Dominando Docker

Uma Abordagem Prática para Desenvolvimento e Implantação de Aplicações

O Docker é uma plataforma de código aberto que automatiza o desenvolvimento, a entrega e a execução de aplicações em contêineres.

Essa tecnologia permite que desenvolvedores empacotem suas aplicações e suas dependências em contêineres, que podem ser executados em qualquer ambiente, desde desenvolvimento até produção.

A principal vantagem do Docker é a consistência em diferentes ambientes, mitigando problemas comuns do tipo "funciona na minha máquina".

1. Conceitos Fundamentais

1.1. O Que São Imagens e Contêineres?

- **Imagem**: Um arquivo imutável que contém tudo o que é necessário para executar uma aplicação, incluindo o código-fonte, bibliotecas e dependências. As imagens são construídas a partir de um Dockerfile.
- **Contêiner**: Uma instância de uma imagem. Os contêineres são isolados uns dos outros, mas podem se comunicar entre si através de redes. Eles são criados a partir de imagens e podem ser iniciados, parados, movidos e deletados.

1.2. O Papel do Dockerfile

O Dockerfile é um arquivo de texto que contém uma série de instruções que o Docker utiliza para construir uma imagem. Ele descreve como a imagem deve ser criada e quais dependências devem ser incluídas.

1.3. Estrutura de Diretórios do Projeto

Para ilustrar o uso do Docker, criaremos uma aplicação simples utilizando Python e Flask. A estrutura de diretórios será a seguinte:

```
my_docker_app/
├── app.py
├── requirements.txt
└── Dockerfile
```

2. Instalando o Docker

Antes de começarmos a construir nossa aplicação, você precisa instalar o Docker em sua máquina.

As instruções a seguir variam de acordo com o sistema operacional:

- **Windows**: Baixe e instale o Docker Desktop.
- **macOS**: Baixe e instale o Docker Desktop.
- **Linux (Ubuntu)**: Execute os seguintes comandos no terminal:

bash
```
sudo apt update
sudo apt install docker.io
sudo systemctl start docker
sudo systemctl enable docker
```

Após a instalação, verifique se o Docker está funcionando corretamente com o comando:

bash
```
docker –version
```

3. Criando uma Aplicação Docker

3.1. Desenvolvendo a Aplicação

Crie um arquivo chamado app.py no diretório my_docker_app e adicione o seguinte código:

python
```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World!"

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

Neste código, utilizamos o Flask para criar um servidor web simples que responde com "Hello, World!" ao acessar a raiz do aplicativo.

3.2. Definindo as Dependências

Crie um arquivo chamado requirements.txt no mesmo diretório e insira as dependências necessárias:

makefile
```
Flask==2.0.1
```

3.3. Criando o Dockerfile

Crie um arquivo chamado Dockerfile no diretório my_docker_app e adicione o seguinte conteúdo:

dockerfile
```
# Usa uma imagem base do Python
FROM python:3.9-slim

# Define o diretório de trabalho
WORKDIR /app

# Copia o arquivo de requisitos e a aplicação para o contêiner
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copia o código da aplicação
COPY app.py app.py

# Expõe a porta 5000
EXPOSE 5000

# Comando para executar a aplicação
CMD ["python", "app.py"]
```

3.4. Construindo a Imagem Docker

No terminal, navegue até o diretório my_docker_app e execute o seguinte comando para construir a imagem Docker:

bash
```
docker build -t my_flask_app .
```

- -t my_flask_app: Define um nome (tag) para a imagem.
- .: Indica que o Dockerfile está localizado no diretório atual.

4. Executando a Aplicação em um Contêiner

4.1. Iniciando o Contêiner

Após a construção da imagem, execute o seguinte comando para iniciar o contêiner:

bash
docker run -d -p 5000:5000 my_flask_app

- -d: Executa o contêiner em segundo plano (modo destacado).
- -p 5000:5000: Mapeia a porta 5000 do contêiner para a porta 5000 da máquina host.

4.2. Acessando a Aplicação

Agora você pode acessar sua aplicação em um navegador web através do endereço http://localhost:5000. Você deve ver a mensagem "Hello, World!".

5. Gerenciando Contêineres

5.1. Listando Contêineres em Execução

Para listar os contêineres em execução, utilize:

bash
docker os

5.2. Parando um Contêiner

Para parar um contêiner, utilize:

bash
docker stop <container_id>

Substitua <container_id> pelo ID do contêiner que você deseja parar.

5.3. Removendo um Contêiner

Para remover um contêiner que não está em execução, use:

bash
docker rm <container_id>

6. Recursos Avançados do Docker

6.1. Docker Compose

O Docker Compose é uma ferramenta que permite definir e executar aplicações compostas por múltiplos contêineres usando um arquivo YAML.

6.1.1. Exemplo de Uso do Docker Compose

Vamos expandir nossa aplicação Flask para incluir um banco de dados PostgreSQL. Crie um arquivo chamado docker-compose.yml no diretório my_docker_app e adicione o seguinte conteúdo:

yaml
```yaml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/mydatabase

  db:
    image: postgres:13
    restart: always
    environment:
      POSTGRES_DB: mydatabase
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

6.1.2. Executando com Docker Compose

Após criar o arquivo docker-compose.yml, execute os seguintes comandos para iniciar a aplicação:

bash
```bash
docker-compose up --build
```

Para parar os serviços, pressione CTRL + C ou use:

bash
```bash
docker-compose down
```

6.2. Persistência de Dados com Volumes

Os volumes no Docker são utilizados para armazenar dados persistentes gerados e utilizados por contêineres. Quando um contêiner é removido, seus dados podem ser mantidos em um volume.

6.2.1. Criando e Usando um Volume

No exemplo acima, o volume postgres_data é utilizado para armazenar os dados do PostgreSQL. Você pode visualizar os volumes existentes com:

bash
```
docker volume ls
```

6.3. Redes no Docker

O Docker permite a criação de redes personalizadas para contêineres, o que é útil para isolar serviços e melhorar a segurança.

6.3.1. Criando uma Rede

Você pode criar uma rede chamada my_network com o seguinte comando:

bash
```
docker network create my_network
```

Para conectar um contêiner a essa rede, use:

bash
```
docker run -d --network my_network my_flask_app
```

6.4. Dockerfile Avançado

Podemos melhorar nosso Dockerfile usando técnicas avançadas, como multi-stage builds e camadas otimizadas.

6.4.1. Exemplo de Dockerfile Otimizado

dockerfile
```
# Fase de construção
FROM python:3.9-slim AS builder

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Fase de execução
FROM python:3.9-slim

WORKDIR /app

# Copiando apenas os arquivos necessários da fase de construção
```

```
COPY --from=builder /usr/local/lib/python3.9/site-packages /usr/local/lib/python3.9/site-packages
COPY app.py .

EXPOSE 5000

CMD ["python", "app.py"]
```

Neste exemplo, utilizamos um estágio de construção separado, o que reduz o tamanho final da imagem.

7. Monitoramento e Logs

O monitoramento de contêineres é crucial em ambientes de produção. O Docker fornece uma interface simples para visualizar logs.

7.1. Visualizando Logs de um Contêiner

Para visualizar os logs de um contêiner, utilize o comando:

bash
```
docker logs <container_id>
```

Se você deseja seguir os logs em tempo real, adicione a opção -f:

bash
```
docker logs -f <container_id>
```

Conclusão

O Docker é uma ferramenta poderosa que facilita o desenvolvimento e a implantação de aplicações em contêineres. Ao adotar práticas recomendadas, como o uso de Dockerfile otimizado e Docker Compose, é possível criar ambientes de desenvolvimento consistentes e eficientes. A compreensão de conceitos como imagens, contêineres, redes e volumes é fundamental para tirar o máximo proveito dessa tecnologia. Com o conhecimento adquirido neste artigo, você está pronto para iniciar sua jornada com Docker e criar aplicações robustas e escaláveis.

EducaCiência FastCode para a comunidade