



Comandos básicos Docker



Thiago S. Adriano [Follow](#)

Oct 7, 2017 · 7 min read



Existem diversos comandos e parâmetros que podemos utilizar quando nós estamos trabalhando com contêineres. vejamos nesse artigo alguns dos comandos considerados principais quando estamos trabalhando com contêineres.

Docker info

docker info

Quando nós acabamos de subir o Docker Engine, nós utilizamos esse comando para verificarmos as informações do nosso Docker Host.

```
Backing Filesystem: extfs
Supports d_type: true
Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
```



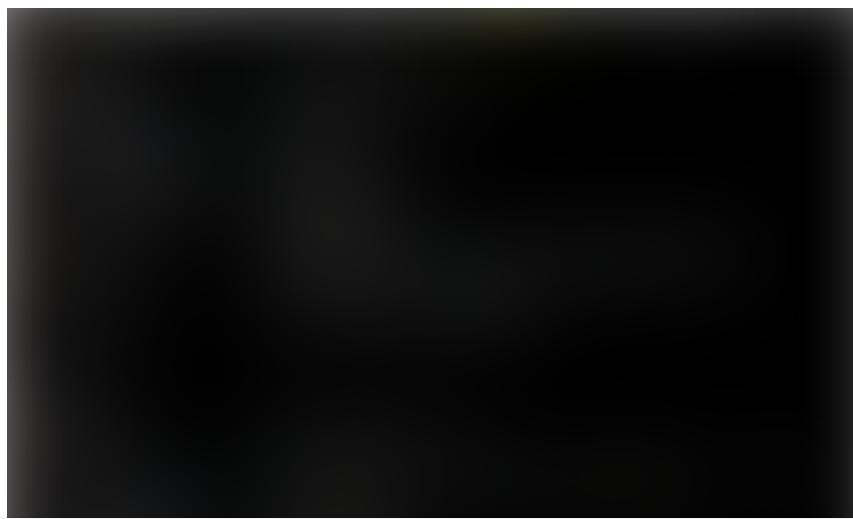
```
runc version: 3f2f8b84a77f73d38244dd690525642a72156c64
init version: 949e6fa
Security Options:
  seccomp
    Profile: default
Kernel Version: 4.9.49-moby
Operating System: Alpine Linux v3.5
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.934GiB
Name: moby
ID: LUED:VFYH:NTU6:ZRU6:XUEO:ZEUI:Q3ES:OYKO:3I56:SOQ6:45SI:TQAR
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
File Descriptors: 16
Goroutines: 25
System Time: 2017-10-07T12:41:35.0076454Z
EventsListeners: 0
Registry: https://index.docker.io/v1/
Experimental: true
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false
```

Docker info

Docker version

```
docker version
```

Com o `version` nós conseguimos ver a versão do nosso Client, para esse artigo estou utilizando o Windows e o Server que para esse exemplo estamos utilizar o Linux. Quanto ao OS/Arch do server nós podemos trabalhar com Windows ou Linux, caso você esteja utilizando o Docker for Windows basta clicar em Swith to Windows Containers que ele irá realizar essa alteração.





Docker version

Docker images

docker images

Nós utilizamos ele para listarmos as imagens que nós temos no nosso host, como eu acabei de instalar o Docker no meu Windows a lista está vazia como podemos ver na imagem a baixo:



- Repository: repositório;
- TAG: tag utilizada no repositório;
- IMAGE ID: o id na nossa imagem;
- Created: data de quando nós criamos a nossa imagem;
- Size: tamanho da imagem;

Docker search

docker search (parametro)

Para que possamos procurar uma imagem, nós podemos utilizar o comando a baixo com o parâmetro nome Ex.: ubuntu, dotnetcore, node ... etc, assim ele irá buscar as imagens que são compatíveis com o nosso server que para esse exemplo estamos utilizando o Linux.





Docker pull

```
docker pull (parametro)
```

Quando encontrarmos a imagem que precisamos para a nossa aplicação, nós precisamos baixar ela para o nosso host, para esse exemplo nós iremos utilizar uma imagem do Node.js

```
docker pull (parametro)
```



Docker pull node (baixando imagem do Node.js)

Como essa é a primeira vez que eu estou baixando o node ele irá demorar um pouco para finalizar esse comando, mas assim que ele baixar a nossa imagem execute o comando **docker images** para que possamos listar ela no nosso host.

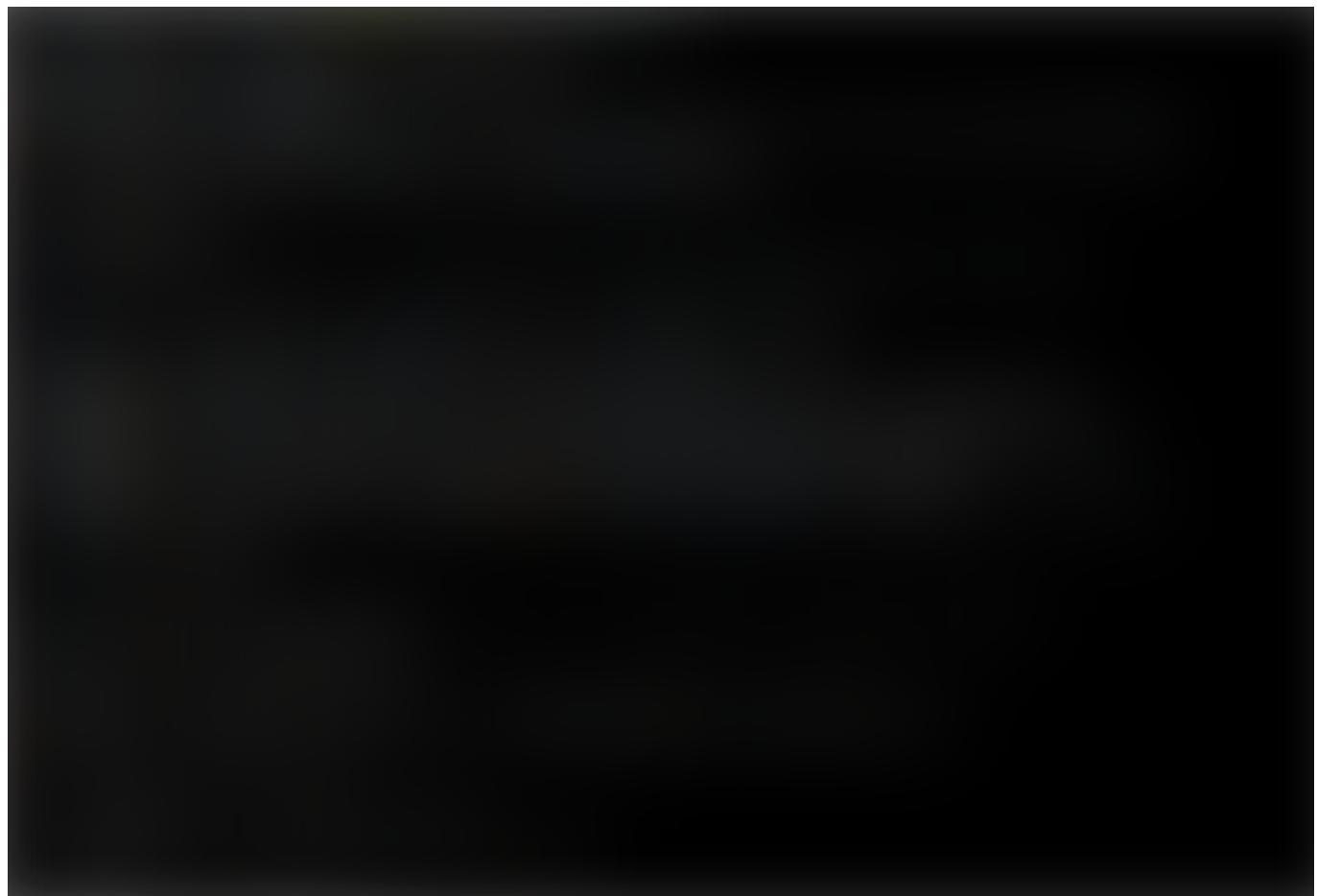


Docker images listando a imagem do nodejs



Para que nós possamos criar um contêiner, nós precisamos de uma imagem. Caso você não tenha essa imagem no seu host ainda ele irá até o repositório central e irá baixar ela para o seu host, em seguida ele irá criar o contêiner. Vamos criar um exemplo com o famoso Hello World. Para isso, execute o comando a baixo:

```
docker run hello-world
```



Hello World docker

Notem na imagem acima que o docker procurou a imagem no nosso host, como ele não encontrou ele baixou ela do docker hub. Agora execute o comando ***docker images*** novamente no seu terminal e note que temos uma nova imagem.



```
docker ps
```

Agora vamos verificar o status do nosso contêiner, execute o comando acima no seu terminal, ele irá retornar a imagem a baixo:



```
docker ps
```

Notem que ele está retornando uma lista vazia, isso acontece porque os contêineres nascem e morrem, quando nos executamos o comando docker run hello-world ele realizou todos os processos e morreu. Para que possamos ver ele podemos executar o comando docker ps com o parâmetro -a, assim teremos uma lista de todos os contêineres do nosso host.

```
docker ps -a
```



```
docker ps -a (listando todos os nossos containers)
```

Docker status

```
docker stats (id ou apelido do container)
```

Para que possamos ter informações sobre um contêiner nos executamos o comando acima, ele nos retorna dados como:

- **CONTAINER** — ID do Container;
- **CPU %** — uso de CPU em porcentagem;
- **MEM USAGE / LIMIT** — Memória usada/Limite que você pode ter setado;



docker stats (buscando dados sobre o nosso contêiner)

Docker inspect

docker inspect (id da imagem ou container)

Caso você precise de mais detalhes sobre a sua imagem ou o seu contêiner, podemos utilizar o comando inspect. Ele irá retornar um json com todas as informações relacionadas a nossa busca. No exemplo a baixo nós estamos executando ele na nossa imagem node.

docker inspect (imagem)

Docker rmi

docker rmi (nome da imagem)



```
docker rmi (deletando uma imagem)
```

Docker exec

```
docker exec (id_container ou nome_container)
```

Com o exec nós podemos executar qualquer comando nos nossos contêineres sem precisarmos estar na console deles.

Vejamos a baixo alguns dos parâmetros que podemos utilizar com ele:

- `-i` permite interagir com o container
- `-t` associa o seu terminal ao terminal do container
- `-it` é apenas uma forma reduzida de escrever `-i -t`
- `--name algum-nome` permite atribuir um nome ao container em execução
- `-p 8080:80` mapeia a porta 80 do container para a porta 8080 do host
- `-d` executa o container em background
- `-v /pasta/host:/pasta/container` cria um volume '/pasta/container' dentro do container com o conteúdo da pasta '/pasta/host' do host

Para que possamos testar ele, iremos baixar uma imagem do ubuntu. Para isso, execute o comando a baixo no seu terminal:

```
docker run -it -d ubuntu /bin/bash
```

Execute o comando `docker ps` agora e note que o nosso contêiner esta em up.



```
docker ps (ubuntu)
```

Até aqui nós temos nenhuma novidade, vamos agora criar um novo diretório dentro do nosso contêiner sem precisarmos entrar nele.

```
docker exec 8b54c76e81b7 mkdir /temp/
```

Agora vamos criar um arquivo dentro do nosso diretório temp que acabamos de criar dentro do nosso contêiner:

```
docker exec 8b54c76e81b7 touch /temp/dotnetsp.txt
```

Docker attach

```
docker attach (id_container ou nome_container)
```

Vamos agora entrar dentro do nosso contêiner e verificar o nosso novo arquivo criado. Para isso execute o comando acima + o nome do ou id do seu contêiner, podemos observar que a execução dele irá nos dar acesso de root no nosso contêiner.

Docker attach Contêiner Ubuntu

Agora execute o comando `ls /temp` para verificar o nosso arquivo dentro diretório que nós criamos chamado temp:



```
docker start (id do container)
```

Quando nós saímos do nosso container com o comando exit, ele mata o nosso processo. Notem na imagem a baixo que ele não está mais sendo listado no nosso comando **docker ps**:



Docker ps (depois que saímos do nosso contêiner com o comando exit)

Para que possamos dar um detach ou em outras palavras sair sem fecharmos ele, nós precisamos dos comandos: Ctrl+p + Ctrl+q. Para que possamos subir ele novamente vamos utilizar o comando start.



Docker start

Agora execute o comando attach novamente para entrar no seu contêiner e depois os comandos : Ctrl+p + Ctrl+q, em seguida podemos verificar que o nosso contêiner ainda está em execução:



Docker ps depois de dettach

Docker stop

```
docker stop (id ou nome container)
```



Docker stop

Mapeamento de Portas

-p host:container

Para que possamos testar o mapeamento de portas iremos baixar a imagem do *nginx* e passarmos para ele a porta 80. Para isso execute o comando a baixo no seu terminal:

```
docker run -it -p 8080:80 nginx
```

Ele irá baixar a imagem do *nginx* para o seu host e abrir ele na porta 8080 do seu host. Para testar assim que ele finalizar a execução do comando acima, abra no seu navegador o endereço: <http://localhost:8080/>, podemos ver o seu retorno na imagem a baixo.

container nginx

Mas executando ele dessa forma nós ficamos com o nosso console travado conforme podemos ver na imagem a baixo:



docker nginx

Para que possamos ter um contêiner sendo executado em background, nós podemos utilizar eles em segundo plano. Para isso, só precisamos passar o comando -d. Caso ainda esteja o seu contêiner ainda esteja sendo executado, execute o comando control c para sair dele, em seguida execute o mesmo comando anterior só que agora com o parâmetro -d:

```
docker run -it -d -p 8080:80 nginx
```

Por fim, execute o comando docker ps e veja ele na lista:



Com isso nós passamos por alguns dos comandos principais do docker.

Docker Commands



266 claps

↑ ↲ 000



WRITTEN BY

Thiago S. Adriano

Enjoy your life

Follow



Esse espaço sera destinado a artigos sobre Docker em português

[Follow](#)

[See responses \(1\)](#)

More From Medium

Related reads

If Statement in JSX



Penny Pang in path2code
Dec 18, 2018 · 1 min read



1.1K



Related reads

Model View Controller in Harbour



José Luis Sánchez in...
Jun 21, 2018 · 9 min read



119



Related reads

Building a developer portfolio with React, Cosmic JS, and

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy. ×

