Professor: Danilo Sibov

Microsserviços | Aula 2

Laboratórios

Este módulo aborda os seguintes tópicos:

- **Laboratório 7 - Docker Compose**

Neste laboratório você aprenderá a criar um ambiente de homologação e produção utilizando o Docker Compose

Etapa 1 - Criar um Docker compose para o ambiente de homologação
Etapa 2 - Executar o Docker compose

Etapa 1 - Criar um Docker Compose para o ambiente de homologação

1. **Para criar um Docker compose, primeiramente, acesse sua máquina virtual**

2. **No terminal, digite** vim app.py

3. **Depois, coloque o código abaixo no seu arquivo.**

```
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Olá Aluno, você viu essa página {} vezes.\n'.format(count)
```

Salve o arquivo, aperte a tecla ESC, depois digite :wq!

4. **Depois de salvar o arquivo anterior, vamos configurar as bibliotecas necessárias para o nosso APP iniciar, execute no terminal:** vim requirements.txt

5. **Aperte a tecla** i **para editar o arquivo e coloque os seguintes caracteres:**

```
flask
redis
```

Salve o arquivo, aperte a tecla ESC, depois digite :wq!

6. **Depois, precisamos criar um Dockerfile, digite** vim Dockerfile

7. **Como essa é apenas uma aplicação de exemplo escrita em python, vamos utilizar a imagem base do nosso container o** python:3.7-alpine**, digite isso na primeira linha:** FROM python:3.7-alpine

```
FROM python:3.7-alpine
```

8. **Depois, vamos configurar uma workdir que é onde vai estar todo o código de nossa aplicação, digite na segunda linha:** WORKDIR /code

```
FROM python:3.7-alpine
WORKDIR /code
```

9. **Os apps escritos em python precisam de uma variável de ambiente chamada** FLASK_APP = <app> **para poder executar um app em python, digite na terceira linha:** ENV FLASK_APP=app.py

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
```

10. **Na quarta linha, vamos configurar o HOST para rodar nosso app, ou seja, vamos colocar localhost para poder funcionar o app, digite:** ENV FLASK_RUN_HOST=0.0.0.0

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
```

11. **Vamos adicionar um repositório com o parâmetro --no-cache (permite que o cache não seja indexado localmente, muito utilizado para manter os contêineres menores), na quinta linha, digite:** apk add --no-cache gcc musl-dev linux-headers

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
```

12. **Aplicações python possuem um txt nomeado como** requirements.txt **que possui os requisitos mínimos para iniciar o app, como por exemplo bibliotecas, na sexta linha digite:** COPY requirements.txt requirements.txt

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
```

13. **Agora, vamos instalar os requisitos mínimos de nosso APP, digite na sétima linha:** RUN pip install -r requirements.txt

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
```

14. **Coloque o APP para escutar na porta 5000, na oitava linha digite:** EXPOSE 5000

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
```

15. **Agora copie o resto dos arquivos para dentro do container, na nova linha, digite:** COPY . .

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
```

16. **Diga ao container qual comando iniciará o nosso APP, digite na última linha:** CMD [“flask”, “run”]

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Verifique se o seu arquivo de Dockerfile está igual ao print anexado acima

17. **Vamos iniciar a construção de nosso Docker compose**

18. **Após salvar o último arquivo, execute no terminal:** vim docker-compose.yaml

19. **Agora, vamos começar a construí-lo, na primeira linha vamos definir a versão do nosso docker compose ou versão do nosso arquivo, digite:** version: “3.9”

```
version: "3.9"
```

20. **Agora, vamos definir os serviços (containers) que serão executados por este Docker compose, na segunda linha digite:** services

```
version: "3.9"
services:
```

21. **Logo abaixo, de maneira indentada, digite** web **(nome do nosso container por assim dizer)**

```
version: "3.9"
services:
  web:
```

22. **Como nós vamos construir uma imagem, vamos mandar o docker compose construir uma imagem local e definir a porta que o container irá escutar, para isto, logo abaixo, de maneira indentada, digite:**

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
```

Coloque os parâmetros build e ports de acordo com a print acima.

23. **Após definir isso, vamos executar o segundo serviço, de maneira indentada, digite:** redis **e depois configure sua imagem para:** “redis:alpine”

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
  redis:
    image: "redis:alpine"
```

Após finalizar todos os passos acima, o seu docker-compose.yaml deve ficar da seguinte maneira: (print acima)

Etapa 2 - Executar o Docker compose

1. **Para executar o Docker compose, digite:** docker-compose up -d (COLOCAR PRINT ABAIXO)

OBS: O parâmetro -d é para executar o Docker Compose em daemon, ou seja, em segundo plano.