AULA 1: CRIE AMBIENTES REAIS COM DOCKER E DÊ FIM AO “NA MINHA MÁQUINA FUNCIONA”

- ➔ **Introdução ao DevOps:** DevOps é mais do que ferramentas; envolve cultura e colaboração entre equipes de desenvolvimento e operações.

- ➔ **Objetivo da Aula:** Aprender a criar uma imagem Docker de uma aplicação simples (API em Python) e resolver o problema do "na minha máquina funciona"

- ➔ **Ambiente Virtual:** A criação de um ambiente virtual em Python é essencial para isolar dependências de projetos.

- ➔ **Docker:** O Docker permite empacotar aplicações e suas dependências em contêineres, garantindo que funcionem em qualquer ambiente.

- ➔ **Contêineres:** são mais leves e eficientes do que máquinas virtuais, pois compartilham o mesmo sistema operacional.

- ➔ **Dockerfile:**
  - ◆ É um arquivo que define como a imagem Docker será construída, incluindo a base (imagem) e as instruções para instalar dependências e rodar a aplicação.
  - ◆ A instrução `FROM` é usada para especificar a imagem base (ex: `FROM python:3.13.4-alpine`).

- ➔ **.dockerignore:** Semelhante ao `.gitignore`, este arquivo é usado para ignorar arquivos e pastas que não devem ser incluídos na imagem Docker, como `venv` e `__pycache__`.

- ➔ **Construção e Execução da Imagem:**
  - ◆ O comando `docker build -t nome-da-imagem .` é usado para construir a imagem.
  - ◆ O comando `docker run -p 8000:8000 nome-da-imagem` é utilizado para executar a aplicação dentro do contêiner.

Questionário da aula com respostas:

O que é Docker?
Uma ferramenta de conteinerização

Qual é a principal vantagem de usar contêineres em vez de máquinas virtuais?

Contêineres compartilham o mesmo kernel do sistema operacional

O que é um Dockerfile?
Um arquivo que define como a imagem Docker deve ser construída

Qual comando é utilizado para construir uma imagem Docker a partir de um Dockerfile?
docker build

O que faz o comando docker run -p 8000:8000?
Executa um contêiner e mapeia a porta 8000 do contêiner para a porta 8000 do host

---

Passos da aula 1:

- Baixar o repositório do projeto https://github.com/guilhermeonrails/ellis
- Abrir no VSCode
- Abrir o arquivo ReadMe
- Criar um ambiente virtual:

```
python -m venv ./venv
```

- Ativar o ambiente virtual:
  1. Se for o primeiro uso no pc: abrir o terminal no modo administrador e executar

     ```
     Set-ExecutionPolicy RemoteSigned
     ```

  2. Depois só precisa rodar

     ```
     venv\Scripts\activate
     ```

- Instalar as dependências rodando

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

- Executar a aplicação rodando

```
uvicorn app:app --reload
```

- **Clicar no link gerado no terminal e acrescentar /docs no fim para rodar a API localmente**
- **Baixar o Docker (versão AMD64, pois meu sistema roda x64) e instalar**
- **Criar o arquivo Dockerfile na pasta do projeto no VSCode**
- **Criar o arquivo .dockerignore com o seguinte conteúdo:**

```
venv
__pycache__
```

- **Realizar o build da imagem rodando no terminal:**

```
docker build -t api .
```

- **No momento do build, pode escolher o nome que quiser para colocar onde tem <api>, o "." indica que o contexto de build é o diretório atual**
- **Abrindo o Docker e clicando em Images, vai aparecer a imagem criada**
- **Para rodar o container:**

```
docker run -p 8000:8000 api
```

- **Clicar no link gerado e substituir por localhost:8000/docs para ver a aplicação rodando**
- **Abrir o Docker e clicar em Containers para visualizar também por lá**

Exercício: Criando uma Imagem Docker para uma Aplicação Simples

Objetivo: Criar uma imagem Docker para uma aplicação simples em Python que simula uma API de gerenciamento de tarefas. Passos:

1. Crie um novo diretório para o seu projeto chamado task-manager
2. Dentro do diretório task-manager, crie um arquivo chamado app.py com o seguinte conteúdo:

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/tasks")
def read_tasks():
    return [{"task": "Learn Docker"}, {"task": "Build a project"}]

@app.post("/tasks")
def create_task(task: str):
    return {"task": task, "status": "created"}
```

3. Crie um arquivo `requirements.txt` no mesmo diretório com o seguinte conteúdo:

```
fastapi
uvicorn
```

4. Crie um arquivo `Dockerfile` no diretório `task-manager` com as instruções necessárias para construir a imagem Docker. Lembre-se de usar uma imagem base do Python e incluir os comandos para instalar as dependências e executar a aplicação:

```
# Imagem base
FROM python:3.10

# Define diretório de trabalho dentro do container
WORKDIR /app

# Copia os arquivos do projeto para o container
COPY . .

# Instala as dependências
RUN pip install --no-cache-dir -r requirements.txt

# Expõe a porta usada pela aplicação
EXPOSE 8000

# Comando para rodar o servidor
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

5. Crie um arquivo `.dockerignore` para ignorar arquivos desnecessários, como `__pycache__` e outros que não precisam ser incluídos na imagem:

```
__pycache__
*.pyc
venv
```

6. Construa a imagem Docker usando o comando `docker build -t task-manager .` no terminal, dentro do diretório `task-manager` (lembre de abrir o Docker para funcionar)

7. Execute a imagem Docker usando o comando `docker run -p 8000:8000 task-manager`

8. Teste a API acessando http://localhost:8000/tasks e http://localhost:8000/docs no seu navegador

EXTRA: O que é a FastAPI?

FastAPI é um *framework* em Python para criar **APIs** modernas, rápidas e fáceis de usar. Ela serve para criar aqueles "serviços" que ficam escutando requisições (GET, POST, DELETE, etc.)

"API" — Application Programming Interface (Interface de Programação de Aplicações)

Você cria rotas como:

- GET /tasks → busca tarefas
- POST /tasks → adiciona tarefa
- DELETE /tasks/nome → deleta tarefa

Essas rotas podem ser acessadas por navegador, app, frontend, Postman, etc.

Vantagens da "FastAPI" :

- Rápida (usa o mesmo motor do Starlette e Uvicorn)
- Fácil de escrever e fácil de testar
- Gera automaticamente uma interface interativa em /docs

- Suporta validações de dados com pydantic
- Ótima para projetos com Docker, microserviços, e sistemas modernos