Docker Tutorial

Comandos Dockerfile Docker Compose

Instalação

```
sudo apt update

sudo apt remove docker docker-engine docker.io

sudo apt install docker.io

sudo systemctl start docker

sudo systemctl enable docker
```

Para não precisar usar o **sudo** toda vez

Principais Comandos do Docker

- `docker version` : para ver a versão do docker instalada
- `docker run -it -d --rm --name containerName --link containerName -e environmentVariable=value -p 8080:8081 -v /home/dck:/var/www imageName comando` : cria e executa uma instância da imagem em um conteiner que será removido **--rm** após a conclusão da execução, executado em segundo plano **-d** *(detatched)*, com um nome dado **--name**. Um comando pode ser executado diretamente na frente do nome da imagem. A porta 8080 da máquina será exposta como 8081 no conteiner com a *flag* **-p**. O volume ou diretório */home/dck* da máquina será utilizado para escrita do diretório */var/www* do conteiner usando a *flag* **-v**. A *flag* **-it** indica a utilização do terminal no modo interativo. A *flag* **--link containerName** expõe o *containerName* para uso em lugar de usar o endereço IP do conteiner. Assim, podemos trocar 172.168.0.4 por *containerName* nas aplicações. A *flag* **-e** é usada para configurar variáveis de ambiente.
- `run vs exec` : **run** inicia um novo conteiner e executa um comando; **exec** executa um comando em um conteiner já ativo.
- `docker run -d --name myNginx -p 8080:80 nginx`
    - `docker exec -it myNginx bash`
    - `apt-get update`
    - `apt-get install -y vim`
    - `cd /usr/share/nginx/html`
    - `vim index.html`
- `docker image ls -q` : lista as imagens instaladas; a *flag* **-q** faz listagem apenas das *hash IDs*.

- `docker container ls -a -q` : lista os conteineres ativos. A *flag* **-a** faz listagem de todos os conteineres, inclusive os inativos; e a *flag* **-q** mostra apenas as *hash IDs*.
- `docker ps -a` : lista os conteineres ativos. A *flag* **-a** faz listagem de todos os conteineres, inclusive os inativos.
- `docker inspect containerName` : traz todas as informações do conteiner, inclusive seu endereço IP.
- `docker rm containerName/ID` : remove o conteiner. O conteiner precisa estar inativo para que funcione. Pode-se usar o início da *hash ID* para localizar o conteiner correto.
- `docker container prune` : remove todos os conteineres inativos de uma vez.
    - uma alternativa é `docker rm $(docker ps -a -q)`.
- `docker rmi imageName/ID` : remove a imagem. Não se pode ter nenhum conteiner ativo usando essa imagem. Pode-se usar o início da *hash ID* para localizar a imagem correta.
- `docker rmi $(docker images -q) -f` : Força **-f** a exclusão de todas as imagens **$(docker images -q)**.
- `docker stop containerName` : finaliza a execução de um container.
- `docker start containerName` : inicia a execução de um container.
- `docker build -t imageName -f pathOfTheDockerFileName .` : compila uma nova imagem baseada no arquivo *Dockerfile* ou *fileName.dockerfile*. A *flag* **-t** dá uma *tag* ou *name* para a imagem e a *flag* **-f** indica o arquivo fonte.

DOCKERFILE

Para compilar a imagem, use o `docker build` já explicado anteriormente.

- `FROM imagem:versão`
- `RUN comando -- exemplo apt-get install algumaCoisa` : são comandos do próprio programa/imagem.
- `EXPOSE 8000` : expõe a porta 8000 para exportar dados. Sempre que acessarem a porta 8000, este conteiner responderá. Não esqueça de executar um `docker run -p porta:8000`.
- `COPY . .` : copia todos os arquivos da pasta atual para dentro da imagem, inclusive o Dockerfile.
- `ENTRYPOINT ["./main", "-f"]` : quando tudo estiver pronto e o programa principal for executar, executará o comando "./main -f".

Exemplo

Usando a linguagem Go (golang) e um arquivo dockerfile.

main.go

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request){
    var texto = "<html><style>img{width:99vw;}</style>\n<img
src=\"https://1.bp.blogspot.com/-
egLGJOFCq7Y/Trkm7dmIIAI/AAAAAAAAAGg/nulr2ixOi88/s1600/final_getsuga_tenshou
_by_24352345.jpg\" alt=\"\"></img>\n</html>"
    fmt.Fprintf(w, texto)
}

func main(){
    http.HandleFunc("/", handler)
    log.Fatal(http.ListenAndServe(":8081", nil))
}
```

Dockerfile

```
FROM golang:1.14

COPY . .

RUN go build main.go

EXPOSE 8081

ENTRYPOINT ["./main"]
```

Execute

```
docker build -t minhaprimeiraimagem .
```

na mesma pasta em que se encontram **main.go** e **Dockerfile**. Os nomes das imagens precisam ser lowercase.

Depois, para executar a imagem,

```
docker run --rm -d -p 8081:8081 --rm minhaprimeiraimagem
```

Agora acesse um navegador com **localhost:8081**.

Docker Compose

Para instalar `sudo apt install docker-compose`.

> docker-compose.yaml

```
version: '3'

services:
    nginx:
        image: nginx
        volumes:
            - ./nginx:/usr/share/nginx/html
        ports:
            - 8080:80
```

Na pasta corrente, crie uma pasta **./nginx**. Dentro dela, escreva qualquer documento **index.html**.

Para startar a composição acima, apenas dê um

> `docker-compose up -d` *(detatched)*

Para terminar

> `docker-compose down`

Alguns navegadores podem reclamar dessa abordagem, portanto é importante testar com mais de um.

Acesse com algum navegador o endereço **localhost:8080**.

> outros comandos

```
build: .  === compila uma imagem Dockerfile constante na pasta atual
```