DOCKER E DOCKER-COMPOSSE PARTE 2

Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

smarppy.com

douglas@smarppy.com

Slides: https://git.io/fpgYO

AGENDA

- Containers
- Seu novo ambiente de desenvolvimento
- Definindo um container com **`dockerfile`**
- A aplicação
- Construindo a imagem
- Iniciando o container
- Compartilhe sua imagem
- Referências

CONTAINERS

CONTAINERS

- Para construir uma aplicação com Docker, é preciso começar da raiz de toda a hierarquia da aplicação: o container.
- Acima do container, temos o serviço (*service*), que define como os containers irão se comportar em produção.
- Finalmente, no topo da lista, temos a pilha (stack), que define as interações entre todos os serviços.

Stack

Service

Container (estamos aqui)

SEU NOVO AMBIENTE DE DESENVOLVIMENTO

SEU NOVO AMBIENTE DE DESENV.

- Normalmente, quando você inicia um novo projeto, primeiro você precisa preparar todas as ferramentas de *runtime* (PHP, Java, Python, Node) em sua máquina.

- Porém, o ambiente de sua máquina precisa ser perfeito para sua aplicação rodar como o esperado. Além de ser idêntico ao ambiente de desenvolvimento.

SEU NOVO AMBIENTE DE DESENV.

- Com **Docker**, você precisa apenar obter uma imagem contendo o *runtime* desejado, nenhuma instalação é necessária.

- Então, sua construção pode conter uma imagem do *runtime* desejado com todas as bibliotecas e dependências necessárias, juntas e organizadas.

DEFININDO UM CONTAINER COM DOCKERFILE

DEF. UM CONTAINER COM DOCKERFILE

- O **Dockerfile** define o que acontece no ambiente dentro do container.
- Acesso a recursos como interface de rede e drivers de disco, são virtualizados dentro do ambiente, de forma isolada do restante do sistema.
- Sendo assim, é preciso mapear portas e diretórios para fora do container, além de especificar quais arquivos devem ser copiados para dentro do container.
- Deste modo, temos a garantia que nossa aplicação terá o mesmo comportamento, onde quer que seja executada.

DEF. UM CONTAINER COM DOCKERFILE

- Crie um novo diretório em um local apropriado.
- Dentro deste diretório, crie um arquivo com o nome "Dockerfile" (sem extensão).
- Copie e cole o código a seguir para dentro do seu Dockerfile:

https://gist.github.com/douglasjunior/7a6ff07235e13883b2819d828ea8d1ed#file-dockerfile

A APLICAÇÃO

A APLICAÇÃO

- Crie mais dois novos arquivos, dentro do mesmo diretório do **Dockerfile**:
  - requirements.txt
  - app.py

- Isso conclui nosso aplicativo, que, como você pode ver, é bastante simples.

- Quando nosso Dockerfile for transformado em uma imagem, os arquivos `app.py` e `requeriments.txt` estarão presentes, graças ao comando `COPY`.

- E ainda, o conteúdo HTML gerado pelo `app.py` estará disponível na porta 80, graças ao comando `EXPOSE`.

CONSTRUINDO A IMAGEM

CONSTRUINDO A IMAGEM

- Estamos prontos para criar a imagem de nossa aplicação. Perceba que o parâmetro `-t` é usado para dar um nome amigável para a imagem.

```
$ docker build -t meuapp .
```

- Para listar a imagem criada:

```
$ docker image ls (ou docker images)
```

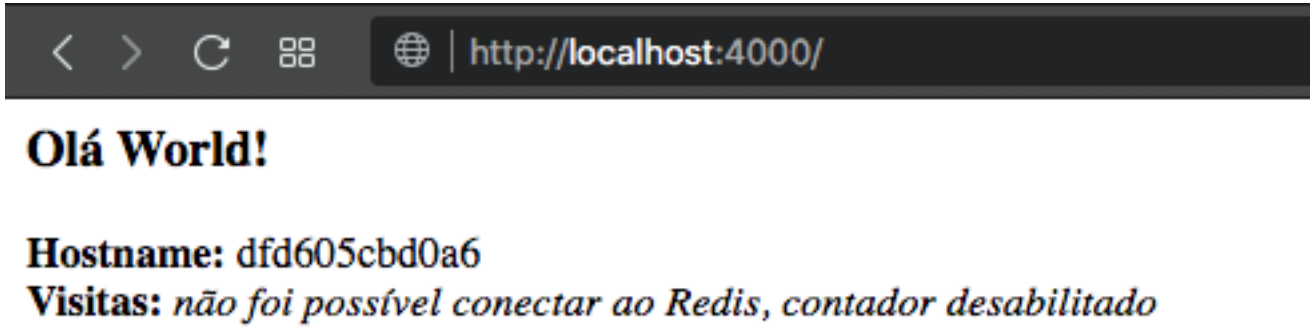
INICIANDO O CONTAINER

INICIANDO O CONTAINER

- Execute sua aplicação, mapeando a porta 80 do container para a porta 4000 da sua máquina (apenas para evitar possível conflitos).

```
$ docker run -p 4000:80 meuapp
```

- Ao acessar o endereço em seu navegador, você deve visualizar algo assim:



INICIANDO O CONTAINER

- Você também pode rodar a aplicação em segundo plano, sem que o terminal fique preso:
- `$ docker run -p 4000:80 meuapp`

- O comando anterior lhe retornará o ID do container em execução. Agora seu container está executando em *background*.

- Você pode consultar os containers em execução com o comando:

`$ docker container ls (ou docker ps)`

- Para interromper o container, execute o commando:
- `$ docker container stop <container-id>`

COMPARTILHE SUA IMAGEM

COMPARTILHE SUA IMAGEM

- Para demonstrar a portabilidade do que acabamos de criar, vamos subir nossa imagem e rodar em algum lugar.

- Para isso precisamos entender o conceito de registro (registry) e repositório (repositories).

- **Repository**: São coleções de imagens.
- **Registry**: São coleções de repositórios.

- Assim como um Gitlab repositor, porém o código já estará "compilado". Uma conta vinculada a um registry pode criar diversos repositórios.

- O `docker` CLI utiliza o registry público por padrão.

COMPARTILHE SUA IMAGEM

- Se você ainda não tem uma conta no Docker, crie uma em hub.docker.com.

- Faça login no registry público em sua máquina local:

```
$ docker login
```

COMPARTILHE SUA IMAGEM

- Adicione uma `tag` à sua imagem!

- A `tag` é opcinal, mas recomendada. Pois os registry utilizam a tag para controlar as versões de suas imagens.

- Recomenda-se utilizar nomes sugestivos para facilitar o controle de versões de sua imagem:

```
$ docker tag meuapp douglasjunior/meuapp-python:v1
```

COMPARTILHE SUA IMAGEM

- **Envie sua imagem marcada com a tag para o repositório:**
- `$ docker push douglasjunior/meuapp-python:v1`

- **Baixe e execute sua imagem, em outro computador, à partir do repositório remoto:**
- `docker run -p 4000:80 douglasjunior/meuapp-python:v1`

REFERÊNCIAS

- Docker Hub - https://hub.docker.com/

DÚVIDAS?

Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

smarppy.com

douglas@smarppy.com

Slides: https://git.io/fpgYO