técnicas:

1. **Apache Hadoop e Apache Spark** como ferramentas de Big Data
2. **Integração com Python** para executar scripts e processar dados usando o Spark
3. **Flask** para criar uma API RESTful

De acordo com os critérios do professor, esse projeto estaria atendendo aos seguintes pontos:

- Usando alguma ferramenta de Big Data com Python (+++): O projeto está usando o Apache Hadoop e o Apache Spark, que são ferramentas de Big Data, e está integrando-as com Python para executar scripts e processar dados.

Como resultado, o projeto deve passar na matéria com uma nota alta, pois está utilizando ferramentas de Big Data e integrando-as com Python, o que é o critério de maior pontuação (+++).

Além disso, o projeto também utiliza outras tecnologias e práticas relevantes, como o Flask para criar uma API RESTful e o SQLAlchemy como um ORM para lidar com as operações de banco de dados. Esses aspectos podem não ser mencionados diretamente nos critérios de avaliação do professor, mas certamente agregam valor ao projeto e demonstram a capacidade de trabalhar com várias tecnologias e bibliotecas em Python.

Em resumo, o projeto atende ao critério de maior pontuação, utilizando ferramentas de Big Data e integrando-as com Python. Isso deve resultar em uma avaliação positiva para a matéria.

## Usando o Hadoop com o Apache Spark

O Apache Spark é uma ferramenta de Big Data que pode ser usada para processar grandes conjuntos de dados. Ele pode ser usado com o Apache Hadoop, que é um framework de software que permite o processamento distribuído de grandes conjuntos de dados em clusters de computadores usando modelos de programação simples. O Hadoop é usado para armazenar e processar grandes conjuntos de dados em um ambiente distribuído, enquanto o Spark é usado para processar esses dados.

O Apache Spark pode ser usado com o Apache Hadoop para processar grandes conjuntos de dados. O Hadoop é usado para armazenar e processar grandes conjuntos de dados em um ambiente distribuído, enquanto o Spark é usado para processar esses dados.

### 1. 1. Instalar e configurar o Apache Spark e o Hadoop

Antes de começar a usar o Apache Spark com o Hadoop em seu projeto, você precisa instalar e configurar ambos em seu sistema.

#### 1.1. Instalar o Apache Spark

Você pode baixar a versão mais recente do Apache Spark no site oficial do projeto: <https://spark.apache.org/downloads.html>. Certifique-se de escolher a versão que é compatível com a versão do Hadoop que você pretende usar.

Depois de baixar o arquivo tar do Apache Spark, extraia-o para a pasta de sua escolha usando o comando `tar -xvf spark-<versão>-bin-hadoop<versão>.tgz`.

1.2. Instalar o Hadoop Você pode baixar a versão mais recente do Hadoop no site oficial do projeto: <https://hadoop.apache.org/releases.html>. Certifique-se de escolher a versão que é compatível com a versão do Apache Spark que você pretende usar.

Depois de baixar o arquivo tar do Hadoop, extraia-o para a pasta de sua escolha usando o comando `tar -xvf hadoop-<versão>.tar.gz`.

1.3. Configurar as variáveis de ambiente Para usar o Apache Spark e o Hadoop em seu projeto, você precisa configurar as variáveis de ambiente em seu sistema. Você pode fazer isso adicionando as seguintes linhas ao seu arquivo `~/ .bashrc`:

```
bash export SPARK_HOME=/caminho/para/o/apache/spark export
HADOOP_HOME=/caminho/para/o/hadoop export PATH=$SPARK_HOME/bin:
$HADOOP_HOME/bin:$PATH
```

Certifique-se de substituir `/caminho/para/o/apache/spark` e `/caminho/para/o/hadoop` pelos caminhos corretos em seu sistema.

### 1. *Usando o Hadoop com o Apache Spark*

Para usar o Hadoop com o Apache Spark, basta definir a variável de ambiente `HADOOP_CONF_DIR` para apontar para a pasta de configuração do Hadoop em seu sistema. Você pode fazer isso adicionando a seguinte linha ao seu arquivo `~/ .bashrc`:

```
bash export HADOOP_CONF_DIR=/caminho/para/a/pasta/de/
configuração/do/hadoop
```

Certifique-se de substituir `/caminho/para/a/pasta/de/configuração/do/hadoop` pelo caminho correto no sistema.

Com a variável de ambiente `HADOOP_CONF_DIR` definida corretamente, o Apache Spark usará a configuração apropriada do Hadoop para executar suas tarefas. Você pode usar o Apache Spark com o Hadoop em seu aplicativo Streamlit como faria normalmente.

## Contribuindo

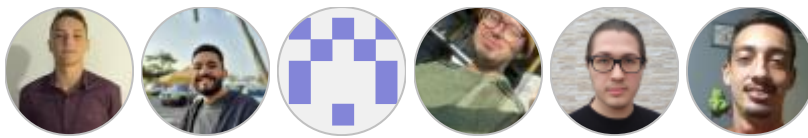
Para contribuir com este projeto, siga estas etapas:

1. Faça um fork do repositório.
2. Crie um branch com suas alterações (`git checkout -b my-feature`).
3. Faça commit das suas alterações (`git commit -m 'Add some feature'`).
4. Faça push para o branch (`git push origin my-feature`).
5. Crie uma nova Pull Request.

## Contribuidores

Agradecemos às seguintes pessoas que contribuíram para este projeto:





## Deploy do WebApp na streamlit.io

Para fazer o deploy do WebApp na streamlit.io, siga estas etapas:

Deploy do WebApp na streamlit.io (clique para expandir)

# Etapa 1: Preparação ----- \* [ ] **\*\*Requisitos de sistema\*\***: Verificar se você possui os requisitos mínimos de sistema para executar a Streamlit.io localmente ou se precisa usar um servidor externo. \* [ ] **\*\*Instalação do Streamlit\*\***: Instalar o Streamlit em seu ambiente local ou servidor externo seguindo as instruções do guia de instalação do Streamlit. \* [ ] **\*\*Configuração do ambiente\*\***: Configurar um ambiente virtual para o seu projeto usando uma ferramenta como o virtualenv ou o conda. # Etapa 2: Desenvolvimento do WebApp ----- \* [ ] **\*\*Criação do WebApp\*\***: Desenvolver o WebApp usando a biblioteca Streamlit e qualquer outro pacote ou biblioteca necessário para o seu projeto. \* [ ] **\*\*Testes locais\*\***: Executar testes locais para garantir que o WebApp esteja funcionando corretamente em seu ambiente local. # Etapa 3: Configuração do repositório ----- \* [ ] **\*\*Criação do repositório\*\***: Criar um repositório Git para o seu projeto em uma plataforma como o GitHub ou o GitLab. \* [ ] **\*\*Estrutura do projeto\*\***: Organizar o projeto em uma estrutura de diretórios adequada, incluindo um arquivo `requirements.txt` para listar todas as dependências do projeto. # Etapa 4: Implantação na Streamlit.io ----- \* [ ] **\*\*Configuração do ambiente na Streamlit.io\*\***: Criar uma conta na Streamlit.io e configurar um novo projeto/aplicativo na plataforma. \* [ ] **\*\*Conexão ao repositório\*\***: Conectar o seu repositório Git à Streamlit.io para que ele possa acessar o código do seu projeto. \* [ ] **\*\*Configuração de variáveis de ambiente\*\***: Configurar as variáveis de ambiente necessárias para o seu projeto, como chaves de API, informações de banco de dados, etc. \* [ ] **\*\*Configuração de build e deploy\*\***: Configurar as opções de build e deploy na Streamlit.io, como definir o comando de build e a pasta de implantação. \* [ ] **\*\*Iniciar o deploy\*\***: Iniciar o processo de deploy na Streamlit.io e aguardar a conclusão. # Etapa 5: Teste e Monitoramento ----- \* [ ] **\*\*Teste do WebApp implantado\*\***: Testar o WebApp implantado para garantir que ele esteja funcionando corretamente na Streamlit.io. \* [ ] **\*\*Monitoramento contínuo\*\***: Configurar ferramentas de monitoramento para acompanhar o desempenho do seu WebApp e detectar quaisquer problemas ou erros. # Etapa 6: Atualizações e Manutenção ----- \* [ ] **\*\*Atualizações contínuas\*\***: Realizar atualizações e melhorias no seu WebApp conforme necessário, fazendo o push das alterações para o repositório conectado à

Streamlit.io. \* [ ] **\*\*Gerenciamento de dependências\*\***: Gerenciar as dependências do projeto, atualizando o arquivo `requirements.txt` conforme novos pacotes forem adicionados ou atualizados. \* [ ] **\*\*Testes e deploy contínuos\*\***: Automatizar o processo de testes e deploy, utilizando ferramentas de integração contínua, como o GitHub Actions ou o GitLab CI/CD, para garantir que as atualizações sejam testadas e implantadas de forma contínua. # Etapa 7: Escalonamento e Otimização ----- \* [ ] **\*\*Gerenciamento de tráfego\*\***: Implementar estratégias de gerenciamento de tráfego, como balanceamento de carga, para lidar com um maior número de usuários e garantir a escalabilidade do seu WebApp. \* [ ] **\*\*Otimização de desempenho\*\***: Identificar e otimizar partes do seu WebApp que possam estar causando lentidão ou gargalos de desempenho. \* [ ] **\*\*Otimização de recursos\*\***: Avaliar o uso de recursos do seu WebApp e fazer ajustes para otimizar o consumo de CPU, memória e outros recursos. # Etapa 8: Monitoramento e Atualizações ----- \* [ ] **\*\*Monitoramento de desempenho\*\***: Continuar monitorando o desempenho do seu WebApp para identificar quaisquer problemas ou gargalos e tomar as medidas corretivas necessárias. \* [ ] **\*\*Atualizações de segurança\*\***: Manter-se atualizado com as atualizações de segurança e patches para as bibliotecas e pacotes utilizados no seu WebApp e aplicá-los regularmente. \* [ ] **\*\*Aprimoramentos contínuos\*\***: Continuar aprimorando o seu WebApp com base no feedback dos usuários, identificando áreas de melhoria e implementando novos recursos ou funcionalidades.

## Deploy do WebApp no Heroku

Para fazer o deploy do WebApp no Heroku, siga estas etapas:

Deploy do WebApp no Heroku (clique para expandir)

# Etapa 1: Preparação ----- \* [ ] **\*\*Requisitos de sistema\*\***: Verificar se você possui os requisitos mínimos de sistema para executar o Streamlit e o Heroku localmente ou se precisa usar um servidor externo. \* [ ] **\*\*Conta no Heroku\*\***: Criar uma conta no Heroku, caso ainda não tenha uma. \* [ ]

**\*\*Instalação do Heroku CLI\*\***: Instalar a interface de linha de comando do Heroku (Heroku CLI) em seu ambiente local, seguindo as instruções do guia de instalação do Heroku CLI. # Etapa 2: Desenvolvimento do WebApp ----- \* [ ]

**\*\*Criação do WebApp\*\***: Desenvolver o WebApp utilizando a biblioteca Streamlit e quaisquer outras bibliotecas necessárias para o seu projeto. \* [ ] **\*\*Testes locais\*\***: Executar testes locais para garantir que o WebApp esteja funcionando corretamente em seu ambiente local. # Etapa 3: Configuração do repositório -----

\* [ ] **\*\*Criação do repositório\*\***: Criar um repositório Git para o seu projeto em uma plataforma como o GitHub ou o GitLab. \* [ ]

**\*\*Estrutura do projeto\*\***: Organizar o projeto em uma estrutura de diretórios adequada, incluindo um arquivo `requirements.txt` para listar todas as dependências do projeto. \* [ ] **\*\*Arquivo Procfile\*\***: Criar um arquivo `Procfile` na

raiz do projeto para especificar os comandos de inicialização do WebApp.

Exemplo de conteúdo para o arquivo `Procfile`: ``arduino web: sh setup.sh && streamlit run app.py `` # Etapa 4: Implantação no Heroku ----- \* [ ]

**\*\*Criação do app no Heroku\*\***: Criar um novo aplicativo (app) no Heroku por meio do painel de controle do Heroku ou via linha de comando usando o Heroku CLI. \* [ ]

**\*\*Conexão ao repositório\*\***: Conectar o seu repositório Git ao aplicativo no Heroku para que ele possa acessar o código do seu projeto. \* [ ]

**\*\*Configuração de variáveis de ambiente\*\***: Configurar as variáveis de ambiente necessárias para o seu projeto, como chaves de API, informações de banco de dados, etc., utilizando o painel de controle do Heroku ou o Heroku CLI. # Etapa 5: Configuração do Heroku Buildpacks ----- \* [ ]

**\*\*Configuração do Buildpack do Python\*\***: Configurar o Buildpack do Python no aplicativo do Heroku para permitir que o Heroku instale as dependências do projeto. \* [ ]

**\*\*Configuração do Buildpack do Streamlit\*\***: Adicionar um segundo Buildpack para o Streamlit, permitindo que o Heroku execute o WebApp corretamente. Exemplo de comandos para configurar os Buildpacks usando o Heroku CLI: ``shell heroku buildpacks:set heroku/python heroku buildpacks:add https://github.com/mixmoar/heroku-buildpack-streamlit `` # Etapa 6: Deploy do WebApp ----- \* [ ]

**\*\*Push para o repositório\*\***: Fazer o push do código do seu projeto para o repositório conectado ao aplicativo no Heroku. \* [ ]

**\*\*Deploy automático\*\***: Configurar o Heroku para realizar o deploy automático do seu WebApp sempre que houver alterações no repositório conectado. Exemplo de comando para realizar o deploy usando o Heroku CLI: ``shell git push heroku main `` \* [ ]

**\*\*Monitorar o processo de deploy\*\***: Acompanhar o processo de deploy no console do Heroku CLI ou através do painel de controle do Heroku.

# Etapa 7: Teste e Monitoramento ----- \* [ ]

**\*\*Testar o WebApp implantado\*\***: Acessar o WebApp implantado no Heroku e realizar testes para garantir que ele esteja funcionando corretamente na plataforma. \* [ ]

**\*\*Monitoramento contínuo\*\***: Configurar ferramentas de monitoramento para acompanhar o desempenho do seu WebApp e detectar quaisquer problemas ou erros. # Etapa 8: Atualizações e Manutenção ----- \* [ ]

**\*\*Atualizações contínuas\*\***: Realizar atualizações e melhorias no seu WebApp conforme necessário, fazendo o push das alterações para o repositório conectado ao Heroku. \* [ ]

**\*\*Gerenciamento de dependências\*\***: Gerenciar as dependências do projeto, atualizando o arquivo `requirements.txt` conforme novos pacotes forem adicionados ou atualizados. \* [ ]

**\*\*Testes e deploy contínuos\*\***: Automatizar o processo de testes e deploy, utilizando ferramentas de integração contínua, como o GitHub Actions ou o GitLab CI/CD, para garantir que as atualizações sejam testadas e implantadas de forma contínua. [![Deploy](https://www.herokucdn.com/deploy/button.svg)](https://heroku.com/deploy)

# Deploy do WebApp no render

Para fazer o deploy do WebApp no render, siga estas etapas:

Deploy do WebApp no render (clique para expandir)

# Etapa 1: Preparação ----- \* [ ] **\*\*Requisitos de sistema\*\***: Verificar os requisitos de sistema do Render para garantir compatibilidade com o seu projeto. \* [ ] **\*\*Conta no Render\*\***: Criar uma conta no Render, caso ainda não possua uma. \* [ ] **\*\*Instalação do Render CLI\*\***: Instalar a interface de linha de comando do Render (Render CLI) em seu ambiente local, seguindo as instruções do guia de instalação do Render CLI. \* [ ] **\*\*Configuração do acesso SSH\*\***: Configurar o acesso SSH para autenticação e gerenciamento de chaves. # Etapa 2: Desenvolvimento do WebApp ----- \* [ ] **\*\*Criação do WebApp\*\***: Desenvolver o WebApp utilizando a biblioteca Streamlit e quaisquer outras bibliotecas necessárias para o seu projeto. \* [ ] **\*\*Testes locais\*\***: Executar testes locais para garantir que o WebApp esteja funcionando corretamente em seu ambiente local. # Etapa 3: Configuração do repositório ----- \* [ ] **\*\*Criação do repositório\*\***: Criar um repositório Git para o seu projeto em uma plataforma como o GitHub ou o GitLab. \* [ ] **\*\*Estrutura do projeto\*\***: Organizar o projeto em uma estrutura de diretórios adequada, incluindo um arquivo `requirements.txt` para listar todas as dependências do projeto. \* [ ] **\*\*Arquivo `render.yaml`\*\***: Criar um arquivo `render.yaml` na raiz do projeto para configurar as opções de build e deploy. Exemplo de conteúdo para o arquivo `render.yaml`:  
yaml ``yaml version: 1 buildCommand: python -m pip install -r requirements.txt startCommand: streamlit run app.py `` # Etapa 4: Implantação no Render ----- \* [ ] **\*\*Criação do serviço Render\*\***: Criar um novo serviço Render por meio do painel de controle do Render ou via linha de comando usando o Render CLI. \* [ ] **\*\*Conexão ao repositório\*\***: Conectar o repositório Git ao serviço Render para permitir que ele acesse o código do projeto. \* [ ] **\*\*Configuração das variáveis de ambiente\*\***: Configurar as variáveis de ambiente necessárias para o projeto, como chaves de API e informações de banco de dados, utilizando o painel de controle do Render ou o Render CLI. # Etapa 5: Configuração avançada ----- \* [ ] **\*\*Customização do ambiente\*\***: Personalizar o ambiente de implantação no Render, como especificar a versão do Python, as configurações de CPU e memória, entre outras opções disponíveis. \* [ ] **\*\*Configuração do domínio personalizado\*\***: Configurar um domínio personalizado para o WebApp usando o painel de controle do Render ou o Render CLI, para que o WebApp seja acessível em um domínio personalizado, como `www.meudominio.com`. # Etapa 6: Configuração do banco de dados e armazenamento ----- \* [ ] **\*\*Integração com banco de dados\*\***: Configurar a integração com um banco de dados, como PostgreSQL, MySQL ou MongoDB, para armazenar e acessar dados a partir do seu WebApp. Aqui está um exemplo de como configurar o PostgreSQL: 1. Crie uma instância do banco de dados PostgreSQL em um serviço como o Amazon RDS ou o

ElephantSQL. 2. Obtenha as informações de conexão, incluindo host, porta, nome do banco de dados, nome de usuário e senha. 3. Adicione essas informações como variáveis de ambiente no Render ou em um arquivo `.env`: `makefile DB_HOST= DB_PORT= DB_NAME= DB_USER= DB_PASSWORD=` 4. No código do seu WebApp, utilize uma biblioteca de conexão ao banco de dados, como SQLAlchemy, para estabelecer a conexão e executar consultas. Exemplo de código Python para configurar a conexão ao PostgreSQL usando SQLAlchemy: `python import os from sqlalchemy import create_engine db_host = os.environ.get('DB_HOST') db_port = os.environ.get('DB_PORT') db_name = os.environ.get('DB_NAME') db_user = os.environ.get('DB_USER') db_password = os.environ.get('DB_PASSWORD') # Configuração da conexão ao PostgreSQL db_url = f'postgresql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}' engine = create_engine(db_url) # Exemplo de consulta result = engine.execute("SELECT * FROM table_name") for row in result: print(row)` # Etapa 7: Teste e Monitoramento ----- \* [ ] **\*\*Testar o WebApp implantado\*\***: Acessar o WebApp implantado no Render e realizar testes para garantir que todas as funcionalidades estejam funcionando corretamente. \* [ ] **\*\*Monitoramento avançado\*\***: Configurar ferramentas de monitoramento avançado, como o Prometheus e o Grafana, para acompanhar o desempenho do WebApp, métricas de utilização de recursos e detecção de erros. # Etapa 8: Atualizações e Manutenção ----- \* [ ] **\*\*Atualizações contínuas\*\***: Realizar atualizações e melhorias no WebApp conforme necessário, fazendo o push das alterações para o repositório conectado ao Render. \* [ ] **\*\*Gerenciamento de dependências\*\***: Gerenciar as dependências do projeto, atualizando o arquivo `requirements.txt` conforme novos pacotes forem adicionados ou atualizados. \* [ ] **\*\*Testes e deploy contínuos\*\***: Automatizar o processo de testes e deploy, utilizando ferramentas de integração contínua, como o GitHub Actions ou o GitLab CI/CD, para garantir que as atualizações sejam testadas e implantadas de forma contínua.

## Deploy do WebApp no PythonAnywhere

Para fazer o deploy do WebApp no PythonAnywhere, siga estas etapas:

Deploy do WebApp no PythonAnywhere (clique para expandir)

# Etapa 1: Preparação ----- Antes de começar o deploy do seu WebApp no PythonAnywhere, é importante realizar algumas etapas de preparação: 1. Crie uma conta no PythonAnywhere, caso ainda não tenha uma. 2. Familiarize-se com a plataforma, navegando pela documentação e explorando os recursos oferecidos. 3. Verifique os requisitos mínimos de sistema do PythonAnywhere para garantir a compatibilidade com o seu projeto. # Etapa 2: Desenvolvimento do WebApp ----- 1. Desenvolva o seu WebApp utilizando a biblioteca Streamlit e outras bibliotecas relevantes para o seu projeto. 2. Realize testes locais para garantir o funcionamento correto do WebApp no seu ambiente de

desenvolvimento. # Etapa 3: Preparação do código para o deploy

----- Antes de fazer o deploy do WebApp no PythonAnywhere, é necessário preparar o código adequadamente: 1. Crie um repositório Git para o seu projeto em uma plataforma como o GitHub ou o GitLab. 2. Organize a estrutura do projeto de forma clara e legível, separando arquivos e pastas conforme necessário. 3. Certifique-se de ter um arquivo `requirements.txt` que liste todas as dependências do projeto. # Etapa 4: Configuração do ambiente

no PythonAnywhere ----- Agora é hora de configurar o ambiente necessário para o deploy do seu WebApp no PythonAnywhere: 1. Faça login na sua conta do PythonAnywhere. 2. Crie um novo "Web app" no PythonAnywhere, especificando o tipo de aplicativo como "Flask". 3. Escolha a opção "Manual configuration" para ter maior controle sobre as configurações do seu aplicativo. 4. Configure as informações do seu WebApp, como o nome do aplicativo, o diretório raiz do aplicativo e o tipo de aplicativo WSGI. # Etapa 5:

Configuração do Virtualenv ----- O uso de um ambiente virtual é recomendado para isolar as dependências do seu WebApp no PythonAnywhere: 1. Acesse o painel de controle do PythonAnywhere e navegue até a seção "Virtualenvs". 2. Crie um novo ambiente virtual especificando o diretório de trabalho e a versão do Python. 3. Ative o ambiente virtual recém-criado. 4. Instale as dependências do projeto utilizando o comando `pip install -r requirements.txt`. # Etapa 6: Configuração do WebApp no PythonAnywhere

----- Agora é necessário configurar o seu WebApp no PythonAnywhere para que ele possa ser executado corretamente: 1. No painel de controle do PythonAnywhere, vá para a seção "Code" e vincule o seu repositório Git ao seu aplicativo. 2. Configure o arquivo `wsgi.py` para importar o aplicativo correto do seu código. 3. Faça as configurações necessárias para garantir que o aplicativo seja corretamente executado pelo PythonAnywhere. # Etapa 7:

Configuração do banco de dados (opcional) ----- Se o seu WebApp requer um banco de dados, você precisa configurar a integração adequada no PythonAnywhere: 1. Escolha um provedor de banco de dados compatível com o PythonAnywhere, como o Amazon RDS ou o ElephantSQL. 2. Crie uma instância do banco de dados no provedor escolhido e obtenha as informações de conexão, como host, porta, nome do banco de dados, nome de usuário e senha. 3. No seu código Python, utilize bibliotecas como SQLAlchemy para estabelecer a conexão com o banco de dados e executar consultas. 4. Certifique-se de configurar as variáveis de ambiente necessárias no PythonAnywhere para armazenar as informações de conexão do banco de dados. # Etapa 8: Configuração do roteamento de tráfego (opcional)

----- Se você deseja utilizar um domínio personalizado para o seu WebApp no PythonAnywhere, siga estas # etapas: 1. Registre um domínio personalizado em um provedor de registro de domínios. 2. No provedor de registro de domínios, configure os registros DNS para apontar para os servidores DNS do PythonAnywhere. 3. No painel de controle do PythonAnywhere, acesse a seção "Web" e clique em "Add a new web app". 4. Siga

as instruções fornecidas pelo PythonAnywhere para configurar o roteamento de tráfego do domínio personalizado para o seu WebApp. # Etapa 9: Teste e monitoramento ----- 1. Acesse o seu WebApp no PythonAnywhere e verifique se tudo está funcionando conforme o esperado. 2. Realize testes extensivos para garantir que todas as funcionalidades estejam operando corretamente. 3. Configure ferramentas de monitoramento, como logs do PythonAnywhere ou ferramentas externas, para acompanhar o desempenho e detectar erros ou problemas no seu WebApp. # Etapa 10: Atualizações e manutenção ----- 1. Faça o upload de atualizações e melhorias para o seu repositório Git. 2. Utilize o painel de controle do PythonAnywhere para sincronizar o seu aplicativo com o repositório Git. 3. Realize testes e deploy contínuos utilizando ferramentas como o GitHub Actions ou o GitLab CI/CD para garantir que as atualizações sejam testadas e implantadas de forma contínua. 4. Mantenha-se atualizado com as atualizações de segurança e correções de bugs, atualizando regularmente as dependências do seu projeto.

## **Licença**

Este projeto está licenciado sob a Licença MIT - consulte o arquivo LICENSE para obter detalhes.