📘 Aula - Dockerizando uma aplicação

Baixar código base dessa aula

Instalação

Primeiramente, você precisa ter instalado em seu sistema o Docker e o Docker Compose. É só seguir a documentação conforme seu sistema operacional.

- Docker Engine
- Docker Compose
- Docker Desktop

> **Dica**
>
> O Docker Desktop agora está disponível para sistemas Linux. O Docker Desktop inclui o Docker

Engine, o cliente Docker CLI e o Docker Compose, facilitando o uso das ferramentas por uma interface gráfica.

Dockerfile

O Dockerfile nos permite criar imagens dos containers. Esse arquivo vai ser a "receita de bolo" que o Docker vai executar para formar o ambiente encapsulado.

O que é imagem? Em resumo, um ambiente encapsulado e pronto para ser replicado em qualquer lugar.

A estrutura desse arquivo deve seguir uma ordem para a execução das tarefas, a cada passo descrito, uma *layer* da imagem é formada e, ao final, o Docker encapsula todas as layers, formando uma imagem. A estrutura dos comandos deve ser: `INSTRUÇÃO argumento`.

Crie o arquivo Dockerfile na raiz da sua aplicação

Importante!

Insira a mesma versão do Python que está instalada em sua máquina, para que não haja conflitos entre as diferentes versões dos programas instalados.

Copiar para àrea de transferência

```
# Dockerfile

# fazendo o pull da imagem oficial no Docker Hu
FROM python:3.10

# setando variáveis de ambiente
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1


# executando a instalação dos pacotes

# definindo o diretório de trabalho no contêine
WORKDIR /code
```

```
# copiando todos os arquivos para o diretório d
COPY . /code/


# atualizando o pip e instalando requerimentos
RUN pip install -U pip
RUN pip install -r requirements.txt
```

Docker Compose

O docker-compose vai servir para a orquestração de containers, ou seja, quando temos mais de um container, vamos usar o docker-compose para realizar o gerenciamento dos contêineres e configurações de comunicação entre eles.

Crie o arquivo docker-compose.yml na raiz da sua aplicação

Copiar para área de transferência

```
# docker-compose.yml
services:
    db:
        image: postgres:latest #(1)
        env_file: .env #(2)
        ports: #(3)
            - 5432:5432
        volumes: #(4)
           - pgdata:/var/lib/postgresql/data

    migration: #(5)
        build: . #(6)
        env_file: .env #(3)
        command: #(7)
            bash -c 'while !</dev/tcp/db/5432; 
        volumes: #(4)
            - .:/code
        depends_on: #(8)
            - db
```

```
    web:
        build: . #(6)
        env_file: .env #(3)
        command: #(7)
            bash -c 'while !</dev/tcp/db/5432; 
        volumes: #(4)
            - .:/code
        stdin_open: true #(9)
        tty: true #(10)
        ports: #(3)
            - 8000:8000

        depends_on: #(8)
            - db
            - migration

volumes: #(4)
    pgdata:
```

Aqui existem alguns pontos importantes:

1. `image` - aqui definimos qual imagem utilizaremos

2. `env_file` - definição do caminho para as variáveis de ambiente

3. `ports` - mapeamento da porta do serviço com a porta da máquina

4. `volumes` - os volumes fornecem a capacidade de conectar caminhos específicos do sistema de arquivos do contêiner de volta à máquina host.
    - **volumes nomeados** - o Docker mantém a localização física no disco e indicamos apenas um nome para o volume, nesse caso, nosso banco de dados será salvo em nosso contêiner no caminho `/var/lib/postgresql/data`

    - **volumes de ligação** - aqui controlamos o ponto de montagem exato no host, ou seja, faz uma ponte de ligação do host com o contêiner, ou seja, todas as alterações que fizermos no host, serão refletidas no contêiner imediatamente.

5. `migration` - aqui estamos definindo um serviço para executar as migrations para nós, para não precisarmos entrar no contêiner manualmente para fazer essa operação.

6. `build` - caminho onde está o Dockerfile para fazer o build

7. `command` - opcional, podemos passar um comando para inicialização

- O PostgreSQL demora um certo tempo para aceitar conexões, e isso é independente do tempo de montagem do container. Portanto, se não houver nenhuma trava, o comando de execução de migrations pode ser rodado antes do banco estar pronto para aceitar conexões. Esse script de bash serve justamente para fazer essa trava.

8. `depends_on` - lista dos serviços que este serviço depende para iniciar, apenas depende que o contêiner seja montado, não que ele esteja pronto.

9. `stdin_open` - equivalente a docker run -i, para abrir o contêiner em modo interativo

10. `tty` - equivalente a docker run -t, para conectar nosso terminal com o contêiner

---

Outras configurações

Variáveis de ambiente

Você pode ter percebido que nossos contêineres possuem uma indicação para variáveis de ambiente `env_file`, onde precisamos definir as variáveis de ambiente de nossa aplicação.

A imagem do PostgreSQL usa várias variáveis de ambiente. A única variável necessária é `POSTGRES_PASSWORD`, o restante é opcional.

Crie um arquivo `.env` na raiz da sua aplicação

Copiar para àrea de transferência

```
#.env
POSTGRES_DB=project
POSTGRES_USER=user
POSTGRES_PASSWORD=password
```

Configurando PostgreSQL

Para que nossa aplicação possa funcionar corretamente, precisamos definir algumas configurações extras.

Primeiramente, como vamos trabalhar com o banco de dados PostgreSQL, devemos alterar nossas configurações em `settings.py`

Copiar para área de transferência

```
# project_base/settings.py
import os


if os.environ.get("TEST"):
    DATABASES = {
        "default": {
            "ENGINE": "django.db.backends.sqlit
            "NAME": BASE_DIR / "db.sqlite3",
        }
    }
else:
    DATABASES = {
        "default": {
            "ENGINE": "django.db.backends.postg
            "NAME": os.environ.get("POSTGRES_DB
            "USER": os.environ.get("POSTGRES_US
            "PASSWORD": os.environ.get("POSTGRE
            "HOST": "db",
            "PORT": 5432,
```

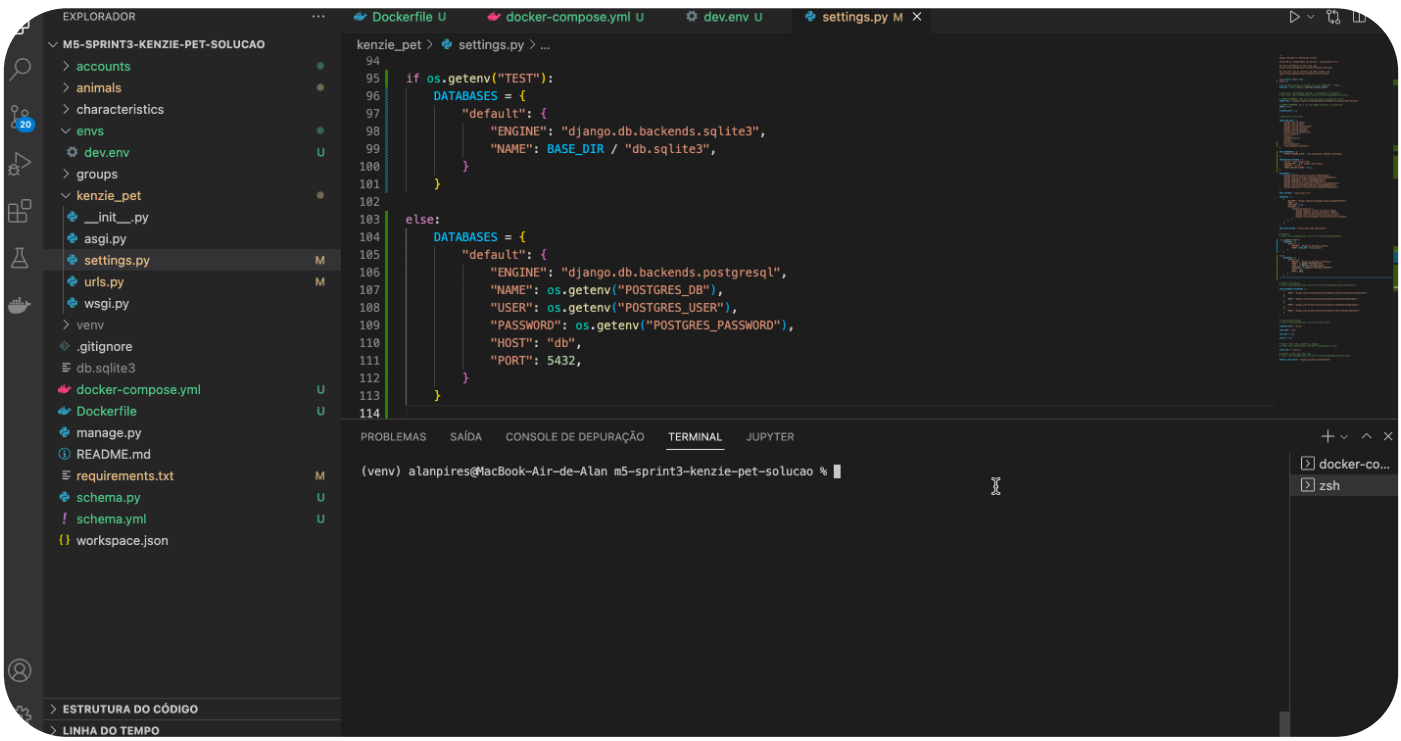
```
        }
    }
```

Aqui definimos uma condicional, para que possamos rodar nossos testes ou nossa aplicação fora dos contêineres, para isso basta executar.

Copiar para àrea de transferência

```
# no terminal

TEST=TEST ./manage.py runserver
```



Subindo nossos contêineres

Para vermos a mágica acontecer e vermos todos os nossos contêineres subindo, podemos executar o seguinte comando:

Copiar para área de transferência

```
docker-compose up
```

Descendo nossos contêineres

Para descer todos os contêineres que estejam em execução, basta executar:

Copiar para área de transferência

```
docker-compose down
```

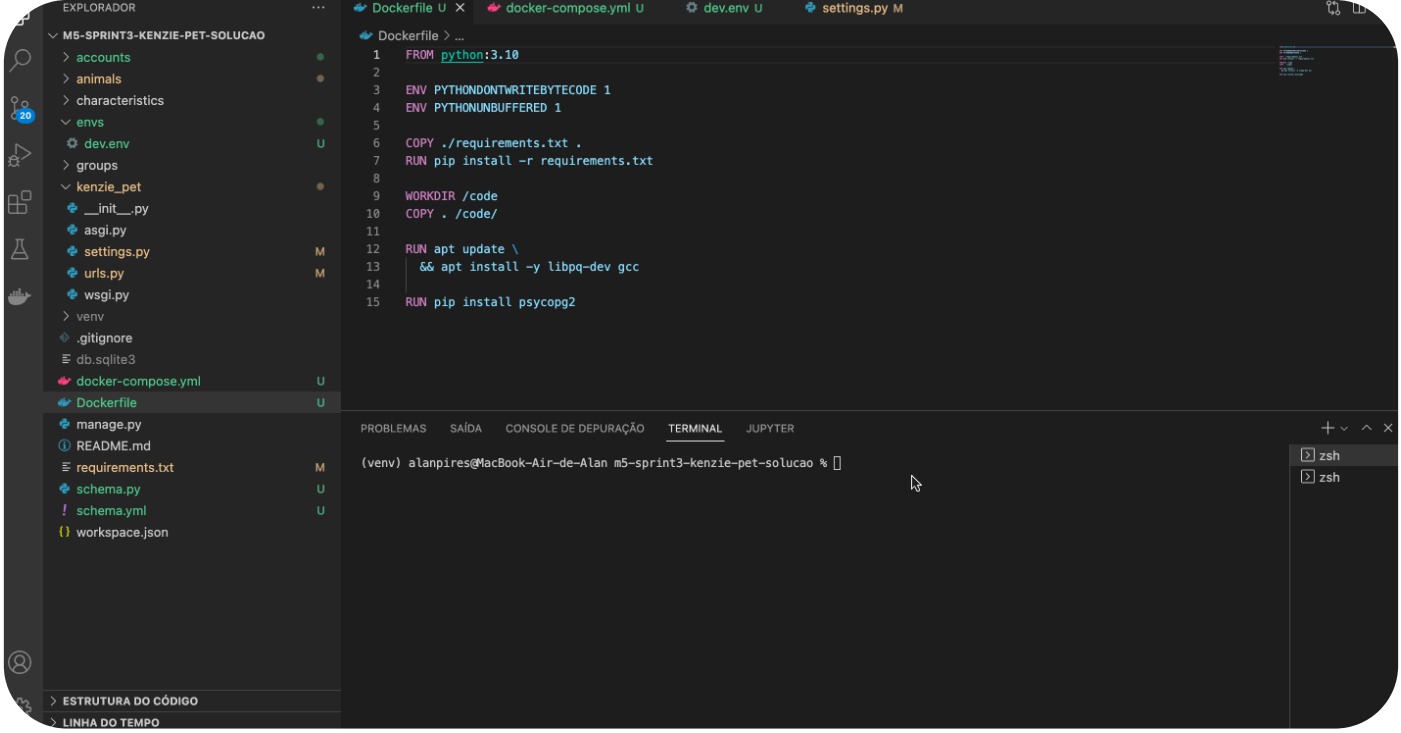
Debugando com ipdb

Se você for debugar com o `ipdb.set_trace()` você precisará abrir outra aba no terminal e fazer um attach no container web. Você pode fazer isso com o comando:

Copiar para área de transferência

```
docker attach <container_web_id>
```

Testando



Referências!

[Using Docker Compose | Docker Docs]

[Métodos de String - Docs | Python]

[Postgres com Docker + Django | Youtube]