

Sumário

1. Oque é Docker e para que serve ?	2
2. Instalando o Docker.....	2
3. Imagens	3
4. Baixando Imagens.....	3
5. Executando Containers	4
6. Listando containers em execução	4
7. Removendo containers.....	5
8. Iniciando nos comandos.	5
9. Trabalhando com as imagens	5
10. Usando volumes	9
11. Construindo nossas próprias imagens.....	11
12. Comunicação entre containers.....	12
13. Trabalhando com Docker compose	12

A large, light blue, semi-transparent watermark of the Docker logo is centered on the page. It features the same whale and containers icon as the header logo, but in a much larger size, and the word "docker" in a large, lowercase, sans-serif font below it.

1. O que é Docker e para que serve ?

O Docker é um sistema de virtualização não convencional. Mas o que isso quer dizer? Em virtualizações convencionais temos um software instalado na máquina Host que irá gerenciar as máquinas virtuais (ex.: VirtualBox, VMWare, Parallels e etc...).

Para cada máquina virtual temos uma instalação completa do S.O. que queremos virtualizar, além de ter o próprio hardware virtualizado.

Se por exemplo eu precisar de uma biblioteca comum para todas as máquinas virtuais, preciso instalar em cada uma delas.

O Docker usa uma abordagem diferente, ele utiliza o conceito de container. Como assim container?

Basicamente uma ferramenta que pode gerenciar vários ambientes ao mesmo tempo.

2. Instalando o Docker

Atualmente Docker está disponível em duas versões Docker Community Edition(CE) e Docker Enterprise Edition(EE).

Em ambas as versões temos acesso a toda a API, basicamente a diferença entre as duas versões é o perfil desejado de aplicações. No EE temos um ambiente homologado pela Docker com toda infraestrutura certificada, segura pensada para o mundo enterprise. Já na versão CE podemos chegar ao mesmo nível que EE porém de uma forma manual.

Nesse link você pode encontrar as distribuições para downloads em cada sistema operacional disponível e os passos para instalação.

Feito a instalação, execute esse comando no terminal **docker --version**. Se a instalação ocorreu com sucesso deve ser impresso algo semelhante a isso Docker **version 17.03.1-ce, build c6d412e**.

3. Imagens

O Docker trabalha com o conceito de imagens, ou seja, para colocar um container em funcionamento o Docker precisa ter a imagem no host.

Essas imagens podem ser baixadas de um repositório (a nomenclatura para esse repositório é registry) ou criadas localmente e compiladas.

4. Baixando Imagens

Para baixar uma imagem podemos usar o comando **docker pull** e o nome da imagem que queremos baixar. Vamos baixar a imagem do Ubuntu, para isso execute o seguinte comando no terminal: **docker pull ubuntu**.

```
➔ ~ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
b6f892c0043b: Pull complete
55010f332b04: Pull complete
2955fb827c94: Pull complete
3deef3fcbd30: Pull complete
cf9722e506aa: Pull complete
Digest: sha256:382452f82a8bbd34443b2c727650af46aced0f94a44463c62a9848133ecb1aa8
Status: Downloaded newer image for ubuntu:latest
➔ ~
```

Para listar todas as imagens podemos usar o comando Docker **images**. O retorno desse comando é algo semelhante a isso:

```
➔ ~ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	ebcd9d4fca80	46 hours ago	118 MB

5. Executando Containers

A partir da imagem podemos iniciar quantos containers quisermos Através do comando `docker run` .

Para acessarmos um terminal do Ubuntu podemos usar o comando `docjer run -i -t ubuntu` ou `Docker run -it ubuntu` . O parâmetro `-i` indica que queremos um container interativo, o `-t` indica que queremos anexar o terminal virtual `tty` do container ao nosso host.

6. Listando containers em execução

Para ver os containers em execução podemos usar o comando `Docker ps` (em outro terminal ou aba), e ele exibirá um retorno parecido com esse:

A terminal window showing the output of the 'docker ps' command. The output is a table with columns: CONTAINER ID, IMAGE, COMMAND, CREATED, STATUS, PORTS, and NAMES. One container is listed with ID 'a6697ed945d5', image 'ubuntu', command '/bin/bash', created '12 hours ago', status 'Up 4 seconds', no ports, and name 'dreamy_bassi'.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a6697ed945d5	ubuntu	"/bin/bash"	12 hours ago	Up 4 seconds		dreamy_bassi

Aqui temos informações sobre os containers em execução, como id, imagem base, comando inicial, há quanto tempo foi criado, status, quais portas estão disponíveis e\ou mapeadas para acesso e o nome do mesmo. Quando não especificamos um nome ao iniciá-lo, será gerado um nome aleatoriamente.

Quando encerramos um container ele não será mais exibindo na saída do comando `Docker ps`, porém isso não significa que o container não existe mais. Para verificar os containers existentes que foram encerrados podemos usar o comando `Docker ps -a` e teremos uma saída parecida com essa:

A terminal window showing the output of the 'docker ps -a' command. The output is a table with columns: CONTAINER ID, IMAGE, COMMAND, CREATED, STATUS, PORTS, and NAMES. One container is listed with ID 'a6697ed945d5', image 'ubuntu', command '/bin/bash', created '12 hours ago', status 'Exited (0) 4 seconds ago', no ports, and name 'dreamy_bassi'.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a6697ed945d5	ubuntu	"/bin/bash"	12 hours ago	Exited (0) 4 seconds ago		dreamy_bassi

Como o próprio `status` do container informa, o mesmo já saiu de execução e no nosso caso saiu com status `0` (ou seja saiu normalmente).

7. Removendo containers

Para remover o container podemos usar o comando `Docker rm` e informar o id do container ou o nome dele. Para nosso caso poderíamos executar o comando `Docker rm 43aac92b4c99` ou `Docker rm dreamy_bassi` para remover o container por exemplo.

Caso tenhamos a necessidade de remover todos os containers (em execução ou encerrados) podemos usar o comando `Docker rm $(Docker ps -qa)`. A opção `-q` do comando `Docker ps` tem como saída somente os ids dos containers, essa lista de ids é passado para o `docker rm` e como isso será removido todos os containers.

Só será possível remover um container caso o mesmo não esteja em execução, do contrário temos que encerrar o container para removê-lo.

8. Iniciando nos comandos.

Docker version – exibe a versão instalada no seu S.O.

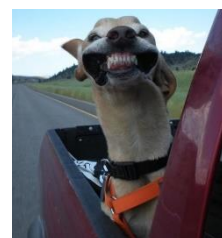
Docker run NOME_DA_IMAGEM – cria um container com a respectiva imagem passada como parâmetro.

9. Trabalhando com as imagens

Podemos utilizar tanto o CMD quanto o Power Shell, porem o recomendado pelo Docker é o Power Shell.

Então larga de charme e abre o Power Shell logo.

Comece utilizando o comando `Docker run hello-world`, o retorno será:



```
PS C:\WINDOWS\system32> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Onde basicamente o Docker tenta localizar a imagem localmente e quando essa tentativa falha ele navega até a Docker store, acha a imagem pedida e realiza o download e criação do novo container com a nova imagem.

Comandos para listagem de containers

Docker ps : lista todos os containers ativos no momento.

Docker ps -a : lista todos os containers independente de seu estado.

O retorno será:

Docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e5031503ef48	hello-world	"/hello"	5 minutes ago	Exited (0) 5 minutes ago		intelligent_mahavira
c5adda9bb5c0	hello-world	"/hello"	20 minutes ago	Exited (0) 20 minutes ago		focused_merkle
20a9cf120e0e	ubuntu	"bash"	33 minutes ago	Exited (0) 33 minutes ago		friendly_archimedes

Agora vamos acessar o terminal de um container, para isso utilize o comando **Docker start -a -i ID_DO_CONTAINER**

O retorno será:

```
PS C:\WINDOWS\system32> docker start -a -i 78cd8926f0ca
root@78cd8926f0ca:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@78cd8926f0ca:/#
```

Após isso nosso container já estará em execução e você pode começar a utilizar os comandos referentes a imagem do mesmo.

Para parar um container, o comando utilizado é **Docker stop**.

Se reparar temos agora vários containers inativos e precisamos remover os mesmo.

Para remover um container o comando utilizado é **Docker rm ID_CONTAINER**.

Porem esse comando removerá somente 1 container e eu quero remover todos inativos de uma só vez, para isso utilizamos o comando **Docker container prune**.

O retorno será:

```
PS C:\WINDOWS\system32> docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N]
```

Uma mensagem perguntando se você deseja remover todos os containers inativos de uma só vez, digite y para sim.

O retorno será parecido com:

```
Deleted Containers:
78cd8926f0cacd894ab5ebc3445380cb7f409601520bf571d86387cd006c9bc9
e5031503ef48235c23cdd97ca8b3b4fc524a8f478dbee1e0e62e8bd29d253d0a
c5adda9bb5c03c9d4c336d3f212a0cd3bfb5f9e22c182d26bbb287ae2fc91d13
20a9cf120e0e266375d28ba7b927875b6f7f8a57b2b058eb101cb58b2fb28625

Total reclaimed space: 14B
PS C:\WINDOWS\system32>
```

Como listar as imagens baixadas do no S.O.

Para isso utilizamos o comando **Docker images**.

O retorno será algo parecido com:

```
PS C:\WINDOWS\system32> docker images
REPOSITORY      TAG         IMAGE ID      CREATED        SIZE
ubuntu          latest      fb52e22af1b0  7 days ago    72.8MB
hello-world     latest     d1165f221234  6 months ago  13.3kB
```

Será apresentado uma listagem de todas as imagens baixadas no S.O.

Para fazer a remoção de uma imagem, o comando utilizado é **Docker rmi NOME_OU_ID_IMAGEM**.

Vinculação de portas no Docker

Para entender melhor como funciona essa questão é necessário que realizemos alguns comandos.

O primeiro vai ser o **Docker -d -P dockersamples/static-site**, onde o parâmetro **-d** servirá para rodar a aplicação em segundo plano, sem travar seu terminal e o **-P** para vincular as portas desse site estático com seu host local.

O retorno será:

```
PS C:\WINDOWS\system32> docker run -d -P dockersamples/static-site
4fd0e7918e0c445a04b6564e31abe60d21e4bc608a766975efefc8c920f202e4
```

Após realizar esse comando, execute Docker ps.

```
PS C:\WINDOWS\system32> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED NAMES	STATUS	PORTS
4fd0e7918e0c	dockersamples/static-site	"/bin/sh -c 'cd /usr..."	20 seconds ago happy_chandrasekhar	Up 17 seconds	0.0.0.0:49154->80/tcp, ::49154->80/tcp, 0.0.0.0:49153->443/tcp, ::49153->443/tcp

Porem a visualização ainda não está muito boa.

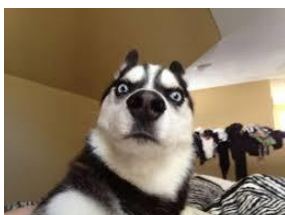
Vamos melhorar essa questão execute o comando **Docker port ID_CONTAINER**, esse comando vai listar somente as portas que estão sendo utilizadas pelo container.



O retorno será:

```
PS C:\WINDOWS\system32> docker port 4fd0e7918e0c
443/tcp -> 0.0.0.0:49153
443/tcp -> ::49153
80/tcp -> 0.0.0.0:49154
80/tcp -> ::49154
PS C:\WINDOWS\system32>
```

Como utilizamos o comando **-P** temos o retorno a cima, mostrando as portas do container e as portas de nosso host local que estão linkadas a elas.



Será que está funcionando??

Para testar vá até seu navegador e digite `localhost:49154`, é necessário que carregue essa página:



Hello Docker!

This is being served from a **docker** container running Nginx.

10. Usando volumes

Para que ser e o que é:

Os volumes servem para salvar dados diretamente no Docker host, dessa forma ao excluir um container, os dados presentes dentro do mesmo não serão perdidos juntamente com nosso container.



Vamos criar um volume dentro de um container:

Para isso utilizamos o comando `Docker run -v "/var/www" NOME_IMAGEM`

Ao executar esse comando o Docker criará um caminho aleatório para a pasta, porem também podemos definir o caminho do arquivo utilizando o comando `Docker -v "CAMINHO_DO_ARQUIVO:Exemplo/Exemplo" ubuntu`.

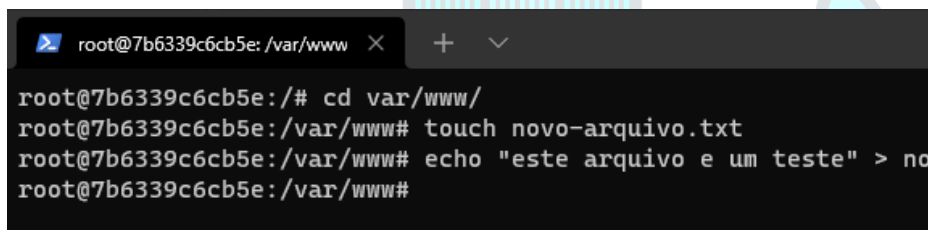
Vamos realizar um teste!

No terminal digite o comando `Docker -it -v "CAMINHO_DO_ARQUIVO:/Exemplo/Exemplo" ubuntu`

Após isso você será direcionado para o terminal do container, ao entrar no terminal navegue até o diretório `var/www/` > execute o comando `touch novo-arquivo.txt` > `echo "este arquivo e um teste" > novo-arquivo.txt` > `novo-arquivo.txt`

```
root@7b6339c6cb5e:/# cd var/www/
root@7b6339c6cb5e:/var/www# touch novo-arquivo.txt
root@7b6339c6cb5e:/var/www# echo "este arquivo e um teste" > novo-arquivo.txt
root@7b6339c6cb5e:/var/www#
```

Eu sei que me julgo agora 'há mais ele falou que não era para utilizar o cmd e ta usando'.



```
root@7b6339c6cb5e:/# cd var/www/
root@7b6339c6cb5e:/var/www# touch novo-arquivo.txt
root@7b6339c6cb5e:/var/www# echo "este arquivo e um teste" > novo-arquivo.txt
root@7b6339c6cb5e:/var/www#
```



Ta vendo o símbolo do power shell ali em cima, então quieto!!

Da mesma forma que podemos link um volume interno do Docker em nossa máquina podemos linkar um volume externo a um container.

Utilizamos basicamente o mesmo comando.

```
PS C:\Users\SUP-05> docker run -p 8080:3000 -v "C:\Users\SUP-05\Desktop\volume-exemplo:/var/www" -w "/var/www" node npm start
```

Para realizar o teste utilizei um arquivo `index.html`, `css`, `json`, `node.js`, após ter criado minha pasta com os códigos, no comando do Docker criei um volume linkado com a pasta desejada onde usei o parâmetro `-p` para vinculação das portas e `-w` para informar onde o Docker vai startar o contêiner e claro o contêiner foi criado em cima de uma versão do `node.js`.

11. Construindo nossas próprias imagens

O Docker também nos dá a possibilidade de criar novas imagens utilizando as informações necessárias.

Para isso precisamos criar um documento chamado Dockerfile, esse documento pode ser criado por exemplo no Visual Code.

Montar uma imagem pode ser necessário quando queremos compartilhar o ambiente de produção com outros desenvolvedores.

Comandos:

FROM: em que imagem nosso Dockerfile irá se basear um exemplo `FROM Node:latest`.

Maintainer: Nome de quem está criando a imagem (Nome da imagem).

Copy: mover por exemplo um código para dentro do container assim que o mesmo é executado.

RUN : Determinar comando que você quer que seja executado assim que inicie a imagem, assim que a imagem seja buildada (Dependências).

EXPOSE: Configuração e mapeamento de porta.

ENTRYPOINT: Comando usado para executar um comando assim que o contêiner for iniciado.

WORKDIR: Caminho onde será executado o comando.

Agora precisamos buildar essa imagem, para fazer isso execute o comando Docker `build -f dockerfile -t nomedaimagem`.

Após isso a imagem estará disponível dentro do Docker.

12. Comunicação entre containers

Por padrão todos os containers ficam conectados na mesma rede, isso acontece porque o próprio Docker já faz esse gerenciamento para os containers.

Com essas configurações é possível que você rode diferentes aplicações em diferentes containers, onde as mesma conseguem se comunicar via rede.

Instalando o Ping no seu container:

```
root@bd11591ebca5:/# apt-get update && apt-get install -y iputils-ping
```

O comando utilizado para isso será esse a cima.

Com a localhost do Docker não é possível realizar o ping pelo o hostname do container, para isso precisamos criar nossa própria rede de containers.

Para isso utilizamos o comando:

```
docker network create --driver bridge minha-rede
```

Criamos uma rede interna para o Docker, agora precisamos vincular essa rede aos containers que serão criados.

Para isso ao criar um container em Docker e vincular a rede criada é necessário adicionar um nome utilizando a tag **-name** Nome_do_container e vincular a rede utilizando o comando **-network** Nome_da_rede.

```
docker run -it -name meu-container-de-ubuntu --network minha-rede ubuntu
```

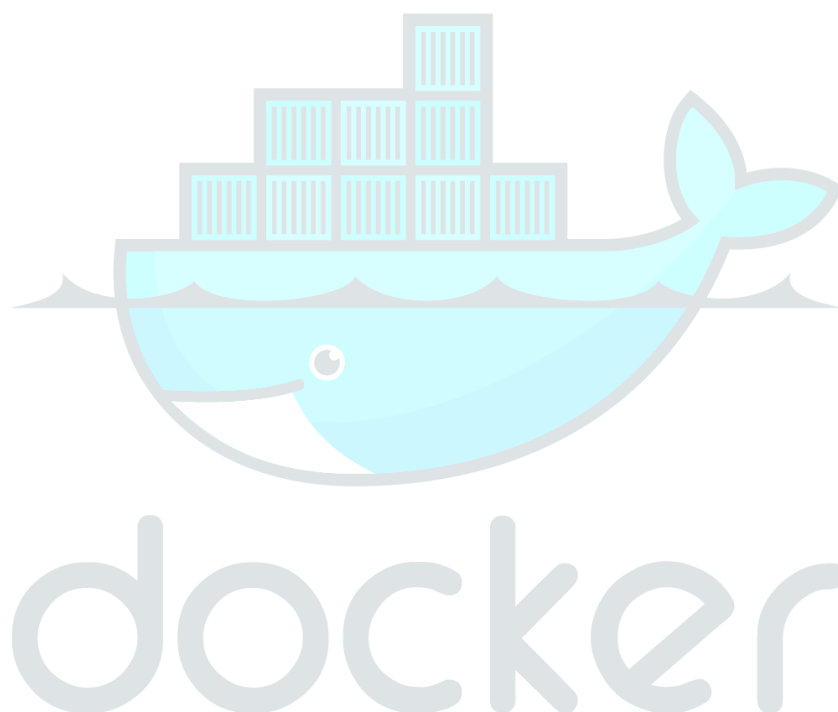
13. Trabalhando com Docker compose

Para que serve o Docker compose?

O Docker Compose serve para facilitar ao subir por exemplo vários containers de uma só vez, como se fosse uma receita de bolo, padrão para casos específicos.

Para utilizar essas facilidades do Docker compose e necessário criar uma arquivo **.yml** onde nesse mesmo arquivo vão as dependências e os parâmetros corretos como alocação de rede, containers à serem criados, redes a serem criadas, etc..

Conhecendo os comandos:



```
1  version: '3'
2  services:
3
4    nginx:
5      build:
6        dockerfile: ./docker/nginx.dockerfile
7        context: .
8      image: douglasq/nginx
9      container_name: nginx
10     ports:
11       - "80:80"
12     networks:
13       - production-network
14     depends_on:
15       - node1
16       - node2
17       - node3
18
19     mongodb:
20       image: mongo
21       networks:
22         - production-network
23
24     node1:
25       build:
26         dockerfile: ./docker/alura-books.dockerfile
27         context: .
28       image: douglasq/alura-books
29       container_name: alura-books1
30       ports:
31         - "3000"
32       networks:
33         - production-network
34       depends_on:
35         - mongodb
36
37     node2:
38       build:
39         dockerfile: ./docker/alura-books.dockerfile
40         context: .
41       image: douglasq/alura-books
42       container_name: alura-books2
43       ports:
44         - "3000"
45       networks:
46         - production-network
47       depends_on:
48         - mongodb
```

```
node3:
  build:
    dockerfile: ./docker/alura-books.dockerfile
    context: .
  image: douglasq/alura-books
  container_name: alura-books3
  ports:
    - "3000"
  networks:
    - production-network
  depends_on:
    - mongodb

networks:
  production-network:
    driver: bridge
```

Este seria um **docker-compose.yml** criado.

Temos os seguintes comandos

Version: Utilizado para especificar a versão do Docker-compose

Services: Utilizado para realizar a estrutura de nossos containers.

Para iniciar a estrutura, primeiro temos de colocar o nome de serviço.

Nesse mesma estrutura temos que colocar a **build**

Dockerfile: especifique o caminho da imagem criada (ou o nome).

Context: .

Image: especifique o nome da imagem

Container_name: especifique o nome do container

Ports: especifique a porta que será utilizada pelo cliente.

Networks: especifique a rede que será utilizada.

Depends_on: se esse arquivo ira depender de alguma outra aplicação para ser executada.

Networks: Responsável por criar uma rede interna para os containers.

Production-network:

Driver: driver da rede

Após fazer a criação do arquivo vamos para o Power Shell e execução, primeiramente utilizamos o comando **docker-compose build**, para que ele possa buildar as imagens, logo em seguida utilizamos o comando **docker-compose up -d**, para que possa subir todas as nossa instâncias.



Agora pega todo o conhecimento e tenta reproduzir.

A baleia tá de olho em!!!

