

# 1 Executando & Inspecionando um Container

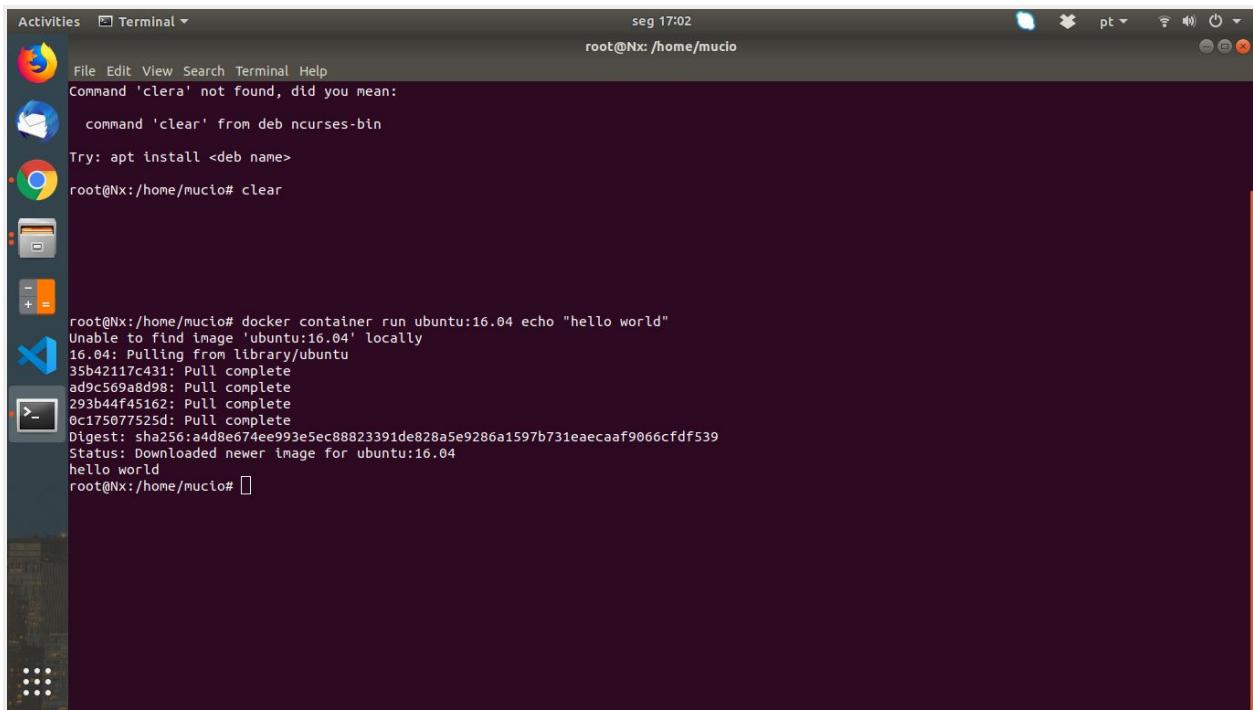
## 1.1 Executando Containers

1. Primeiro, inicie um container, e observe a saída:

```
$ docker container run ubuntu:16.04 echo "hello world"
```

O trecho `ubuntu:16.04` indica a imagem que queremos usar para definir esse contêiner; inclui o sistema operacional subjacente e todo o seu sistema de arquivos.

indicates the *image* we want to use to define this container; it includes the underlying operating system and its entire filesystem. `echo "hello world"` é o comando que queremos executar dentro do contexto do container definido por essa imagem.



```
Activities Terminal seg 17:02
root@Nx:/home/mucio
File Edit View Search Terminal Help
Command 'clera' not found, did you mean:
  command 'clear' from deb ncurses-bin
Try: apt install <deb name>
root@Nx:/home/mucio# clear
root@Nx:/home/mucio# docker container run ubuntu:16.04 echo "hello world"
Unable to find image 'ubuntu:16.04' locally
16.04: Pulling from library/ubuntu
35b42117c431: Pull complete
ad9c569a8d98: Pull complete
293b44f45162: Pull complete
0c175077525d: Pull complete
Digest: sha256:a4dbe674ee993e5ec88823391de828a5e9286a1597b731eaecaaf9066cfdf539
Status: Downloaded newer image for ubuntu:16.04
hello world
root@Nx:/home/mucio#
```

2. Agora crie outro container da mesma imagem e execute um comando diferente dentro dele:

```
$ docker container run ubuntu:16.04 ps -ef
```

```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container run ubuntu:16.04 ps -ef
UID      PID  PPID  C STIME TTY          TIME CMD
root         1      0  31 20:03 ?        00:00:00 ps -ef
root@Nx:/home/mucio#
```

3. ps -ef era PID 1 dentro do contêiner na última etapa; tente fazer o comando ps -ef no prompt do host e veja qual processo é o PID 1 aqui.

## 1.2 Listing Containers

1. Tente listar todos os seus contêineres em execução no momento:::

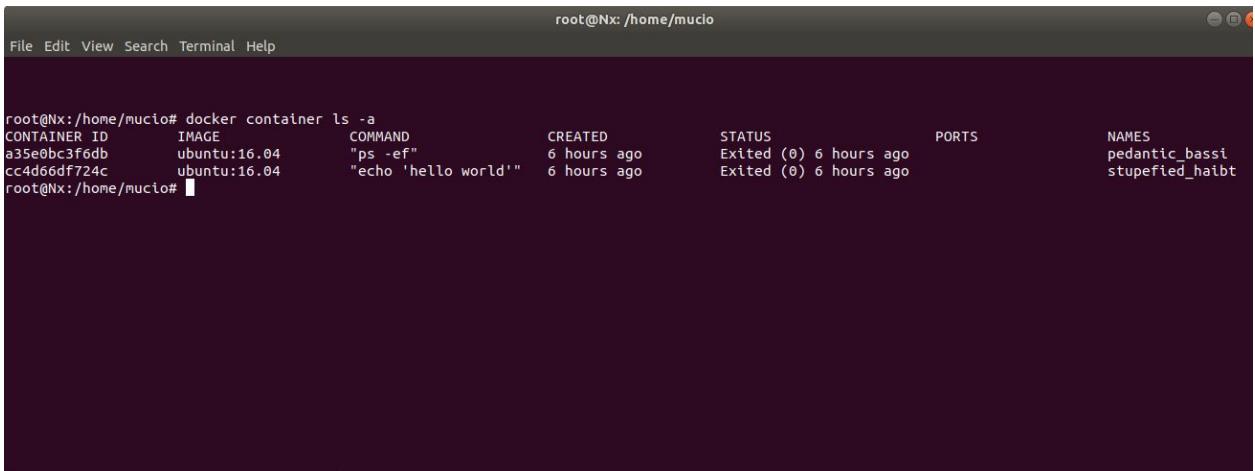
```
$ docker container ls
```

```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container ls
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS          NAMES
root@Nx:/home/mucio#
```

Não há nada listado, pois os contêineres executados executaram um único comando e foram encerrados quando terminaram.

- 2.Lista interrompida, assim como a execução de contêineres com a flag -a :

```
$ docker container ls -a
```



```
root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
a35e0bc3f6db        ubuntu:16.04       "ps -ef"           6 hours ago       Exited (0) 6 hours ago
cc4d66df724c        ubuntu:16.04       "echo 'hello world'"   6 hours ago       Exited (0) 6 hours ago
root@Nx:/home/mucio#
```

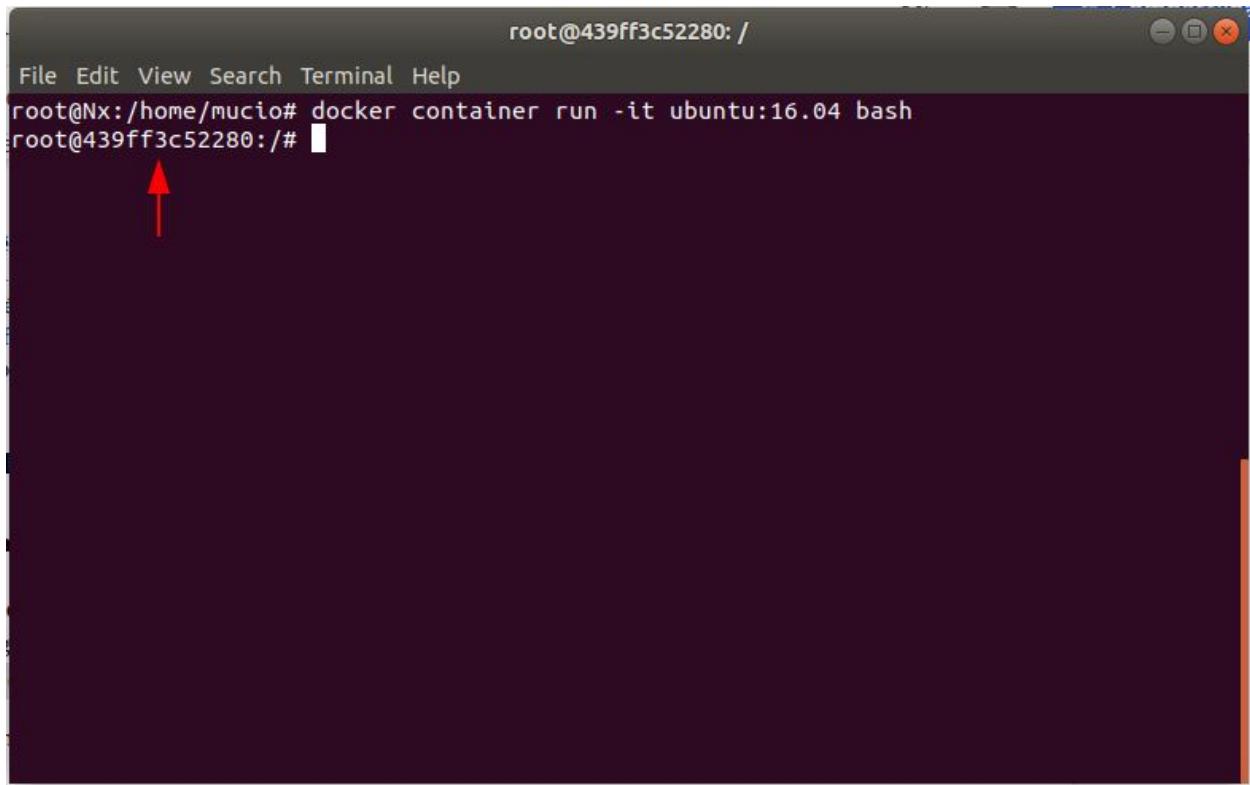
### 1.3 Conclusão

Neste exercício, você executou seu primeiro contêiner usando `docker container run`, e explorou a importância do processo do PID 1 em um contêiner. Este processo é um membro da árvore PID do host como qualquer outro, mas é 'containerizado' através de ferramentas como namespaces do kernel, fazendo com que este processo e seus filhos se comportem como se fossem a raiz de uma árvore PID, com seu próprio sistema de arquivos e pilha de rede. Além disso, esse processo define o estado do contêiner; Se o processo sair, o contêiner será interrompido.

## 2 Contêineres Interativos

### 2.1 Escrevendo para contêineres

1. Crie um container usando a imagem `ubuntu:16.04`, e conecte-se ao modo bash interativo usando a flag `-i` (também a flag `-t`, para requisitar uma conexão com o terminal TTY):  
\$ `docker container run -it ubuntu:16.04 bash`



```
root@439ff3c52280: /  
File Edit View Search Terminal Help  
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash  
root@439ff3c52280:/#
```

2. Explore o sistema de arquivos dos seus container's com `ls`, e então crie um novo arquivo, assim:

```
$ ls  
$ touch test.dat  
$ ls
```

A screenshot of a terminal window titled "root@439ff3c52280:/". The window shows a sequence of commands being run in a Docker container:

```
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash
root@439ff3c52280:/# ls
bin dev home lib64 mnt proc run srv tmp var
boot etc lib media opt root sbin sys usr
root@439ff3c52280:/# touch test.dat
root@439ff3c52280:/# ls
bin dev home lib64 mnt proc run srv test.dat tmp var
boot etc lib media opt root sbin sys usr
```

The terminal interface includes standard Linux navigation keys like Esc, F1-F12, Home, End, PgUp, PgDn, and arrow keys.

3. Saia da conexão com o container:

```
$ Exit
```

```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash
root@439ff3c52280:/# ls
bin dev home lib64 mnt proc run srv tmp var
boot etc lib media opt root sbin sys usr
root@439ff3c52280:/# touch test.dat
root@439ff3c52280:/# ls
bin dev home lib64 mnt proc run srv test.dat usr
boot etc lib media opt root sbin sys tmp var
root@439ff3c52280:/# exit ←
exit
root@Nx:/home/mucio#
```

4. Execute o mesmo comando acima para iniciar um contêiner no mesmo:

```
$ docker container run -it ubuntu:16.04 bash
```

```
root@9d01aa5b4291: /  
File Edit View Search Terminal Help  
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash  
root@439ff3c52280:/# ls  
bin dev home lib64 mnt proc run srv tmp var  
boot etc lib media opt root sbin sys usr  
root@439ff3c52280:/# touch test.dat  
root@439ff3c52280:/# ls  
bin dev home lib64 mnt proc run srv test.dat usr  
boot etc lib media opt root sbin sys tmp var  
root@439ff3c52280:/# exit  
exit  
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash  
root@9d01aa5b4291:/#
```

5. Tente encontrar seu arquivo `test.dat` dentro desse novo container; Ele não será encontrado; Saia deste container por enquanto, da mesma forma acima.

```
root@9d01aa5b4291: /  
File Edit View Search Terminal Help  
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash  
root@9d01aa5b4291:/# ls  
bin dev home lib64 mnt proc run srv tmp var  
boot etc lib media opt root sbin sys usr  
root@9d01aa5b4291:/#
```

Observação: Isso acontece pois você executou uma nova instância da imagem.

## 2.2 Reconectando Containers

1. Gostaríamos de recuperar as informações escritas em nosso container no primeiro exemplo, mas iniciar um novo container não nos levou até lá. Em vez disso, precisamos reiniciar o container original e nos reconectar a ele. Liste todos os contêineres parados:

```
$ docker container ls -a
```

```

root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
9d01aa5b4291        ubuntu:16.04       "bash"              3 minutes ago     Exited (0) 8 seconds ago
439ff3c52280        ubuntu:16.04       "bash"              10 minutes ago   Exited (0) 5 minutes ago
a35e0bc3f6db        ubuntu:16.04       "ps -ef"           6 hours ago      Exited (0) 6 hours ago
cc4d6df724c        ubuntu:16.04       "echo 'hello world'" 6 hours ago      Exited (0) 6 hours ago
root@Nx:/home/mucio# 

```

2. Podemos reiniciar um container, com o ID do Container listado na primeira coluna. Use o ContainerID para o primeiro container criado com bash ubuntu:16.04 com este comando:

```
$ docker container start <Container ID>
```

```

root@Nx:/home/mucio# docker container start 9d01aa5b4291
root@Nx:/home/mucio# docker container exec -it 9d01aa5b4291 bash
root@9d01aa5b4291:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@9d01aa5b4291:/# exit
exit
root@Nx:/home/mucio# docker container start 439ff3c52280
439ff3c52280
root@Nx:/home/mucio# 

```

3. Reconecte ao seu container com docker container exec (podemos usar isso para executar qualquer comando em um Container Docker em execução):

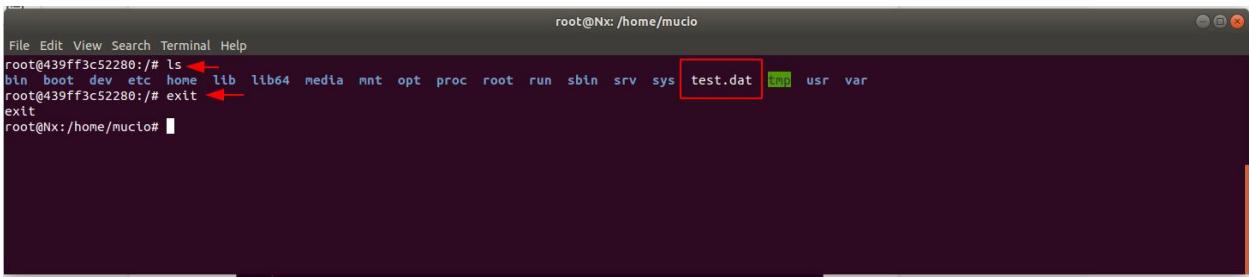
```
$ docker container exec -it <Container ID> bash
```

```

root@439ff3c52280:/#
File Edit View Search Terminal Help
439ff3c52280        ubuntu:16.04       "bash"              10 minutes ago   Exited (0) 5 minutes ago
a35e0bc3f6db        ubuntu:16.04       "ps -ef"           6 hours ago      Exited (0) 6 hours ago
cc4d6df724c        ubuntu:16.04       "echo 'hello world'" 6 hours ago      Exited (0) 6 hours ago
root@Nx:/home/mucio# docker container start 9d01aa5b4291
9d01aa5b4291
root@Nx:/home/mucio# docker container exec -it 9d01aa5b4291 bash
root@9d01aa5b4291:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@9d01aa5b4291:/# exit
exit
root@Nx:/home/mucio# docker container start 439ff3c52280
439ff3c52280
root@Nx:/home/mucio# docker container exec -it 439ff3c52280 bash
root@439ff3c52280:/# 

```

4. Liste o conteúdo do sistema de arquivo novamente com o comando ls, seu test.dat deve estar onde você deixou. Saia do container novamente usando o comando Exit.

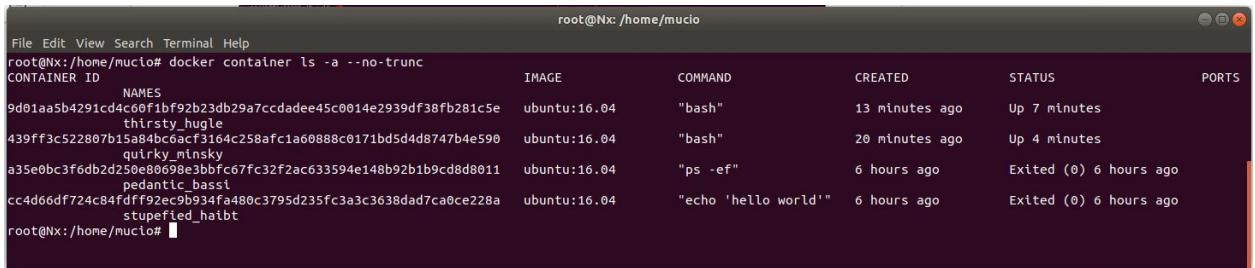


```
root@Nx:/home/mucio# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys test.dat tmp usr var
root@Nx:/home/mucio# exit
```

## 2.3 Mais opções de listagem de containers

1. Nesta última etapa, vimos como obter uma parte do ContainerID de todos nossos containers usando docker container ls -a. Caso queira obter todo o ContainerID tente adicionar a flag --no-trunc:

```
$ docker container ls -a --no-trunc
```

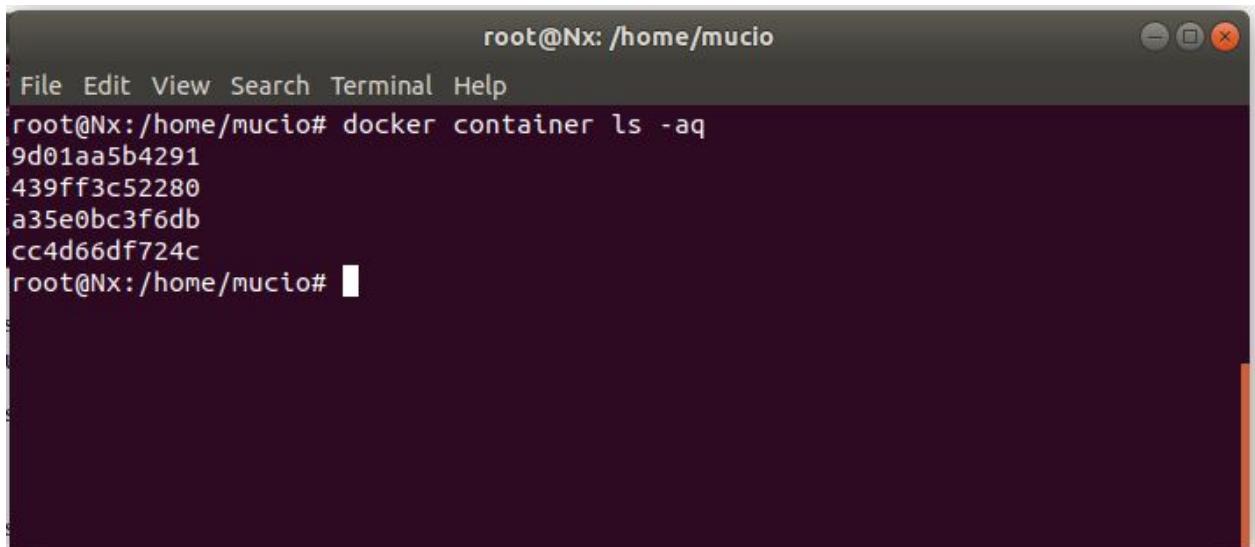


CONTAINER ID	NAMES	IMAGE	COMMAND	CREATED	STATUS	PORTS
9d01aa5b4291cd4c60f1bf92b23db29a7ccdaee45c0014e2939df38fb281c5e	thirsty_hugle	ubuntu:16.04	"bash"	13 minutes ago	Up 7 minutes	
439ff3c522807b15a84bc6acf3164c258afc1a60888c0171bd5d4d8747b4e590	quirky_minsky	ubuntu:16.04	"bash"	20 minutes ago	Up 4 minutes	
a35e0bc3f6db2d250e80698e3bbfc67fc32f2ac633594e148b92b1b9cd8d8011	pedantic_bassi	ubuntu:16.04	"ps -ef"	6 hours ago	Exited (0) 6 hours ago	
cc4d66df724c84fdf92ec9b934fa480c3795d235fc3a3c3638dad7ca0ce228a	stupefied_haibt	ubuntu:16.04	"echo 'hello world'"	6 hours ago	Exited (0) 6 hours ago	

Esse ID longo é o mesmo que a string retornada após o início de um container com docker container run.

2. Liste somente o containerID:

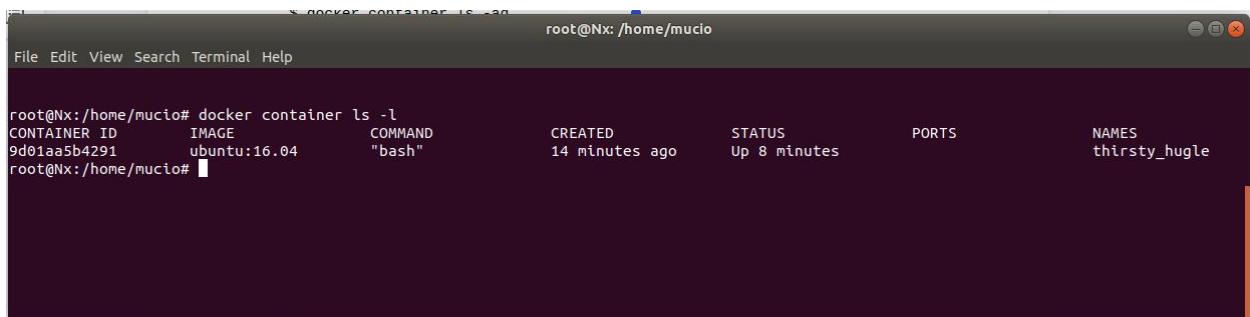
```
$ docker container ls -aq
```



```
root@Nx:/home/mucio# docker container ls -aq
9d01aa5b4291
439ff3c52280
a35e0bc3f6db
cc4d66df724c
root@Nx:/home/mucio#
```

3. Liste o último container iniciado:

```
$ docker container ls -l
```



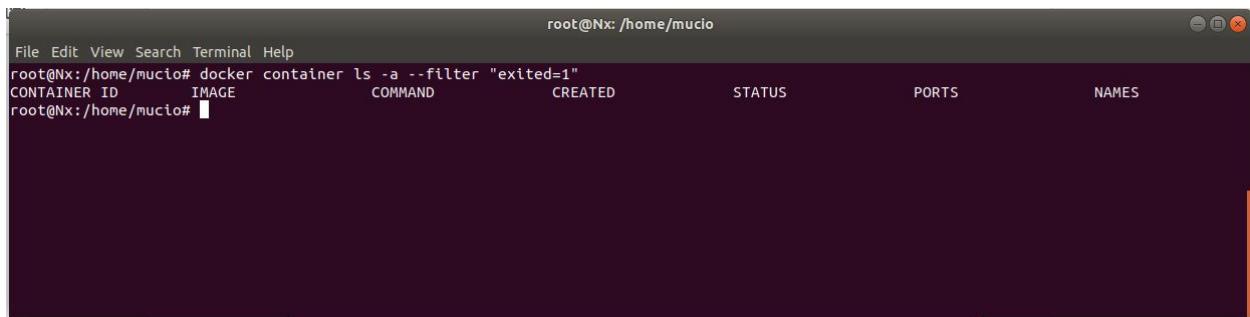
A terminal window titled "root@Nx:/home/mucio" showing the command "docker container ls -l". The output lists one container:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9d01aa5b4291	ubuntu:16.04	"bash"	14 minutes ago	Up 8 minutes		thirsty_hugle

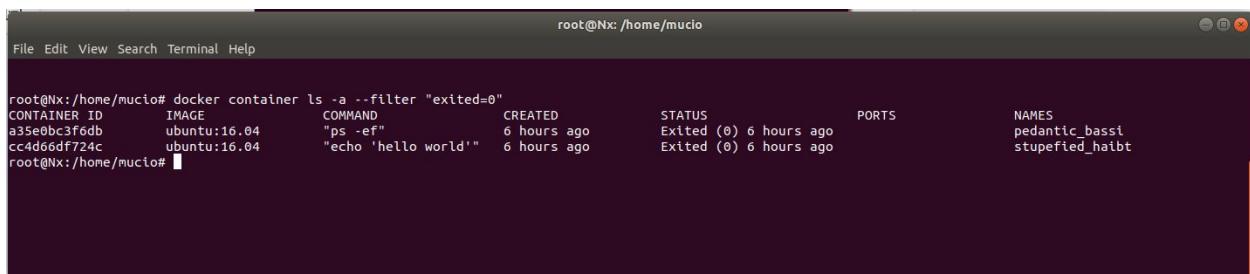
4. Finalmente, você pode também filtrar resultados com a flag `--filter`; Por exemplo, tente filtrando por código de saída [exit code]:

```
$ docker container ls -a --filter "exited=1"
```

```
$ docker container ls -a --filter "exited=0"
```



A terminal window titled "root@Nx:/home/mucio" showing the command "docker container ls -a --filter \"exited=1\"". The output shows no containers.



A terminal window titled "root@Nx:/home/mucio" showing the command "docker container ls -a --filter \"exited=0\"". The output lists two containers that have exited:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a35e0bc3f6db	ubuntu:16.04	"ps -ef"	6 hours ago	Exited (0) 6 hours ago		pedantic_bassi
cc4d66df724c	ubuntu:16.04	"echo 'hello world'"	6 hours ago	Exited (0) 6 hours ago		stupefied_hatbt

## 2.4 Conclusão

Nesta demonstração, você viu que os arquivos adicionados ao sistema de arquivos de um contêiner não foram adicionados a todos os contêineres criados a partir da mesma imagem. As alterações no sistema de arquivos de um contêiner são locais e existem apenas na camada R / W desse contêiner em particular. Você também aprendeu como reiniciar um container Docker parado usando `docker container start`, como executar um comando em um contêiner em execução usando `docker container exec`, e também aprendeu como reiniciar um contêiner Docker parado usando `docker container start`, como executar um comando em um contêiner em execução usando `docker container exec`, e também viu mais algumas opções para listar contêineres via `docker container ls`.

## 3 Contêineres destacados e registro em log

### 3.1 Execute um Container em Background

1. Primeiro tente executar um container normalmente, como já aprendeu até aqui; os fluxos STDOUT e STDERR de qualquer que seja o PID 1 dentro do contêiner são direcionados para o terminal:

```
$ docker container run ubuntu:14.04 ping 127.0.0.1 -c 10
```

The screenshot shows a terminal window titled "root@Nx:/home/mucio". The command entered was "docker container run ubuntu:14.04 ping 127.0.0.1 -c 10". The output shows the container pulling the image from the library and then performing a ping test. The ping statistics show 10 packets transmitted, 10 received, 0% packet loss, and an average round-trip time of 9210ms.

```
root@Nx:/home/mucio# docker container run ubuntu:14.04 ping 127.0.0.1 -c 10
Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
a7344f52cb74: Pull complete
515c9bb51536: Pull complete
e1eabe0537eb: Pull complete
4701f1215c13: Pull complete
Digest: sha256:2f7c79927b346e436cc14c92bd4e5bd778c3bd7037f35bc639ac1589a7acf90
Status: Downloaded newer image for ubuntu:14.04
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.090 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.045 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.070 ms

--- 127.0.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9210ms
rtt min/avg/max/mdev = 0.033/0.059/0.090/0.015 ms
root@Nx:/home/mucio#
```

2. O mesmo processo pode ser executado em segundo plano com a flag -d:

```
$ docker container run -d ubuntu:14.04 ping 127.0.0.1
```

The screenshot shows a terminal window titled "root@Nx:/home/mucio". The command entered was "docker container run -d ubuntu:14.04 ping 127.0.0.1". The output shows the container running in the background, with its ID displayed at the end of the line.

```
root@Nx:/home/mucio# docker container run -d ubuntu:14.04 ping 127.0.0.1
4846b84a8adfb7e5a37003b26ff4e976599ed0a88726c891cff932d007cbd81
root@Nx:/home/mucio#
```

3. Encontre o ID desse segundo contêiner e use-o para inspecionar os registros gerados:

```
$ docker container logs <container ID>
```

```
root@Nx:/home/mucio# docker container logs 4846b84a8adf
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.032 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.128 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.046 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.029 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.202 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.067 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=11 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=14 ttl=64 time=0.099 ms
64 bytes from 127.0.0.1: icmp_seq=15 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=16 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=17 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=18 ttl=64 time=0.076 ms
64 bytes from 127.0.0.1: icmp_seq=19 ttl=64 time=0.074 ms
64 bytes from 127.0.0.1: icmp_seq=20 ttl=64 time=0.049 ms
64 bytes from 127.0.0.1: icmp_seq=21 ttl=64 time=0.051 ms
64 bytes from 127.0.0.1: icmp_seq=22 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=23 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=24 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=25 ttl=64 time=0.041 ms
64 bytes from 127.0.0.1: icmp_seq=26 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=27 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=28 ttl=64 time=0.052 ms
64 bytes from 127.0.0.1: icmp_seq=29 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=30 ttl=64 time=0.031 ms
64 bytes from 127.0.0.1: icmp_seq=31 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=32 ttl=64 time=0.053 ms
```

Esses logs correspondem a STDOUT e STDERR do container PID1 do PID do container 1.

### 3.2 Anexando à saída do contêiner

1. Podemos anexar um terminal à saída do PID 1 de um contêiner com o comando attach; tente com o último container que você fez no passo anterior

```
$ docker container attach <container ID>
```

```
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container attach 4846b84a8adf
64 bytes from 127.0.0.1: icmp_seq=276 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=277 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=278 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=279 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=280 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=281 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=282 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=283 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=284 ttl=64 time=0.046 ms
64 bytes from 127.0.0.1: icmp_seq=285 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=286 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=287 ttl=64 time=0.039 ms
```

2. Podemos deixar o modo anexado pressionando CTRL+C. Depois de fazer isso, liste seus contêineres em execução; você deve ver que o contêiner ao qual você se conectou foi morto, já que o CTRL + C emitiu o PID 1 morto no contêiner e, portanto, o próprio contêiner.

```
File Edit View Search Terminal Help
root@Nx:/home/mucio#
64 bytes from 127.0.0.1: icmp_seq=346 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=347 ttl=64 time=0.049 ms
^C
--- 127.0.0.1 ping statistics ---
347 packets transmitted, 347 received, 0% packet loss, time 354293ms
rtt min/avg/max/mdev = 0.027/0.058/0.202/0.020 ms
root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
4846b84a8adf      ubuntu:14.04       "ping 127.0.0.1"   6 minutes ago    Exited (0) 21 seconds ago
e7d74dfd4ca3      ubuntu:14.04       "ping 127.0.0.1 -c 10" 7 minutes ago    Exited (0) 7 minutes ago
9d01aa5b4291      ubuntu:16.04       "bash"              25 minutes ago   Up 19 minutes
439ff3c52280      ubuntu:16.04       "bash"              32 minutes ago   Up 16 minutes
a35e0bc3f6db      ubuntu:16.04       "ps -ef"            7 hours ago     Exited (0) 7 hours ago
cc4d66df724c      ubuntu:16.04       "echo 'hello world'" 7 hours ago     Exited (0) 7 hours ago
root@Nx:/home/mucio#
```

3. Tente executar a mesma coisa no modo interativo desanexado:

```
$ docker container run -d -it ubuntu:14.04 ping 127.0.0.1
```

```
File Edit View Search Terminal Help
root@Nx:/home/mucio#
root@Nx:/home/mucio# docker container run -d -it ubuntu:14.04 ping 127.0.0.1
ac7a0beb6fc5c5687106d2f946c6f25c9eb395a28156f598c020316459a80351
root@Nx:/home/mucio#
```

4. Anexar a este contêiner como você fez o primeiro, mas desta vez desanexar com CTRL + P + Q e listar seus contêineres em execução. Nesse caso, o contêiner ainda deve estar feliz em execução em segundo plano depois de desanexar dele.

```

root@Nx:/home/mucio#
File Edit View Search Terminal Help
439ff3c52280      ubuntu:16.04    "bash"          32 minutes ago   Up 16 minutes
a35e0bc3f6db      ubuntu:16.04    "ps -ef"        7 hours ago     Exited (0) 7 hours ago
cc4d66df724c      ubuntu:16.04    "echo 'hello world'" 7 hours ago     Exited (0) 7 hours ago
root@Nx:/home/mucio# clear
root@Nx:/home/mucio# docker container run -d -it ubuntu:14.04 ping 127.0.0.1
ac7a0beb6fc5c5687106d2f946c6f25c9eb395a28156f598c020316459a80351
root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
ac7a0beb6fc5        ubuntu:14.04      "ping 127.0.0.1"   About a minute ago   Up 59 seconds
4846b84a8adf       ubuntu:14.04      "ping 127.0.0.1"   8 minutes ago     Exited (0) 2 minutes ago
e7d74df4ca3        ubuntu:14.04      "ping 127.0.0.1 -c 10" 9 minutes ago     Exited (0) 9 minutes ago
9d01aa5b4291       ubuntu:16.04      "bash"            27 minutes ago    Up 21 minutes
439ff3c52280       ubuntu:16.04      "bash"            34 minutes ago    Up 18 minutes
a35e0bc3f6db       ubuntu:16.04      "ps -ef"          7 hours ago     Exited (0) 7 hours ago
cc4d66df724c       ubuntu:16.04      "echo 'hello world'" 7 hours ago     Exited (0) 7 hours ago
root@Nx:/home/mucio# 

```

### 3.3 Opções de log

1. Vimos anteriormente como ler todo o log do PID 1 de um contêiner. Também podemos usar alguns sinalizadores flags para controlar quais logs são exibidos. **--tail n** limita a exibição para as últimas **n** linhas; Experimentá-lo com o contêiner que deve estar sendo executado a partir da última etapa:

```
$ docker container logs --tail 5 <container ID>
```

```

root@Nx:/home/mucio#
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container logs --tail 5 4846b84a8adf
64 bytes from 127.0.0.1: icmp_seq=347 ttl=64 time=0.049 ms

--- 127.0.0.1 ping statistics ---
347 packets transmitted, 347 received, 0% packet loss, time 354293ms
rtt min/avg/max/mdev = 0.027/0.058/0.202/0.020 ms
root@Nx:/home/mucio# 

```

2. Também podemos seguir os logs conforme são gerados com **-f**:

```
$ docker container logs -f <containerID>
```

```
root@Nx:/home/mucio# docker container logs -f 4846b84a8adf
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.032 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.128 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.046 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.029 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.202 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.067 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=11 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=14 ttl=64 time=0.099 ms
64 bytes from 127.0.0.1: icmp_seq=15 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=16 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=17 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=18 ttl=64 time=0.076 ms
64 bytes from 127.0.0.1: icmp_seq=19 ttl=64 time=0.074 ms
64 bytes from 127.0.0.1: icmp_seq=20 ttl=64 time=0.049 ms
64 bytes from 127.0.0.1: icmp_seq=21 ttl=64 time=0.051 ms
64 bytes from 127.0.0.1: icmp_seq=22 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=23 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=24 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=25 ttl=64 time=0.041 ms
64 bytes from 127.0.0.1: icmp_seq=26 ttl=64 time=0.058 ms
```

(CTRL+C para sair do modo seguinte e/ou seguidor).

3. Finalmente, tente combinar as flags **tail** e **follow** para começar a seguir os registros de um ponto anterior.

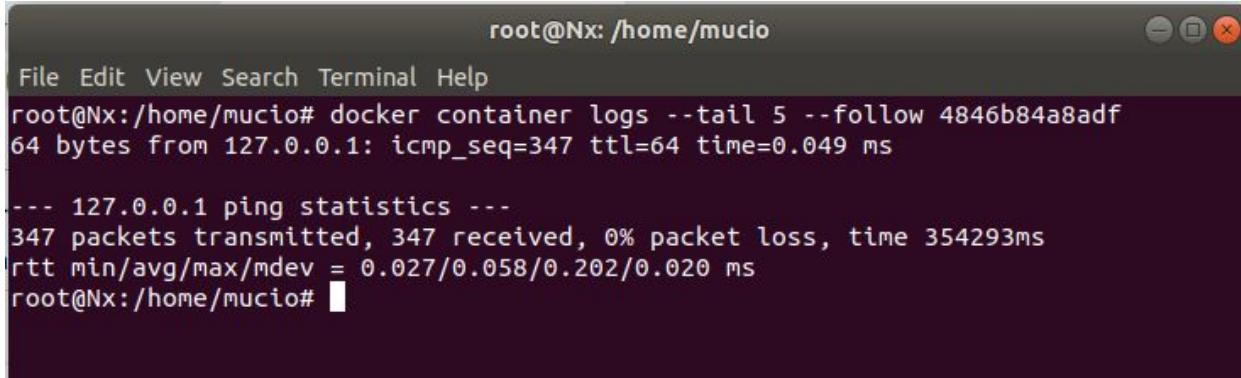
```
root@Nx:/home/mucio# docker container logs --tail --follow 4846b84a8adf
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.032 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.128 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.046 ms
```

Continuação...

```
64 bytes from 127.0.0.1: icmp_seq=344 ttl=64 time=0.045 ms
64 bytes from 127.0.0.1: icmp_seq=345 ttl=64 time=0.083 ms
64 bytes from 127.0.0.1: icmp_seq=346 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=347 ttl=64 time=0.049 ms

--- 127.0.0.1 ping statistics ---
347 packets transmitted, 347 received, 0% packet loss, time 354293ms
rtt min/avg/max/mdev = 0.027/0.058/0.202/0.020 ms
root@Nx:/home/mucio#
```

Ou também:



A screenshot of a terminal window titled "root@Nx: /home/mucio". The window has a standard Linux desktop interface with a title bar and window controls. The terminal content shows the output of the command "docker container logs --tail 5 --follow 4846b84a8adf". It displays several lines of log output, including ICMP traffic and ping statistics, identical to the previous screenshot.

```
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container logs --tail 5 --follow 4846b84a8adf
64 bytes from 127.0.0.1: icmp_seq=347 ttl=64 time=0.049 ms

--- 127.0.0.1 ping statistics ---
347 packets transmitted, 347 received, 0% packet loss, time 354293ms
rtt min/avg/max/mdev = 0.027/0.058/0.202/0.020 ms
root@Nx:/home/mucio#
```

### 3.4 Conclusão

Neste cenário, vimos como executar processos em background [segundo plano]. In this scenario, we saw how to run processes in the background, attach to them and inspect their logs. Também vimos um exemplo explícito de como matar o PID 1 em um container que mata o próprio container.

## 4 Iniciando, Parando, Ispencionando e Deletando Containers

### 4.1 Iniciando e Reiniciando Containers

1. Comece executando um servidor de tomcat em segundo plano e verifique se ele está realmente em execução:

```
$ docker container run -d tomcat
$ docker container ls
```

```

root@Nx:/home/mucio#
File Edit View Search Terminal Help

root@Nx:/home/mucio# docker container run -d tomcat
Unable to find image 'tomcat:latest' locally
latest: Pulling from library/tomcat
6f2f362378c5: Pull complete
494c27a8a6b8: Pull complete
7596bb83081b: Pull complete
372744b62d49: Pull complete
fb72767f9beb: Pull complete
3fe571af508a: Pull complete
3e6725074325: Pull complete
9ffd5dadda90: Pull complete
4e051c2969b0: Pull complete
b326a8316680: Pull complete
Digest: sha256:cea26a23e1ebdbbebdddde1e02a10e655b0b386d8de6002301a037a08be87a12f
Status: Downloaded newer image for tomcat:latest
4fa78b5d3c79649c9710336214f9e82a2d22f5d8fa916a3bb1f714544f080b29
root@Nx:/home/mucio# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
STATUS              PORTS              NAMES
4fa78b5d3c79        tomcat              "catalina.sh run"   About a minute ago
Up About a minute   8080/tcp           mystifying_mirzakhani
root@Nx:/home/mucio#

```

2. Pare o container usando docker container stop, e verifique se o container está realmente parado:

```

$ docker container stop <container ID>
$ docker container ls -a

```

```

root@Nx:/home/mucio#
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container stop 4fa78b5d3c79
4fa78b5d3c79
root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS             NAMES
4fa78b5d3c79        tomcat              "catalina.sh run"   4 minutes ago     Exited (143) 8 seconds ago
akhani
ac7a0beb6fc5        ubuntu:14.04       "ping 127.0.0.1"    14 hours ago      Exited (137) 14 hours ago
4846bb84a8df        ubuntu:14.04       "ping 127.0.0.1"    14 hours ago      Exited (0) 14 hours ago
tia
e7d74dfd4ca3        ubuntu:14.04       "ping 127.0.0.1 -c 10" 14 hours ago      Exited (0) 14 hours ago
y
9d01aa5b4291        ubuntu:16.04       "bash"
439ff3c52280        ubuntu:16.04       "bash"
a35e0bc3f6db        ubuntu:16.04       "ps -ef"
cc4d66df724c        ubuntu:16.04       "echo 'hello world'" 21 hours ago      Exited (0) 21 hours ago
root@Nx:/home/mucio# clear

```

3. Inicie o container novamente com docker container start, e anexar a ele ao mesmo tempo:

```
$ docker container start -a <containerID>
```

```
root@Nx:/home/mucio# docker container start -a 4fa78b5d3c79
02-Jul-2019 16:45:41.498 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version: Apache Tomcat/8.5.42
02-Jul-2019 16:45:41.502 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built: Jun 4 2019 20:29:04 UTC
02-Jul-2019 16:45:41.502 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server number: 8.5.42.0
02-Jul-2019 16:45:41.502 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name: Linux
02-Jul-2019 16:45:41.502 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version: 4.18.0-25-generic
02-Jul-2019 16:45:41.502 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture: amd64
02-Jul-2019 16:45:41.502 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home: /usr/local/openjdk-8/jre
02-Jul-2019 16:45:41.503 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version: 1.8.0_212-b04
02-Jul-2019 16:45:41.503 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor: Oracle Corporation
02-Jul-2019 16:45:41.503 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_BASE: /usr/local/tomcat
02-Jul-2019 16:45:41.503 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_HOME: /usr/local/tomcat
02-Jul-2019 16:45:41.504 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.util.logging.config.file=/local/tomcat/conf/logging.properties
02-Jul-2019 16:45:41.504 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
02-Jul-2019 16:45:41.504 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djdk.tls.ephemeralDHKeySize=0
02-Jul-2019 16:45:41.504 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.protocol.handler.pkgs=org.apache.catalina.webresources
02-Jul-2019 16:45:41.504 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dorg.apache.catalina.securityListener.UMASK=0027
```

- Desanexar e parar o container com CTRL+C, então reinicie o container sem anexar e seguir os logs a partir das 10 lines anteriores.

```
root@Nx:/home/mucio#
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container restart 4fa78b5d3c79
4fa78b5d3c79
root@Nx:/home/mucio#
```

- Finalmente, pare o container com docker container kill:

```
$ docker container kill <containerID>
```

```
root@Nx:/home/mucio# docker container kill 4fa78b5d3c79
4fa78b5d3c79
root@Nx:/home/mucio#
```

Ambos stop e kill manda/enviam uma SIGKILL para PID 1 no container; a diferença é que stop primeiro envia/manda um SIGTERM, aguarda um período de carência (padrão 10 segundos) antes de enviar o SIGKILL, while kill dispara o SIGKILL imediatamente.

## 4.2 Inspecionando um contêiner com contêiner docker inspecionar

- Inicie seu servidor TomCat novamente, em seguida, inspecione os detalhes do container usando docker container inspect:

```
$ docker container start <containerID>
$ docker container inspect <container ID>
```

```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container start 4fa78b5d3c79
4fa78b5d3c79
root@Nx:/home/mucio# docker container inspect 4fa78b5d3c79
[
  {
    "Id": "4fa78b5d3c79649c9710336214f9e82a2d22f5d8fa916a3bb1f714544f080b29",
    "Created": "2019-07-02T16:37:59.083757332Z",
    "Path": "catalina.sh",
    "Args": [
      "run"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 7303,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2019-07-02T16:53:38.558025701Z",
      "FinishedAt": "2019-07-02T16:52:10.266761011Z"
    },
    "Image": "sha256:5377fd8533c34a4b1a909ff56d1ccc5b61f8bda1901b1c3c7e69cc3534cf25c2",
    "ResolvConfPath": "/var/lib/docker/containers/4fa78b5d3c79649c9710336214f9e82a2d22f5d8fa916a3bb1f714544f080b29/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/4fa78b5d3c79649c9710336214f9e82a2d22f5d8fa916a3bb1f714544f080b29/hostname",
    "HostsPath": "/var/lib/docker/containers/4fa78b5d3c79649c9710336214f9e82a2d22f5d8fa916a3bb1f714544f080b29/hosts",
    "LogPath": "/var/lib/docker/containers/4fa78b5d3c79649c9710336214f9e82a2d22f5d8fa916a3bb1f714544f080b29/4fa78b5d3c79649c9710336214f9e82a2d22f5d8fa916a3bb1f714544f080b29-json.log",
    ...
  }
]
```

Continuando [Parte final do retorno]...

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
    "SandboxKey": "/var/run/docker/netns/d5733c7efec",
    "SecondaryIPAddresses": null,
    "SecondaryIPv6Addresses": null,
    "EndpointID": "91789659fc772693bc8e209d020b9f393267892fb8625a1a05f2e41802863c2f",
    "Gateway": "172.17.0.1",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "172.17.0.2",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "MacAddress": "02:42:ac:11:00:02",
    "Networks": [
        "bridge": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": null,
            "NetworkID": "c88155b25ff3c5c0d4a7932d707460aa664eeaaaae786efba7cf6ca37f5c70
18e",
            "EndpointID": "91789659fc772693bc8e209d020b9f393267892fb8625a1a05f2e4180286
3c2f",
            "Gateway": "172.17.0.1",
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:11:00:02",
            "DriverOpts": null
        }
    }
]
root@Nx:/home/mucio#
```

2. Encontre o IP do container e o ID longo do container no JSON de saída do comando inspect. Se você souber o nome da chave da propriedade que está procurando, experimente canalizar para o grep:

```
$ docker container inspect <container ID> | grep IPAddress
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container inspect 4fa78b5d3c79 | grep IPAddress
    "SecondaryIPAddresses": null,
    "IPAddress": "172.17.0.2",
    "IPAddress": "172.17.0.2",
root@Nx:/home/mucio#
```

3. Agora tente grepping para Cmd, o comando PID 1 sendo executado por este contêiner. A pesquisa de texto simples do grep nem sempre retorna resultados úteis.

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container inspect 4fa78b5d3c79 | grep Cmd
    "Cmd": [
root@Nx:/home/mucio#
```

4. Outra maneira de filtrar esse JSON é com a flag `--format`. A sintaxe segue o pacote de texto/modelo da Go: <http://golang.org/pkg/text/template/>. Por exemplo, para encontrar o valor de `Cmd` que tentamos para o grep acima, tente:

```
$ docker container inspect --format='{{.Config.Cmd}}' <container ID>
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container inspect --format='{{.Config.Cmd}}' 4fa78b5d3c79
[catalina.sh run]
root@Nx:/home/mucio#
```

5. Chaves aninhadas no JSON retornado por o comando `docker container inspect` podem ser encadeados juntos dessa maneira.

Tente modificar o exemplo para retornar o IP address do grep feito anteriormente.

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container inspect --format='{{.NetworkSettings.Networks.bridge.IPAddress}}' 4fa78b5d3c79
172.17.0.2
root@Nx:/home/mucio#
```

6. Finalmente, podemos extrair todos pares chave/valor para um dado objeto usando uma função Json:

```
$ docker container inspect --format='{{json .Config}}' <container ID>
```

```

root@Nx:/home/mucio# docker container inspect --format='[{"NetworkSettings.Networks.bridge.IPAddress"}]' 4fa78b5d3c79
172.17.0.2
root@Nx:/home/mucio# docker container inspect --format='[{"json .Config"}]' 4fa78b5d3c79
[{"Hostname":"4fa78b5d3c79","Domainname":"","User":"","AttachStdin":false,"AttachStdout":false,"AttachStderr":false,"ExposedPorts":{"8080/tcp":{}}, "Tty":false,"OpenStdin":false,"StdinOnce":false,"Env":["PATH=/usr/local/tomcat/bin:/usr/local/openjdk-8/bin:/usr/local/bin:/usr/sbin:/usr/bin:/bin","LANG=C.UTF-8","JAVA_HOME=/usr/local/openjdk-8","JAVA_VERSION=8u212-b04","JAVA_BASE_URL=https://github.com/AdoptOpenJDK/openjdk8-upstream-binaries/releases/download/jdk8u212-b04/OpenJDK8U-","JAVA_URL_VERSION=8u212b04","CATALINA_HOME=/usr/local/tomcat","TOMCAT_NATIVE_LIBDIR=/usr/local/tomcat/native-jni-lib","LD_LIBRARY_PATH=/usr/local/tomcat/native-jni-lib","GPG_KEYS=05AB33110949707C93A279E3D3EFE6B686867BA6 07E48665A34DCAFAE522E5E6266191C37C037D42 47309207D818FD8DCD3F83F1931D684307A10A5 541FBE7D8F78B25E055DDEE13C370389288584E7 61B832AC2F1C5A90F0F9B00A1C506407564C17A3 713DA88BE50911535FE716F5208B0AB1D63011C7 79F7026C690BA50B92CD8866A3AD3F4F22C4FED 9BA44C2621385CB966EBA586F72C284D731FABEE A27677289986DB50844682F8ACB77FC2E86E29AC A9C5DF4D22E99998D9875A5110C01C5A2F6059E7 DCFD35E0BF8CA7344752DE8B6FB21E8933C60243 F3A04C595DB5B6A5F1ECA43E3B7BBB100D811B8E F7DA48BB64BCB84ECBA7EE6935CD23C10D498E23","TOMCAT_MAJOR=8","TOMCAT_VERSION=8.5.42","TOMCAT_SHA512=3e6b38e48d315d142e96f8e3809c86632f3c3903f8751c6602581a587edf840893ffoc737a65fcf9560a495b0118b5b8d60d41ce7947fe2abe34a89839b640f","TOMCAT_TGZ_URLS=https://www.apache.org/dyn/closer.cgi?action=download&filename=tomcat/tomcat-8/v8.5.42/bin/apache-tomcat-8.5.42.tar.gz \thhttps://www-us.apache.org/dist/tomcat/tomcat-8/v8.5.42/bin/apache-tomcat-8.5.42.tar.gz \thhttps://archive.apache.org/dist/tomcat/tomcat-8/v8.5.42/bin/apache-tomcat-8.5.42.tar.gz \thhttps://www.apache.org/dyn/closer.cgi?action=download&filename=tomcat/tomcat-8/v8.5.42/bin/apache-tomcat-8.5.42.tar.gz.asc \thhttps://www.apache.org/dist/tomcat/tomcat-8/v8.5.42/bin/apache-tomcat-8.5.42.tar.gz.asc \thhttps://archive.apache.org/dist/tomcat/tomcat-8/v8.5.42/bin/apache-tomcat-8.5.42.tar.gz.asc","Cmd":["catalina.sh","run"],"ArgsEscaped":true,"Image":"tomcat","Volumes":null,"WorkingDir":"/usr/local/tomcat","Entrypoint":null,"OnBuild":null,"Labels":{}}
root@Nx:/home/mucio#

```

## 4.3 Deletando Containers

1. Inicie três contêineres no modo de segundo plano e, em seguida, pare o primeiro.

```

root@Nx:/home/mucio# docker container run -d ubuntu:16.04
ec6a98278017bfbca5213e0e0c6c4432ac7f4a3b478de7a10b9b07f46f3fc52f
root@Nx:/home/mucio# docker container run -d ubuntu:16.04
bb98ded9cc45f2e7079bd097f1a30af4cd1f327815b2bc60916fc5f2b289e3d
root@Nx:/home/mucio# docker container run -d ubuntu:16.04
6f03966bcc07fab5c0bdf07b112d6087086c0d6f118c71b418a7e4750a0c9cce
root@Nx:/home/mucio# docker container stop ec6a98278017bfbca5213e0e0c6c4432ac7f4a3b478de7a10b9b07f46f3fc52f
ec6a98278017bfbca5213e0e0c6c4432ac7f4a3b478de7a10b9b07f46f3fc52f
root@Nx:/home/mucio#

```

2. Liste somente containers que tiveram esse parâmetro “exited” usando a flag `--filter` que aprendemos anteriormente e a opção `status=exited`.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6f03966bcc07	ubuntu:16.04	"bin/bash"	2 minutes ago	Exited (0) 2 minutes ago		romantic_k
bb98ded9cc45	ubuntu:16.04	"bin/bash"	2 minutes ago	Exited (0) 2 minutes ago		affectiona
rley	ubuntu:16.04	"bin/bash"	2 minutes ago	Exited (0) 2 minutes ago		
ec6a98278017	ubuntu:16.04	"bin/bash"	2 minutes ago	Exited (0) 2 minutes ago		jovial_sir
ac7a0beb6fc5	ubuntu:14.04	"ping 127.0.0.1"	15 hours ago	Exited (137) 15 hours ago		dazzling_r
4846b84a8adf	ubuntu:14.04	"ping 127.0.0.1"	15 hours ago	Exited (0) 15 hours ago		suspicious
ia	ubuntu:14.04	"ping 127.0.0.1 -c 10"	15 hours ago	Exited (0) 15 hours ago		
e7d74dfd4ca3	ubuntu:14.04	"ping 127.0.0.1 -c 10"	15 hours ago	Exited (0) 15 hours ago		laughing_s
9d01aa5b4291	ubuntu:16.04	"bash"	16 hours ago	Exited (0) 15 hours ago		thirsty_hu
439ff3c52280	ubuntu:16.04	"bash"	16 hours ago	Exited (0) 15 hours ago		quirky_min
a35e0b3c3fdb	ubuntu:16.04	"ps -ef"	22 hours ago	Exited (0) 22 hours ago		pedantic_b
cc4d66df724c	ubuntu:16.04	"echo 'hello world'"	22 hours ago	Exited (0) 22 hours ago		stupefied

3. Delete/exclua o container que você parou acima com docker container rm, e faça a mesma operação de

listagem acima para confirmar que foi removida:

```
$ docker container rm <container ID>
$ docker container ls ...
```

```
root@Nx:/home/mucio# docker container rm ec6a98278017bfbc5213e0e0c6c4432ac7f4a3b478de7a10b9b07f46f3fc52f
ec6a98278017bfbc5213e0e0c6c4432ac7f4a3b478de7a10b9b07f46f3fc52f
root@Nx:/home/mucio# docker container ls -a --filter status=exited
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
6f03966bcc07      ubuntu:16.04       "/bin/bash"         3 minutes ago     Exited (0) 3 minutes ago
bb98ded9cc45      ubuntu:16.04       "/bin/bash"         3 minutes ago     Exited (0) 3 minutes ago
rley                ac7a0beb6fc5      "ping 127.0.0.1"   15 hours ago      Exited (137) 15 hours ago
4846b84a8adf      ubuntu:14.04       "ping 127.0.0.1"   15 hours ago      Exited (0) 15 hours ago
ia                 e7d74df4ca3      "ping 127.0.0.1 -c 10" 15 hours ago      Exited (0) 15 hours ago
9d01aa5b4291      ubuntu:16.04       "bash"              16 hours ago      Exited (0) 15 hours ago
439ff3c52280      ubuntu:16.04       "bash"              16 hours ago      Exited (0) 15 hours ago
a35e0bc3f6db      ubuntu:16.04       "ps -ef"            22 hours ago      Exited (0) 22 hours ago
cc4d66df724c      ubuntu:16.04       "echo 'hello world'" 22 hours ago      Exited (0) 22 hours ago
root@Nx:/home/mucio# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
4fa78b5d3c79      tomcat              "catalina.sh run"  About an hour ago Up     About an hour    8080/tcp
root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
6f03966bcc07      ubuntu:16.04       "/bin/bash"         4 minutes ago     Exited (0) 4 minutes ago
bb98ded9cc45      ubuntu:16.04       "/bin/bash"         4 minutes ago     Exited (0) 4 minutes ago
rley                4fa78b5d3c79      "catalina.sh run"  About an hour ago Up     About an hour    8080/tcp
ia                 ac7a0beb6fc5      "ping 127.0.0.1"   15 hours ago      Exited (137) 15 hours ago
4846b84a8adf      ubuntu:14.04       "ping 127.0.0.1"   15 hours ago      Exited (0) 15 hours ago
ia                 e7d74df4ca3      "ping 127.0.0.1 -c 10" 15 hours ago      Exited (0) 15 hours ago
9d01aa5b4291      ubuntu:16.04       "bash"              16 hours ago      Exited (0) 15 hours ago
439ff3c52280      ubuntu:16.04       "bash"              16 hours ago      Exited (0) 15 hours ago
a35e0bc3f6db      ubuntu:16.04       "ps -ef"            22 hours ago      Exited (0) 22 hours ago
cc4d66df724c      ubuntu:16.04       "echo 'hello world'" 22 hours ago      Exited (0) 22 hours ago
root@Nx:/home/mucio#
```

4. Agora faça o mesmo com um dos contêineres que ainda estão em execução; Aviso: docker container rm, não excluirá um container que está em execução, a menos que passemos a flag -f, forçando a execução do comando [f - force]. Delete o segundo container iniciado acima:

```
$ docker container rm -f <container ID>
```

```

root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
6f03966bcc07      ubuntu:16.04       "/bin/bash"            6 minutes ago      Exited (0) 6 minutes ago
bb98ded9cc45      ubuntu:16.04       "/bin/bash"            6 minutes ago      Exited (0) 6 minutes ago
rley
4fa78b5d3c79      tomcat             "catalina.sh run"    About an hour ago Up     About an hour          8080/tcp
khani
ac7a0beb6fc5      ubuntu:14.04       "ping 127.0.0.1"      15 hours ago       Exited (137) 15 hours ago
4846b84a8adf      ubuntu:14.04       "ping 127.0.0.1"      15 hours ago       Exited (0) 15 hours ago
ia
e7d74dfd4ca3      ubuntu:14.04       "ping 127.0.0.1 -c 10" 15 hours ago       Exited (0) 15 hours ago
9d01aa5b4291      ubuntu:16.04       "bash"                16 hours ago       Exited (0) 15 hours ago
439ff3c52280      ubuntu:16.04       "bash"                16 hours ago       Exited (0) 15 hours ago
a35e0bc3f6db      ubuntu:16.04       "ps -ef"              22 hours ago       Exited (0) 22 hours ago
cc4d66df724c      ubuntu:16.04       "echo 'hello world'" 22 hours ago       Exited (0) 22 hours ago
root@Nx:/home/mucio# docker container rm -f bb98ded9cc45
bb98ded9cc45
root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
6f03966bcc07      ubuntu:16.04       "/bin/bash"            7 minutes ago      Exited (0) 7 minutes ago
4fa78b5d3c79      tomcat             "catalina.sh run"    About an hour ago Up     About an hour          8080/tcp
khani
ac7a0beb6fc5      ubuntu:14.04       "ping 127.0.0.1"      15 hours ago       Exited (137) 15 hours ago
4846b84a8adf      ubuntu:14.04       "ping 127.0.0.1"      15 hours ago       Exited (0) 15 hours ago
ia
e7d74dfd4ca3      ubuntu:14.04       "ping 127.0.0.1 -c 10" 15 hours ago       Exited (0) 15 hours ago
9d01aa5b4291      ubuntu:16.04       "bash"                16 hours ago       Exited (0) 15 hours ago
439ff3c52280      ubuntu:16.04       "bash"                16 hours ago       Exited (0) 15 hours ago
a35e0bc3f6db      ubuntu:16.04       "ps -ef"              22 hours ago       Exited (0) 22 hours ago
cc4d66df724c      ubuntu:16.04       "echo 'hello world'" 22 hours ago       Exited (0) 22 hours ago
root@Nx:/home/mucio#

```

5. Tente usar as flags do comando `docker container ls` que aprendemos anteriormente para remover o último contêiner que foi executado, ou todos os contêineres parados

```

root@Nx:/home/mucio#
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker rm $(docker ps -a -q --filter "status=exited")
6f03966bcc07
ac7a0beb6fc5
4846b84a8adf
e7d74dfd4ca3
9d01aa5b4291
439ff3c52280
a35e0bc3f6db
cc4d66df724c
root@Nx:/home/mucio#

```

Tente usando o `docker container` com a flag `ls`, assim, `docker container ls`, aprendemos anteriormente a remover o último contêiner que foi executado ou todos os contêineres interrompidos. Lembre-se de que você pode passar a saída de um comando de shell cmd-A para uma variável de outro comando cmd-B com sintaxe como, por exemplo cmd-B `$(cmd-A)`.

**Exemplo na prática:** Você pode iniciar todas instâncias/containers, suponha que você tenha que subir vários containers ao mesmo tempo, é só fazer isso: `docker start $(docker ps -a -q)`

## 4.4 Conclusão

Neste cenário, você aprendeu como usar `docker container start`, `stop`, `rm` e `kill`. Você também viu o comando `docker container inspect`, que retorna metadados sobre um determinado container.

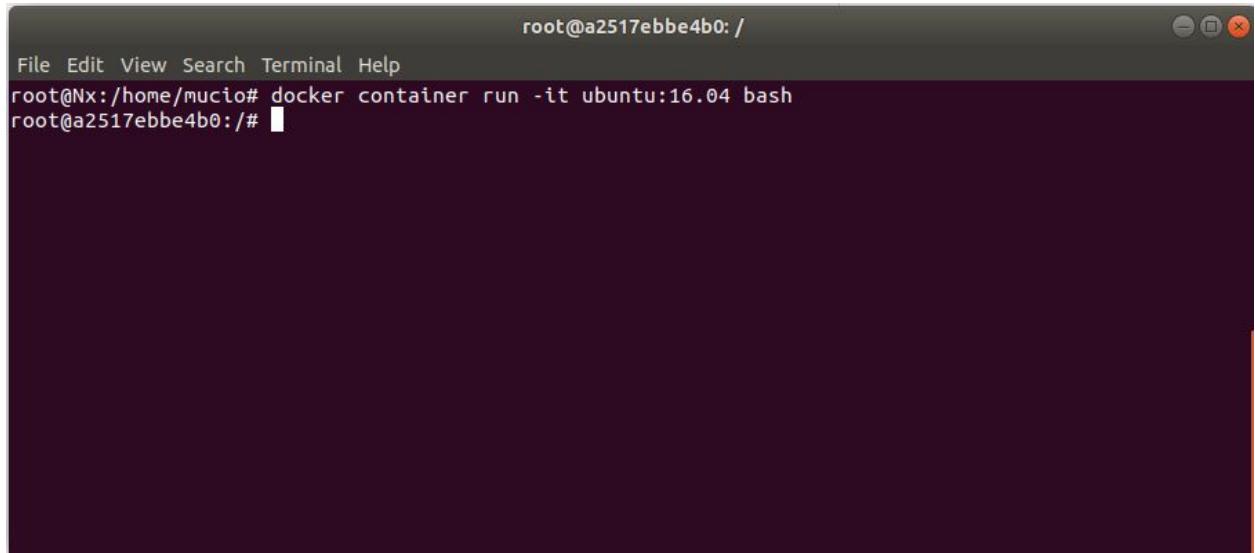
In this scenario, you learned how to use `docker` to start, stop and delete containers. You also saw the `docker container inspect` command, which returns metadata about a given container.

## 5 Criação interativa de imagens

### 5.1 Modificando um Container

1. Inicie um terminal bash em um container ubuntu:

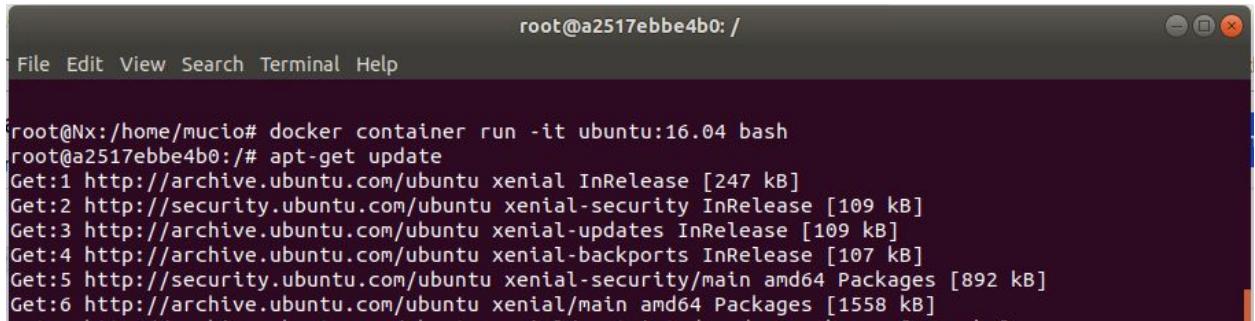
```
$ docker container run -it ubuntu:16.04 bash
```



```
root@a2517ebbe4b0: /  
File Edit View Search Terminal Help  
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash  
root@a2517ebbe4b0:/#
```

2. Instale um par de software neste contêiner (`vim` e `wget`) - não há nada de especial sobre ambos, são utilitários simples linux, qualquer alteração no sistema de arquivos será necessária. Depois, saia do contêiner:

```
$ apt-get update
```



```
root@a2517ebbe4b0: /  
File Edit View Search Terminal Help  
root@Nx:/home/mucio# docker container run -it ubuntu:16.04 bash  
root@a2517ebbe4b0:/# apt-get update  
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]  
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]  
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]  
Get:4 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]  
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [892 kB]  
Get:6 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]  
Get:7 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]  
Get:8 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
```

```
$ apt-get install -y wget vim
```

```

root@a2517ebbe4b0:/# apt-get install -y wget vim
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates file libexpat1 libgpm2 libidn11 libmagic1 libmpdec2 libpython3.5
  libpython3.5-minimal libpython3.5-stdlib libsqlite3-0 libssl1.0.0 mime-support openssl
  vim-common vim-runtime
Suggested packages:
  gpm ctags vim-doc vim-scripts vim-gnome-py2 | vim-gtk-py2 | vim-gtk3-py2 | vim-athena-py2
  | vim-nox-py2
The following NEW packages will be installed:
  ca-certificates file libexpat1 libgpm2 libidn11 libmagic1 libmpdec2 libpython3.5

$ exit

```

```

root@Nx: /home/mucio
File Edit View Search Terminal Help
Processing triggers for ca-certificates (20170717~16.04.2) ...
Updating certificates in /etc/ssl/certs...
148 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
root@a2517ebbe4b0:/# exit
exit
root@Nx:/home/mucio#

```

3. Finalmente, tente docker container diff para ver o que mudou em relação a um contêiner em relação à sua imagem. você precisa pegar o ContainerID, via docker container ls -a, primeiro:

```

$ docker container ls -a
$ docker container diff <container id>

```

```

root@Nx: /home/mucio
File Edit View Search Terminal Help

root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS
           PORTS     NAMES
a2517ebbe4b0        ubuntu:16.04      "bash"        9 minutes ago   Exited (0) 2 minute
5 ago              gracious_knuth
4fa78b5d3c79        tomcat:           "catalina.sh run"  2 hours ago    Up About an hour
          8080/tcp      mystifying_mirzakhani
root@Nx:/home/mucio# docker container diff a2517ebbe4b0
C /etc
C /etc/ld.so.cache
A /etc/wgetrc
A /etc/mailcap.order
A /etc/ca-certificates
A /etc/ca-certificates/update.d
A /etc/magic.mime
A /etc/python3.5
A /etc/python3.5/sitecustomize.py
A /etc/ssl
A /etc/ssl/certs
A /etc/ssl/certs/DuoVadis Root CA.pem

```

Certifique-se de que os resultados do diff fazem sentido para você antes de prosseguir.

## 5.2 Capturando Estado do Container como uma Imagem com docker container commit

1. A instalação do wget e do vim na última etapa escreveu informações na camada de leitura / gravação do contêiner. Agora, vamos salvar essa camada de leitura / gravação como uma nova camada de imagem somente leitura, a fim de criar uma nova imagem que reflita nossas adições, por meio do docker container commit:

```
$ docker container commit <container id> <your docker store ID>/myapp:1.0
```

```
root@Nx:/home/mucio# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
a2517ebbe4b0        ubuntu:16.04       "bash"              30 minutes ago   Exited (0) 23 minutes ago
es ago
4fa78b5d3c79        tomcat             "catalina.sh run"  2 hours ago      Up 2 hours
          8080/tcp
mystifying_mirzakhani
root@Nx:/home/mucio# docker container commit a2517ebbe4b0 muciojaziel/myapp:1.0
sha256:885fc6e2fbccfaa5b00f252149842ef88986542d9444b9008de369d8a095eb819
root@Nx:/home/mucio#
```

Observação: Aqui é preciso ter cadastrado sua conta no PWD. Substitua meu nome por sua Docker ID do PWD.

2. Verifique se você pode ver sua nova imagem listando todas as suas imagens:

```
$ docker image ls
```

```
root@Nx:/home/mucio# docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
muciojaziel/myapp  1.0      885fc6e2fbccf  2 minutes ago  205MB
dbaontap/node-divide-app  latest  a6301120b6cc  6 days ago    165MB
dbaontap/node-multiply-app  latest  56b756f6d340  6 days ago    165MB
dbaontap/node-subtract-app  latest  104488c48925  6 days ago    165MB
dbaontap/node-add-app     latest  bc00e1413ed6  6 days ago    165MB
oraclelinux          7-slim   d94f4e9e5c13  7 days ago    118MB
ubuntu               16.04   13c9f1285025  13 days ago   119MB
tomcat               latest   5377fd8533c3  2 weeks ago   506MB
ubuntu               14.04   2c5e00d77a67  6 weeks ago   188MB
dbaontap/node-add-app  <none>   dc126322ee0c  20 months ago  202MB
root@Nx:/home/mucio#
```

3. Crie um container executando o bash usando sua nova imagem, e verifique que o vim and wget estão instalados:

```
$ docker container run -it <your docker store ID>/myapp:1.0 bash
$ which vim
$ which wget
```

```
root@675868a54a1e:/ # docker container run -it muciojaziel/myapp:1.0 bash
root@675868a54a1e:/ # which vim
/usr/bin/vim
root@675868a54a1e:/ # which wget
/usr/bin/wget
root@675868a54a1e:/ #
```

4. Crie um arquivo em seu contêiner e confirme isso como uma nova imagem. Use o mesmo nome de imagem, mas marque-o como 1.1.

```
root@Nx:/home/mucio# ls
bin dev home lib64 mnt      opt root sbin sys usr
boot etc lib media novoArquivo.txt proc run srv tmp var
root@675868a54a1e:/ # exit
root@Nx:/home/mucio# docker container commit 675868a54a1e muciojaziel/myapp:1.1
sha256:6411ab568f05a3789b344074fc724166dd1bd2817a49784398434ee952e3cea2
root@Nx:/home/mucio# docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
muciojaziel/myapp   1.1      6411ab568f05  22 seconds ago  205MB
muciojaziel/myapp   1.0      885fc6e2fbef  8 minutes ago  205MB
dbaontap/node-divide-app    latest    a6301128b6ea  6 days ago   165MB
dbaontap/node-multiply-app   latest    56b756f6d340  6 days ago   165MB
dbaontap/node-subtract-app   latest    104488c48925  6 days ago   165MB
dbaontap/node-add-app       latest    bc00e1413ed6  6 days ago   165MB
oraclelinux          7-slim   d94f4e9e5c13  7 days ago   118MB
ubuntu               16.04   13c9f1285025  13 days ago  119MB
tomcat               latest   5377fd8533c3  2 weeks ago  506MB
ubuntu               14.04   2c5e00d77a67  6 weeks ago  188MB
dbaontap/node-add-app   <none>  dc126322ee0c  20 months ago 202MB
root@Nx:/home/mucio#
```

5. Finalmente, execute `docker container diff` no seu contêiner mais recente; a saída faz sentido? Fazer o que você acha que os prefixos A, C e D no começo de cada linha significam?

```
root@Nx:/home/mucio# docker container diff 675868a54a1e
A /novoArquivo.txt
C /root
C /root/.bash_history
root@Nx:/home/mucio#
```

### 5.3 Conclusão

Neste exercício, você viu como inspecionar o conteúdo da leitura / gravação de um contêiner posteriormente com `docker container diff`, e commitar essas alterações em uma nova camada de imagem com `docker`

container commit.

## 6 Criando Images com Dockerfiles (1/2)

### 6.1 Escrevendo e fazendo Building um Dockerfile

1. Crie uma pasta chamada myimage, e um arquivo de texto chamado Dockerfile, dentro dessa pasta. No Dockerfile, inclua as seguintes instruções:

1. FROM ubuntu:16.04
- 2.
3. RUN apt-get update
4. RUN apt-get install -y iutils-ping

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# mkdir myimage
```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Help
root@Nx:/home/mucio# cd myimage
root@Nx:/home/mucio/myimage# touch Dockerfile
```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Help
root@Nx:/home/mucio/myimage# vim Dockerfile
```

Observação: Depois de digitar,

- tecle: Esc.
  - Digite : (dois pontos)
  - digite wq
  - O resultado será :wq (você está salvando e saindo do arquivo)
  - Tecle enter

2. Construa sua imagem com o comando build:

```
$ docker image build -t <username>/myimage .
```

```
root@Nx:/home/mucio/myimage
File Edit View Search Terminal Help
root@Nx:/home/mucio/myimage# docker image build -t muciojaziel/myimage .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/3 : RUN apt-get update
--> Running in e36fe5d80d7f
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [893 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Get:6 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [12.7 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [567 kB]
Get:9 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages [6121 B]
Get:10 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [176 kB]
```

Observação: Lembre-se do ponto, como no print.

```
sl27 amd64 3.4.10-4ubuntu1.5 [22.0 kB]
Get:10 http://archive.ubuntu.com/ubuntu xenial/main amd64 iutils-ping amd64 3:2
0121221-5ubuntu2 [52.7 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 1305 kB in 1s (847 kB/s)
Selecting previously unselected package libgmp10:amd64.
(Reading database ... 4777 files and directories currently installed.)
Preparing to unpack .../libgmp10_2%3a6.1.0+dfsg-2_amd64.deb ...
```

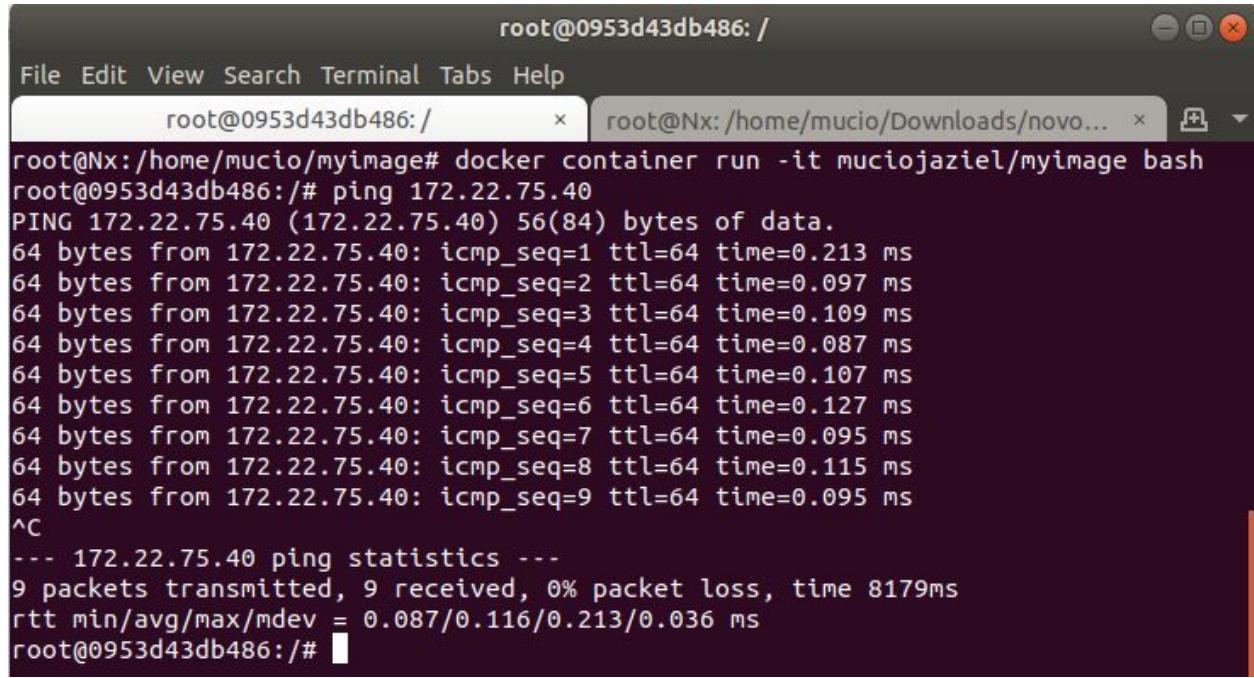
Observação: Alguns avisos, é normal.

```
Setting up tputts-ping (3.20121221-5ubuntu2) ...
Setcap is not installed, falling back to setuid
Processing triggers for libc-bin (2.23-0ubuntu11) ...
Removing intermediate container 967679b3c577
--> 05b70822e55c
Successfully built 05b70822e55c
Successfully tagged muciojaziel/myimage:latest
root@Nx:/home/mucio/myimage#
```

3. Verifique se sua nova imagem existe com docker image ls, em seguida, use-o para executar um contêiner e pingar algo de dentro desse contêiner.

```
root@Nx:/home/mucio/myimage
File Edit View Search Terminal Help
root@Nx:/home/mucio/myimage# docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED             SIZE
muciojaziel/myimage    latest   05b70822e55c  3 minutes ago  149MB
go                  1.1      6411ab568f05  47 hours ago   205MB
muciojaziel/myapp      1.0      885fc6e2fbcf  2 days ago     205MB
dbaontap/node-divide-app  latest   a6301128b6ea  8 days ago     165MB
dbaontap/node-multiply-app  latest   56b756f6d340  8 days ago     165MB
dbaontap/node-subtract-app  latest   104488c48925  8 days ago     165MB
dbaontap/node-add-app      latest   bc00e1413ed6  8 days ago     165MB
oraclelinux           7-slim   d94f4e9e5c13  9 days ago     118MB
ubuntu               16.04    13c9f1285025  2 weeks ago    119MB
tomcat                latest   5377fd8533c3  2 weeks ago    506MB
ubuntu               14.04    2c5e00d77a67  7 weeks ago
```

```
root@0953d43db486:/
File Edit View Search Terminal Help
root@Nx:/home/mucio/myimage# docker container run -it muciojaziel/myimage bash
root@0953d43db486:/#
```



A screenshot of a terminal window titled "root@0953d43db486: /". The window has two tabs: "root@0953d43db486:/" and "root@Nx:/home/mucio/Downloads/novo...". The content of the terminal shows:

```
root@Nx:/home/mucio# docker container run -it muciojaziel/myimage bash
root@0953d43db486:/# ping 172.22.75.40
PING 172.22.75.40 (172.22.75.40) 56(84) bytes of data.
64 bytes from 172.22.75.40: icmp_seq=1 ttl=64 time=0.213 ms
64 bytes from 172.22.75.40: icmp_seq=2 ttl=64 time=0.097 ms
64 bytes from 172.22.75.40: icmp_seq=3 ttl=64 time=0.109 ms
64 bytes from 172.22.75.40: icmp_seq=4 ttl=64 time=0.087 ms
64 bytes from 172.22.75.40: icmp_seq=5 ttl=64 time=0.107 ms
64 bytes from 172.22.75.40: icmp_seq=6 ttl=64 time=0.127 ms
64 bytes from 172.22.75.40: icmp_seq=7 ttl=64 time=0.095 ms
64 bytes from 172.22.75.40: icmp_seq=8 ttl=64 time=0.115 ms
64 bytes from 172.22.75.40: icmp_seq=9 ttl=64 time=0.095 ms
^C
--- 172.22.75.40 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8179ms
rtt min/avg/max/mdev = 0.087/0.116/0.213/0.036 ms
root@0953d43db486:/#
```

## 6.2 Construindo Cache

1. Para acelerar a criação de imagens, o Docker preserva um cache de etapas de compilação anteriores. Tente executar o mesmo comando de compilação que você fez anteriormente:

```
$ docker image build -t <username>/myimage .
```

Cada etapa deve executar imediatamente e reportar using cache indica que a etapa não foi repetida, mas foi obtida do cache de construção.

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage    x  root@Nx: /home/mucio/Downloads/novo...  x  +
root@Nx:/home/mucio/myimage# docker image build -t muciojziel/myimage .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/3 : RUN apt-get update
--> Using cache
--> 2c8400fa4737
Step 3/3 : RUN apt-get install -y iutils-ping
--> Using cache
--> 05b70822e55c
Successfully built 05b70822e55c
Successfully tagged muciojziel/myimage:latest
root@Nx:/home/mucio/myimage#
```

2. Abra seu Dockerfile e adicione outra etapa RUN no final para instalar o vim.

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage    x  root@Nx: /home/mucio/Downloads/novo...  x  +
root@Nx:/home/mucio/myimage# vim Dockerfile
root@Nx:/home/mucio/myimage#
```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage x root@Nx: /home/mucio/Downloads/novo... x
FROM ubuntu:16.04

RUN apt-get update
RUN apt-get install -y iputils-ping
RUN apt-get install -y vim ←
```

3. Construa a imagem novamente como acima; quais etapas o cache é usado?

```
root@Nx:/home/mucio/myimage# docker image build -t muciojziel/myimage .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/4 : RUN apt-get update
--> Using cache
--> 2c8400fa4737
Step 3/4 : RUN apt-get install -y iputils-ping
--> Using cache
--> 05b70822e55c
Step 4/4 : RUN apt-get install -y vim
--> Running in d348908e91eb
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  file libexpat1 libgpm2 libmagic1 libmpdec2 libpython3.5 libpython3.5-minimal
  libtinfo5 libutil-lsb3.0 libvte1.0-vim-compat vim-common
  vim-runtime vim-tiny
```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage  x  root@Nx: /home/mucio/Downloads/novo...  x  ↻
0 upgraded, 14 newly installed, 0 to remove and 5 not upgraded.
Need to get 12.3 MB of archives.
After this operation, 58.4 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu xenial/main amd64 libgpm2 amd64 1.20.4-6.
1 [16.5 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 libmagic1 amd64
1:5.25-2ubuntu1.2 [216 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 file amd64 1:5.
25-2ubuntu1.2 [21.2 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 libexpat1 amd64
2.1.0-7ubuntu0.16.04.4 [71.4 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial/main amd64 libmpdec2 amd64 2.4.2-1
[82.6 kB]
Get:6 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 libssl1.0.0 amd
64 1.0.2g-1ubuntu4.15 [1084 kB]
Get:7 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython3.5-mi
nimal amd64 3.5.2-2ubuntu0~16.04.5 [524 kB]
Get:8 http://archive.ubuntu.com/ubuntu xenial/main amd64 mime-support all 3.59ub
    auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/ex (ex) in aut
o mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/editor (editor
) in auto mode
Processing triggers for libc-bin (2.23-0ubuntu11) ...
Removing intermediate container d348908e91eb
    --> 1e45a400d9db
Successfully built 1e45a400d9db
Successfully tagged muciojazieli/myimage:latest
root@Nx:/home/mucio/myimage#
```

4. Construa a imagem novamente; quais etapas usam o cache desta vez?

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage  x  root@Nx: /home/mucio/Downloads/novo...  x
root@Nx:/home/mucio/myimage# docker image build -t muciojaziel/myimage .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/4 : RUN apt-get update
--> Using cache
--> 2c8400fa4737
Step 3/4 : RUN apt-get install -y iutils-ping
--> Using cache
--> 05b70822e55c
Step 4/4 : RUN apt-get install -y vim
--> Using cache
--> 1e45a400d9db
Successfully built 1e45a400d9db
Successfully tagged muciojaziel/myimage:latest
root@Nx:/home/mucio/myimage#
```

5. Finalmente, troque a ordem dos dois comandos RUN para instalar o ping e o vim no Dockerfile e construa uma última vez. Quais etapas são armazenadas em cache dessa vez?

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage  x  root@Nx: /home/mucio/Downloads/novo...  x
FROM ubuntu:16.04

RUN apt-get update
RUN apt-get install -y vim
RUN apt-get install -y iutils-ping
~
```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage x root@Nx: /home/mucio/Downloads/novo... x
root@Nx: /home/mucio/myimage# docker image build -t muciojaziel/myimage .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/4 : RUN apt-get update
--> Using cache
--> 2c8400fa4737
Step 3/4 : RUN apt-get install -y vim
--> Running in 01184bcd9633
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  file libexpat1 libgpm2 libmagic1 libmpdec2 libpython3.5 libpython3.5-minimal
  libpython3.5-stdlib libssqlite3-0 libssl1.0.0 mime-support vim-common
  vim-runtime
Suggested packages:
  vim-doc vim-scripts vim-zsh vim-gnome vim-gtk vim-atk vim-atk3 vim

```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage x root@Nx: /home/mucio/Downloads/novo... x
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/editor (editor )
) in auto mode
Processing triggers for libc-bin (2.23-0ubuntu11) ...
Removing intermediate container 01184bcd9633
--> 24b042451c45
Step 4/4 : RUN apt-get install -y iputils-ping
--> Running in 344c330184dd
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libffi6 libgmp10 libgnutls-openssl27 libgnutls30 libhogweed4 libidn11
  libnettle6 libp11-kit0 libtasn1-6
Suggested packages:
  gnutls-bin
The following NEW packages will be installed:
  iputils-ping libffi6 libgmp10 libgnutls-openssl27 libgnutls30 libhogweed4
  libidn11 libnettle6 libp11-kit0 libtasn1-6

```

### 6.3 O comando `history`

1. O comando `docker image history` nos permite inspecionar o histórico de cache de construção de uma imagem. Experimente com sua nova imagem:

```
$ docker image history <image id>
```

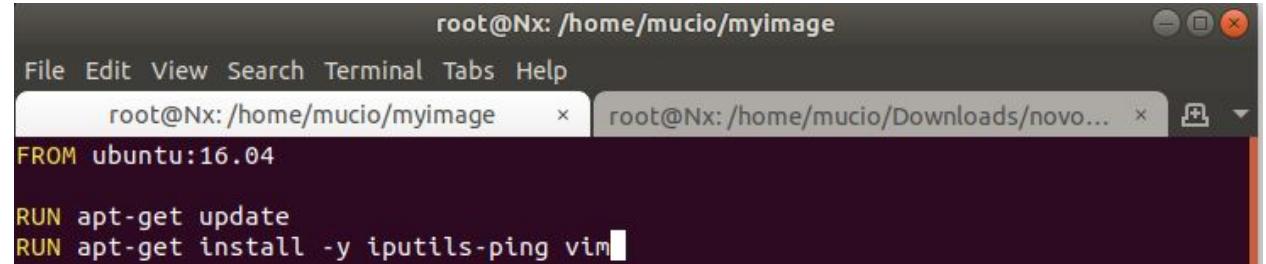
Observe o ID da camada criada para o comando apt-get update.

```
root@Nx:/home/mucio# docker image history muciojaziel/myimage
IMAGE           CREATED          CREATED BY
SIZE            COMMENT
86493f284b09   3 minutes ago   /bin/sh -c apt-get install -y iputils-ping
ng              4.27MB
24b042451c45   3 minutes ago   /bin/sh -c apt-get install -y vim
                57.5MB
2c8400fa4737   29 minutes ago  /bin/sh -c apt-get update
                25.2MB
13c9f1285025   2 weeks ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]
                0B
<missing>       2 weeks ago    /bin/sh -c mkdir -p /run/systemd && echo
'do...          7B
<missing>       2 weeks ago    /bin/sh -c set -xe  && echo '#!/bin/sh'
'> /...          745B
<missing>       2 weeks ago    /bin/sh -c rm -rf /var/lib/apt/lists/*
                0B
<missing>       2 weeks ago    /bin/sh -c #(nop) ADD file:24352f4a071cb
97b3...        119MB
root@Nx:/home/mucio#
```

2. Substitua os dois comandos RUN que instalaram iputils-ping e vim com um único comando:

...

```
RUN apt-get install -y iputils-ping vim
```



A screenshot of a terminal window titled "root@Nx: /home/mucio/myimage". The window has a dark background and light-colored text. It shows the beginning of a Dockerfile with the line "FROM ubuntu:16.04" followed by two RUN commands: "RUN apt-get update" and "RUN apt-get install -y iputils-ping vim". The cursor is visible at the end of the second RUN command.

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage  x  root@Nx: /home/mucio/Downloads/novo...  x
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get install -y iputils-ping vim
```

3. Crie a imagem novamente e execute o docker image history nesta nova imagem. Como a history mudou?

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage      x root@Nx: /home/mucio/Downloads/novo... x [+]
<missing>          2 weeks ago      /bin/sh -c #(nop) ADD file:24352f4a071cb
97b3... 119MB
root@Nx:/home/mucio# clear

root@Nx:/home/mucio# cd myimage
root@Nx:/home/mucio/myimage# vim Dockerfile
root@Nx:/home/mucio/myimage# docker image build -t muciojaziel/myimage .
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/3 : RUN apt-get update
--> Using cache
--> 2c8400fa4737
Step 3/3 : RUN apt-get install -y iputils-ping vim
--> Running in 7003a37d6a02
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  file libexpat1 libffi6 libgnutls10 libgnutls-openssl27 libgnutls30 libgnutls27

```

```
root@Nx:/home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx:/home/mucio/myimage  x  root@Nx:/home/mucio/Downloads/novo...  x
Setting up vim-common (2:7.4.1689-3ubuntu1.3) ...
Setting up libpython3.5:amd64 (3.5.2-2ubuntu0~16.04.5) ...
Setting up vim-runtime (2:7.4.1689-3ubuntu1.3) ...
Setting up vim (2:7.4.1689-3ubuntu1.3) ...
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vim (vim) in a
uto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vimdiff (vimdi
ff) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rvim (rvim) in
auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rview (rview)
in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vi (vi) in aut
o mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/view (view) in
auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/ex (ex) in aut
o mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/editor (editor
) in auto mode
Processing triggers for libc-bin (2.23-0ubuntu11) ...
Removing intermediate container 7003a37d6a02
--> 79e1855d745d
Successfully built 79e1855d745d
Successfully tagged muciojaziel/myimage:latest
root@Nx:/home/mucio/myimage#
```

## 6.4 Conclusão

Até agora, vimos como escrever um Dockerfile básico usando os comandos FROM e RUN, algumas noções básicas de como o caching de imagem funciona e o comando docker image history. Depois de alguma discussão, veremos como definir alguns comandos e opções padrão para execução como o PID 1 de contêineres criados a partir de uma imagem definida pelo nosso Dockerfile.

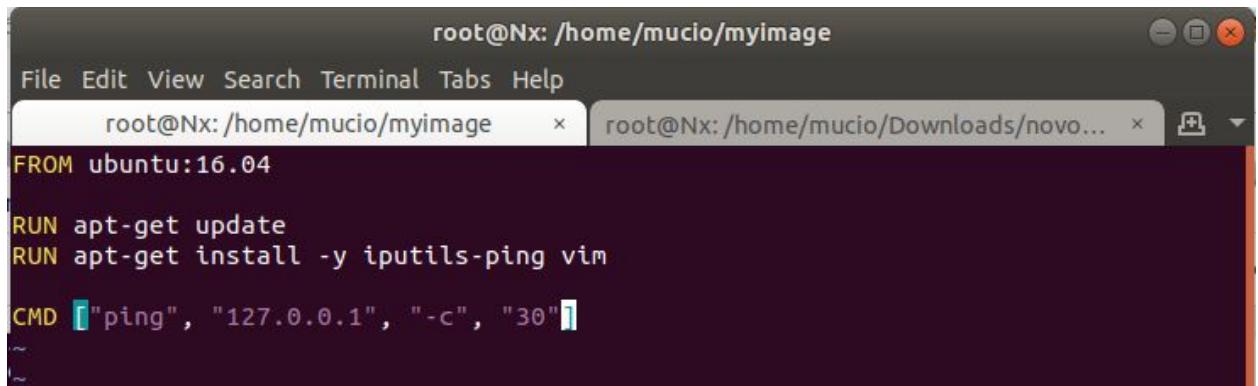
## 7 Criando Imagens com Dockerfiles (2/2)

### 7.1 Comandos Padrão via CMD

1. Adicione a seguinte linha ao seu Dockerfile do último problema, na parte inferior

```
...
CMD ["ping", "127.0.0.1", "-c", "30"]
```

Isso define o ping como o comando padrão para ser executado em um contêiner criado a partir dessa imagem e também define alguns parâmetros para esse comando.

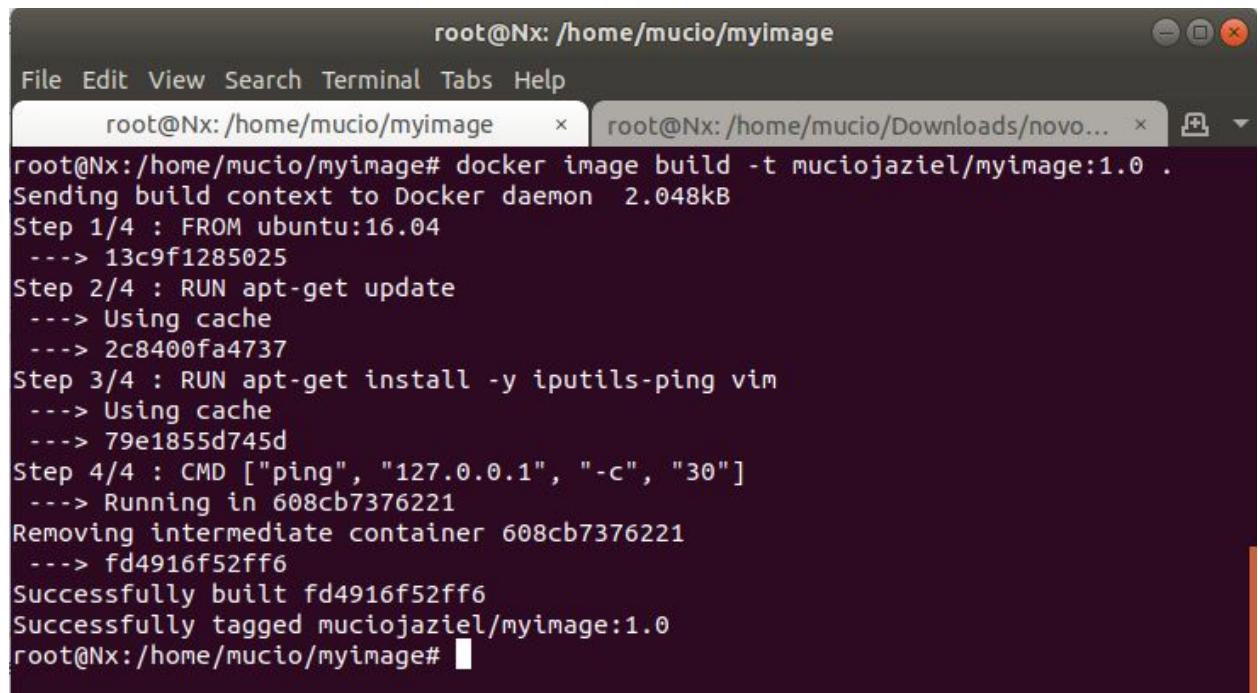


A screenshot of a terminal window titled "root@Nx: /home/mucio/myimage". The window has two tabs: "root@Nx: /home/mucio/myimage" and "root@Nx: /home/mucio/Downloads/novo...". The content of the terminal shows a Dockerfile:

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get install -y iputils-ping vim
CMD ["ping", "127.0.0.1", "-c", "30"]
```

2. Reconstruindo (Rebuild) sua imagem:

```
$ docker image build -t <username>/myimage:1.0 .
```



A screenshot of a terminal window titled "root@Nx: /home/mucio/myimage". The window has two tabs: "root@Nx: /home/mucio/myimage" and "root@Nx: /home/mucio/Downloads/novo...". The content of the terminal shows the Docker build process:

```
root@Nx:/home/mucio/myimage# docker image build -t muciojaziel/myimage:1.0 .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/4 : RUN apt-get update
--> Using cache
--> 2c8400fa4737
Step 3/4 : RUN apt-get install -y iputils-ping vim
--> Using cache
--> 79e1855d745d
Step 4/4 : CMD ["ping", "127.0.0.1", "-c", "30"]
--> Running in 608cb7376221
Removing intermediate container 608cb7376221
--> fd4916f52ff6
Successfully built fd4916f52ff6
Successfully tagged muciojaziel/myimage:1.0
root@Nx:/home/mucio/myimage#
```

3. Execute um contêiner da sua nova imagem sem nenhum comando fornecido:

```
$ docker container run <username>/myimage:1.0
```

```
root@Nx:/home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx:/home/mucio/myimage  x  root@Nx:/home/mucio/Downloads/novo...  x
root@Nx:/home/mucio/myimage# docker container run muciojaziel/myimage:1.0
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.036 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.058 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.076 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.049 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.073 ms
64 bytes from 127.0.0.1: icmp_seq=11 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=0.029 ms
```

4. Você deve ver o comando fornecido pelo parâmetro CMD no Dockerfile em execução. Tente fornecer explicitamente um comando ao executar um contêiner:

```
$ docker container run <username>/myimage:1.0 echo "hello world"
```

```
root@Nx:/home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx:/home/mucio/myimage  x  root@Nx:/home/mucio/Downloads/novo...  x
root@Nx:/home/mucio/myimage# docker container run muciojaziel/myimage:1.0 echo "hello world"
hello world
root@Nx:/home/mucio/myimage#
```

Fornecer um comando em `docker container run` substitui o comando definido pelo CMD.

## 7.2 Comandos padrões via ENTRYPOINT

1. Substitua a instrução **CMD** no seu Dockerfile por um **ENTRYPOINT**: ...

```
ENTRYPOINT ["ping"]
```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage  x  root@Nx: /home/mucio/Downloads/novo...  x
FROM ubuntu:16.04

RUN apt-get update
RUN apt-get install -y iputils-ping vim

ENTRYPOINT ["ping"]
~
```

2. Crie a imagem e use-a para executar um contêiner sem argumentos de processo:

```
$ docker image build -t <username>/myimage:1.0 .
```

```
root@Nx:/home/mucio/myimage# docker image build -t muciojaziel/myimage:1.0 .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM ubuntu:16.04
--> 13c9f1285025
Step 2/4 : RUN apt-get update
--> Using cache
--> 2c8400fa4737
Step 3/4 : RUN apt-get install -y iputils-ping vim
--> Using cache
--> 79e1855d745d
Step 4/4 : ENTRYPOINT ["ping"]
--> Running in bb37b19f6f08
Removing intermediate container bb37b19f6f08
--> bd1f4ef5ebe0
Successfully built bd1f4ef5ebe0
Successfully tagged muciojaziel/myimage:1.0
root@Nx:/home/mucio/myimage#
```

```
$ docker container run <username>/myimage:1.0
```

O que deu errado?

```
root@Nx:/home/mucio/myimage# docker container run muciojaziel/myimage:1.0
Usage: ping [-aAbBdFhLnOqrUvV] [-c count] [-i interval] [-I interface]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
root@Nx:/home/mucio/myimage#
```

3. Tente rodar com um argumento após o nome da imagem:

```
$ docker container run <username>/myimage:1.0 127.0.0.1
```

Os tokens fornecidos após um nome de imagem são enviados como argumentos para o comando especificado por ENTRYPOINT.

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx:/home/mucio/myimage  x  root@Nx:/home/mucio/Downloads/novo...  x  [+]
root@Nx:/home/mucio/myimage# docker container run muciojaziel/myimage:1.0 127.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.165 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.072 ms
```

### 7.3 CMD e ENTRYPOINT juntos

1. Abra seu Dockerfile e modifique a instrução ENTRYPOINT para incluir 2 argumentos para o comando ping:

Obs: Nesse ponto você já deve saber editar arquivos pelo terminal.

```
...
ENTRYPOINT ["ping", "-c", "3"]
```

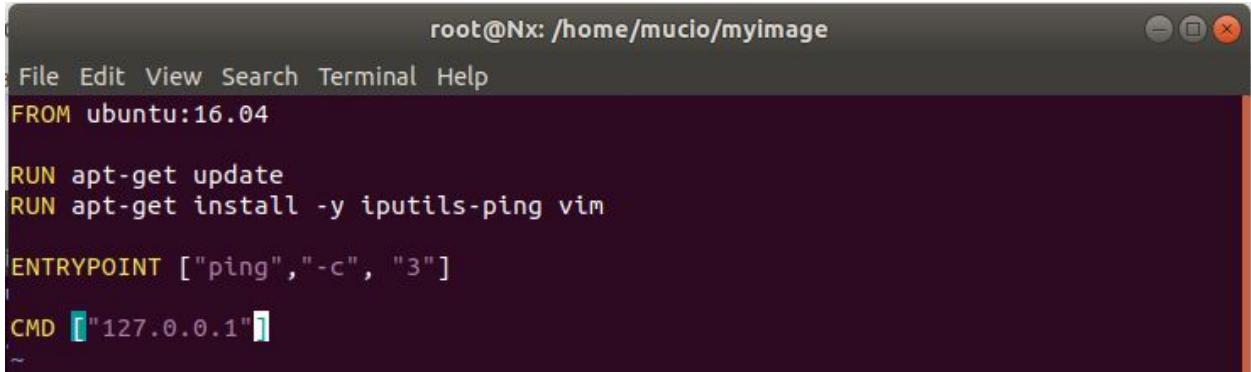
```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Help
FROM ubuntu:16.04

RUN apt-get update
RUN apt-get install -y iputils-ping vim

ENTRYPOINT ["ping","-c", "3"]
```

2. Se CMD e ENTRYPOINT ambos são especificados em um Dockerfile, os tokens listados no CMD são usados como parâmetros padrão para o comando ENTRYPOINT. Adicione um CMD com um IP padrão para ping:

```
...
CMD ["127.0.0.1"]
```



```

root@Nx: /home/mucio/myimage
File Edit View Search Terminal Help
FROM ubuntu:16.04

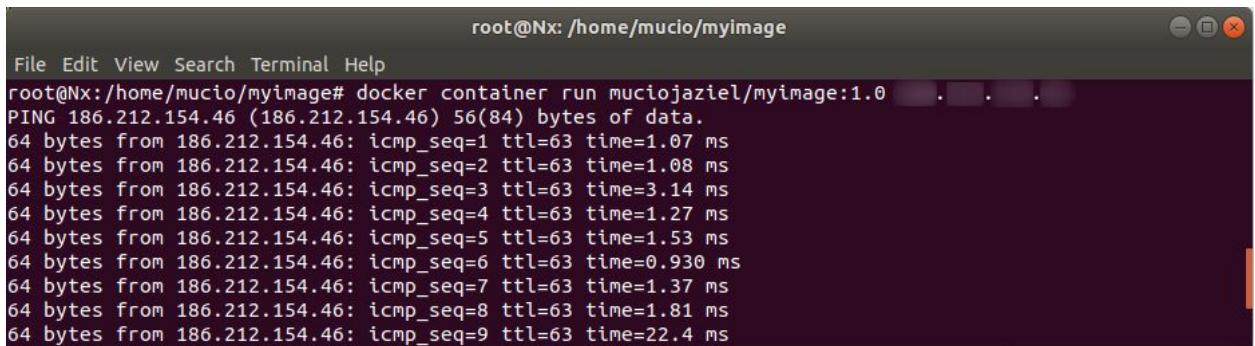
RUN apt-get update
RUN apt-get install -y iputils-ping vim

ENTRYPOINT ["ping","-c", "3"]

CMD ["127.0.0.1"]
~
```

3. Construa a imagem e execute um contêiner com um IP público como um argumento para docker container run:

```
$ docker container run <username>/myimage:1.0 <some public ip>
```

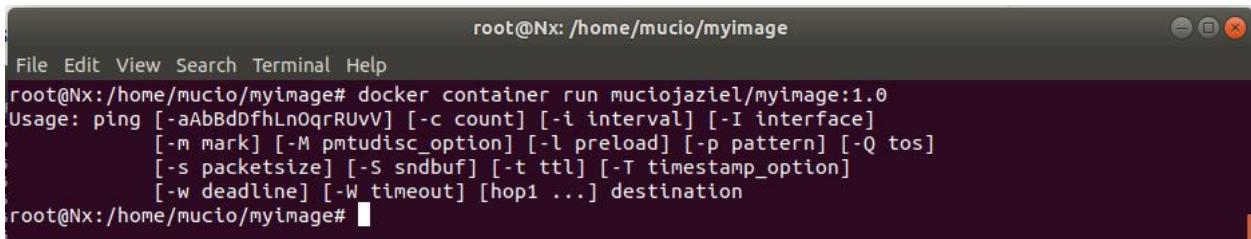


```

root@Nx: /home/mucio/myimage
File Edit View Search Terminal Help
root@Nx:/home/mucio/myimage# docker container run muciojziel/myimage:1.0 . . .
PING 186.212.154.46 (186.212.154.46) 56(84) bytes of data.
64 bytes from 186.212.154.46: icmp_seq=1 ttl=63 time=1.07 ms
64 bytes from 186.212.154.46: icmp_seq=2 ttl=63 time=1.08 ms
64 bytes from 186.212.154.46: icmp_seq=3 ttl=63 time=3.14 ms
64 bytes from 186.212.154.46: icmp_seq=4 ttl=63 time=1.27 ms
64 bytes from 186.212.154.46: icmp_seq=5 ttl=63 time=1.53 ms
64 bytes from 186.212.154.46: icmp_seq=6 ttl=63 time=0.930 ms
64 bytes from 186.212.154.46: icmp_seq=7 ttl=63 time=1.37 ms
64 bytes from 186.212.154.46: icmp_seq=8 ttl=63 time=1.81 ms
64 bytes from 186.212.154.46: icmp_seq=9 ttl=63 time=22.4 ms
```

4. Execute outro contêiner sem nenhum argumento após o nome da imagem. Explique a diferença de comportamento entre esses dois últimos contêineres.

Obs:



```

root@Nx: /home/mucio/myimage
File Edit View Search Terminal Help
root@Nx:/home/mucio/myimage# docker container run muciojziel/myimage:1.0
Usage: ping [-aAbBdDfhLn0qrRUVv] [-c count] [-i interval] [-I interface]
           [-m mark] [-M pmtdisc_option] [-l preload] [-p pattern] [-Q tos]
           [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
           [-w deadline] [-W timeout] [hop1 ...] destination
root@Nx:/home/mucio/myimage#
```

## 7.4 Conclusão

Neste exercício, encontramos os comandos CMD e ENTRYPOINT do Dockerfile. Eles são úteis para definir o processo padrão para ser executado como PID 1 dentro do contêiner no Dockerfile, tornando nossos contêineres mais semelhantes como também executáveis e adicionando clareza a exatamente qual processo deve ser executado nos contêineres de uma determinada imagem.

## 8 Dockerizando uma Aplicação

### 8.1 Configurando um aplicativo Java

*Uma maneira alternativa de fazer isso, veja abaixo (Exercício Opcional)*

1. instale o Java 8:

```
$ sudo apt-get install openjdk-8-jdk
```

```
root@Nx: /home/mucio/myimage
File Edit View Search Terminal Tabs Help
root@Nx: /home/mucio/myimage x root@Nx: /home/mucio/myimage x
root@Nx:/home/mucio/myimage# apt-get install openjdk-8-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libjavascriptcoregtk-1.0-0 libmypaint libwebkitgtk-1.0-0 mypaint-brushes
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  ca-certificates-java fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni
  libice-dev libpthread-stubs0-dev libsm-dev libx11-dev libx11-doc libxau-dev libxcb1-dev
  libxdmcp-dev libxt-dev openjdk-8-jdk-headless openjdk-8-jre openjdk-8-jre-headless
  x11proto-core-dev x11proto-dev xorg-sgml-doctools xtrans-dev
```

```
root@Nx: /home/mucio/javahelloworld
File Edit View Search Terminal Help
droolsjbpm-tools-distribution-6.5.0.Final  minikube      Public      'VirtualBox VMs'
Dropbox                                     Music        snap
eclipse-workspace                           myimage     Templates
root@Nx:/home/mucio# cd javahelloworld
root@Nx:/home/mucio/javahelloworld# touch HelloWorld.java
root@Nx:/home/mucio/javahelloworld# vim HelloWorld.java
```

2. Crie um diretório chamado `javahelloworld`; nesse diretório, crie um arquivo chamado `HelloWorld.java`, contendo o seguinte code [os números ao lado não precisa]:

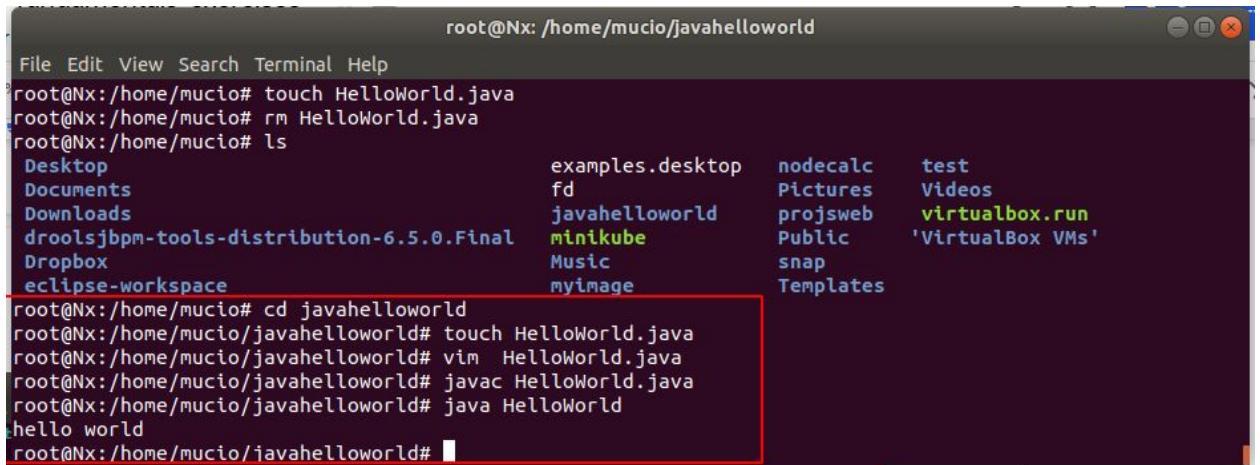
```
public class HelloWorld
{
    public static void main (String [] args)
    {
        System.out.println("hello world");
    }
}
```



```
root@Nx: /home/mucio/javahelloworld
File Edit View Search Terminal Help
public class HelloWorld
{
    public static void main (String [] args)
    {
        System.out.println("hello world");
    }
}
~
```

3. Compile e execute sua nova aplicação:

```
$ javac HelloWorld.java
$ java HelloWorld
```



```
root@Nx: /home/mucio/javahelloworld
File Edit View Search Terminal Help
root@Nx:/home/mucio# touch HelloWorld.java
root@Nx:/home/mucio# rm HelloWorld.java
root@Nx:/home/mucio# ls
Desktop examples.desktop nodecalc test
Documents fd Pictures Videos
Downloads javahelloworld projswb virtualbox.run
droolsjbpm-tools-distribution-6.5.0.Final minikube Public 'VirtualBox VMs'
Dropbox Music snap
eclipse-workspace myimage Templates
root@Nx:/home/mucio# cd javahelloworld
root@Nx:/home/mucio/javahelloworld# touch HelloWorld.java
root@Nx:/home/mucio/javahelloworld# vim HelloWorld.java
root@Nx:/home/mucio/javahelloworld# javac HelloWorld.java
root@Nx:/home/mucio/javahelloworld# java HelloWorld
hello world
root@Nx:/home/mucio/javahelloworld#
```

Se tudo correr bem, você tem um programa “hello world” rodando em Java; Em seguida, gostaríamos de colocar esse aplicativo em container, capturando seu ambiente em uma imagem descrita por um Dockerfile.

## 8.2 Dockerize seu App

1. Em sua pasta `javahelloworld`, crie um Dockerfile que baseia sua imagem fora da imagem base `java: 8`:

adicone essa linha no Dockerfile

```
FROM java: 8
```

2. Adicione seu código-fonte à sua imagem usando um novo comando Dockerfile, `COPY`:

...

```
COPY HelloWorld.java /
```

A sintaxe do COPY, é COPY <target> <destination>, que copia arquivos do contexto de construção para a imagem. Se <target> for um diretório, todo o conteúdo desse diretório será copiado para <destination>.

3. Compile seu aplicativo na imagem anexando ao seu Dockerfile:

```
...
RUN javac HelloWorld.java
```

4. Use ENTRYPOINT para executar seu aplicativo automaticamente quando um contêiner for iniciado a partir desta imagem:

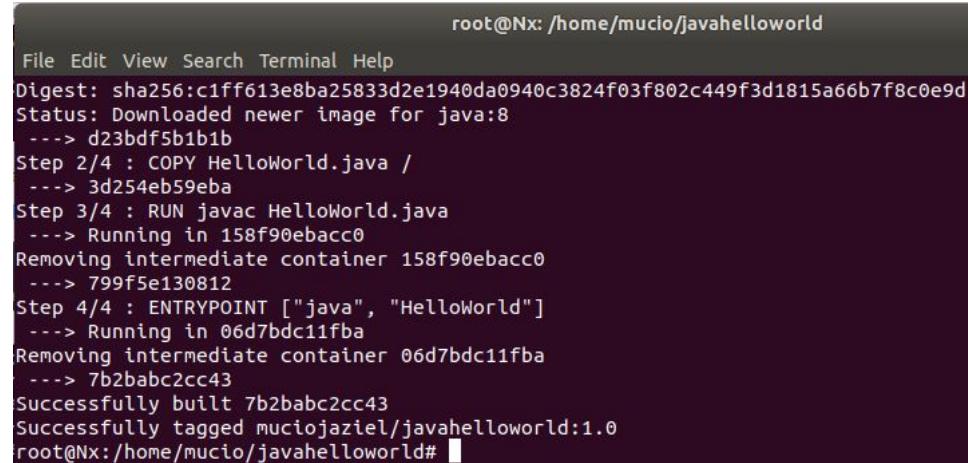
```
...
ENTRYPOINT ["java", "HelloWorld"]
```

Obs: Todos comandos estão no Dockerfile



```
root@Nx:/home/mucio/javahelloworld
File Edit View Search Terminal Help
FROM java:8
COPY HelloWorld.java /
RUN javac HelloWorld.java
ENTRYPOINT ["java", "HelloWorld"]
```

5. Crie a imagem, use-a para executar um contêiner executando o comando padrão ENTRYPOINT e observe a saída.



```
root@Nx:/home/mucio/javahelloworld
File Edit View Search Terminal Help
Digest: sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7f8c0e9d
Status: Downloaded newer image for java:8
--> d23bdf5b1b1b
Step 2/4 : COPY HelloWorld.java /
--> 3d254eb59eba
Step 3/4 : RUN javac HelloWorld.java
--> Running in 158f90ebacc0
Removing intermediate container 158f90ebacc0
--> 799f5e130812
Step 4/4 : ENTRYPOINT ["java", "HelloWorld"]
--> Running in 06d7bdc11fba
Removing intermediate container 06d7bdc11fba
--> 7b2babc2cc43
Successfully built 7b2babc2cc43
Successfully tagged muciojaziel/javahelloworld:1.0
root@Nx:/home/mucio/javahelloworld#
```

Executando a imagem:

```
root@Nx:/home/mucio/javahelloworld# docker container run muciojaziel/javahelloworld:1.0
hello world
root@Nx:/home/mucio/javahelloworld#
```

### 8.3 Reestruture sua Aplicação

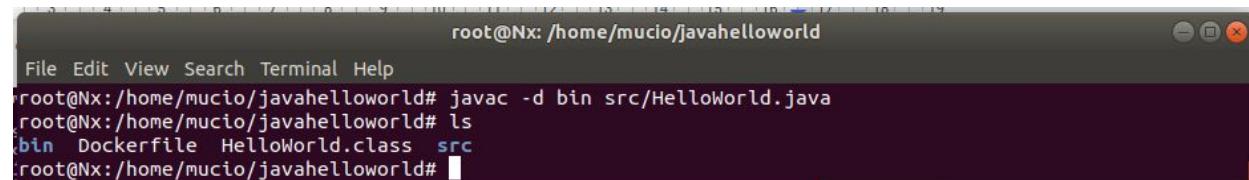
- Depois de executar um contêiner da sua imagem com o comando padrão, tente substituir o comando ENTRYPPOINT por --entrypoint; Vamos executar o bash, para que possamos explorar nosso ambiente conteinerizado de forma interativa com um shell bash:

```
$ docker container run -it --entrypoint bash <your java image ID>
```

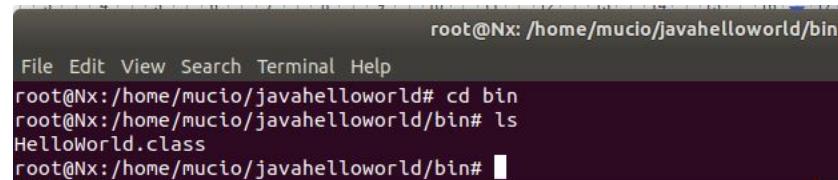
```
root@Nx:/home/mucio/javahelloworld# docker container run -it --entrypoint bash muciojaziel/javahelloworld:1.0
root@129f6496ae91:/# ls
HelloWorld.class  bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
HelloWorld.java   boot  etc  lib  media  opt  root  sbin  sys  usr
root@129f6496ae91:/#
```

- Encontre onde está o código fonte HelloWorld.java e onde está o binário compilado. Podemos reestruturar nosso aplicativo para ser um pouco mais organizado da seguinte forma; de volta em sua máquina host no diretório javahelloworld, crie dois subdiretórios src e bin. Coloque seu código fonte java neste diretório src e recompile para que o binário acabe no diretório bin:

```
$ javac -d bin src/HelloWorld.java
```



```
root@Nx:/home/mucio/javahelloworld
File Edit View Search Terminal Help
root@Nx:/home/mucio/javahelloworld# javac -d bin src/HelloWorld.java
root@Nx:/home/mucio/javahelloworld# ls
bin  Dockerfile  HelloWorld.class  src
root@Nx:/home/mucio/javahelloworld#
```



```
root@Nx:/home/mucio/javahelloworld/bin
File Edit View Search Terminal Help
root@Nx:/home/mucio/javahelloworld# cd bin
root@Nx:/home/mucio/javahelloworld/bin# ls
HelloWorld.class
root@Nx:/home/mucio/javahelloworld/bin#
```

- Execute o aplicativo novamente com java -cp bin HelloWorld para garantir que tudo ainda esteja funcionando; Se estiver, estamos prontos para modificar nosso Dockerfile para refletir essa estrutura organizacional em nossa imagem.

```
root@Nx:/home/mucio/javahelloworld/bin# cd ..
root@Nx:/home/mucio/javahelloworld# java -cp bin HelloWorld
hello world
root@Nx:/home/mucio/javahelloworld#
```

- Modifique a instrução COPY para copiar todos os arquivos da pasta src do seu host para sua imagem: /home/root/javahelloworld/src

```
...  
COPY src /home/root/javahelloworld/src
```

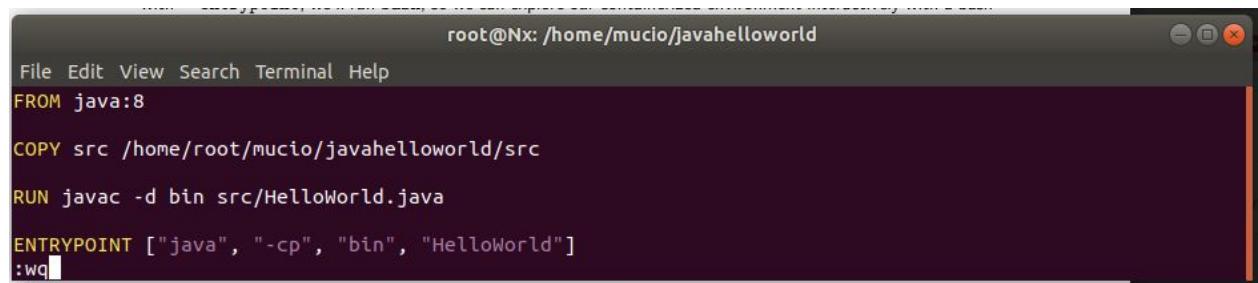
5. Modifique a instrução RUN para compilar o código referenciando a pasta `src` correta e para colocar o compilado código na pasta `bin`:

```
...  
RUN javac -d bin src/HelloWorld.java
```

6. Modifique ENTRYPOINT para especificar `java -cp bin`:

```
...  
ENTRYPOINT ["java", "-cp", "bin", "HelloWorld"]
```

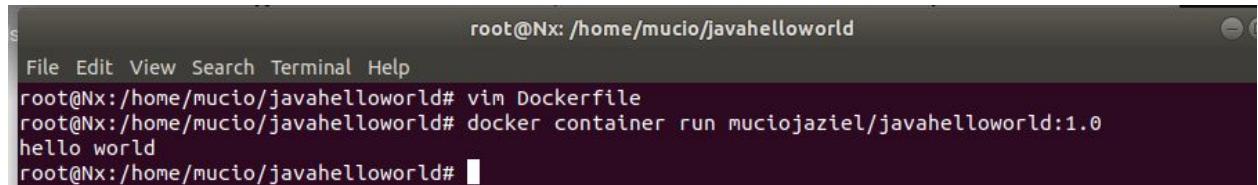
Obs: Os passos 4,5,6 estão contidos nessa imagem.



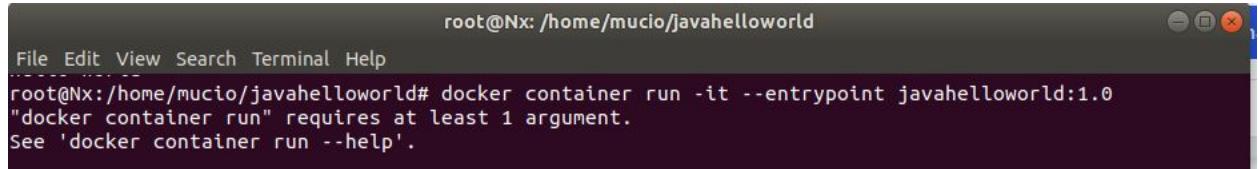
```
root@Nx: /home/mucio/javahelloworld  
File Edit View Search Terminal Help  
FROM java:8  
  
COPY src /home/root/mucio/javahelloworld/src  
  
RUN javac -d bin src/HelloWorld.java  
  
ENTRYPOINT ["java", "-cp", "bin", "HelloWorld"]  
:wq
```

7. Tente construir uma execução sua imagem agora; você deve obter um erro desde que seu código-fonte está agora em `/home/root/javahelloworld/src`, e o comando RUN disse ao compilador para procurar em `./src`. Por padrão, os comandos são executados na raiz do sistema de arquivos da imagem, mas podemos modificá-lo com a instrução WORKDIR Dockerfile, que define a posição no sistema de arquivos para todos os comandos subsequentes.

Obs: Neste treinamento Funcionou, caso não funcione, você usa o WORKDIR.



```
root@Nx: /home/mucio/javahelloworld  
File Edit View Search Terminal Help  
root@Nx:/home/mucio/javahelloworld# vim Dockerfile  
root@Nx:/home/mucio/javahelloworld# docker container run muciojaziel/javahelloworld:1.0  
hello world  
root@Nx:/home/mucio/javahelloworld#
```



```
root@Nx: /home/mucio/javahelloworld  
File Edit View Search Terminal Help  
root@Nx:/home/mucio/javahelloworld# docker container run -it --entrypoint javahelloworld:1.0  
"docker container run" requires at least 1 argument.  
See 'docker container run --help'.
```

8. Antes da instrução RUN, adicione:

```

...
WORKDIR /home/root/javahelloworld

root@Nx:/home/mucio/javahelloworld
File Edit View Search Terminal Help
COPY src /home/root/mucio/javahelloworld/src
WORKDIR /home/root/javahelloworld
RUN javac -d bin src/HelloWorld.java
ENTRYPOINT ["java", "-cp", "bin", "HelloWorld"]
:wq

```

### Desafio, tente executar os passos 09 e 10!

9. Tente construir sua imagem e executar o contêiner com o comando padrão ENTRYPOINT novamente. Ainda há um erro, mas um diferente; modifique seu Dockerfile para corrigir isso, reconstrua sua imagem e verifique se tudo está funcionando corretamente novamente.
10. Finalmente, substitua o comando padrão ENTRYPOINT e execute o bash, como fizemos acima. Verifique se a estrutura de diretórios que você descreveu no Dockerfile está refletida no sistema de arquivos da sua imagem.

## 8.4 Conclusão

Neste exercício, você fez o docker de um aplicativo simples e aprendeu a manipular e navegar pelo sistema de arquivos de uma imagem com os comandos COPY e WORKDIR do Dockerfile. Com alguma prática, escrever um Dockerfile que constrói uma imagem de aplicativo segue um padrão muito semelhante à configuração nativa do aplicativo, expressando coisas com os comandos RUN, COPY, WORKDIR e ENTRYPOINT.

## 8.5 Exercício Opcional

**Tente esse exercício. Alguns conhecimentos adquiridos dão suporte a execução dessa tarefa.**

Para uma maneira alternativa de criar o aplicativo Java sem ter que instalar o Java no host, você pode usar um contêiner Java.

1. Execute um contêiner Java no modo interativo:

```
$ docker container run --rm -it java:8 bash
```

2. Instalar um editor dentro do contêiner, VIM ou nano

```
$ apt-get update && apt-get install -y vim
```

```
# ou install nano # apt-get update && apt-get install -y nano
```

3. Agora, ainda dentro do contêiner, crie um diretório `javahelloworld` e crie o arquivo `HelloWorld.java`

nele usando o editor que você acabou de instalar, por exemplo, com o vim que seria:

```
$ mkdir javahelloworld $ cd javahelloworld $ vim HelloWorld.java
```

Adicione o código a este arquivo, salve e saia do editor

4. Ainda dentro do contêiner Java, compile e execute o aplicativo

```
$ javac HelloWorld.java $ java HelloWorld
```

## 9 Gerenciando Imagens

### 9.1 Marcando [TAG - Tagueando] e Listando Imagens

1. Baixe a imagem ubuntu:16.04 da Docker Store:

```
$ docker image pull ubuntu:16.04
```

```
root@Nx:/home/mucio/novaimg# docker image pull ubuntu:16.04
16.04: Pulling from library/ubuntu
Digest: sha256:a4d8e674ee993e5ec88823391de828a5e9286a1597b731eaecaaf9066cfdf539
Status: Image is up to date for ubuntu:16.04
root@Nx:/home/mucio/novaimg#
```

2. Faça uma nova tag desta imagem::

```
$ docker image tag ubuntu:16.04 my-ubuntu:dev
```

```
root@Nx:/home/mucio/novaimg# docker image tag ubuntu:16.04 my-ubuntu:dev
root@Nx:/home/mucio/novaimg#
```

3. Liste suas imagens:

```
$ docker image ls
```

```
root@Nx:/home/mucio/novaimg# docker image ls
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
mociojaziel/javahelloworld  1.0        7b2babcc2cc43  3 days ago   643MB
mociojaziel/myimage    1.2        ad484e5655b7   4 days ago   206MB
mociojaziel/myimage    1.3        ad484e5655b7   4 days ago   206MB
mociojaziel/myimage    1.0        bd1f4ef5eb0   7 days ago   206MB
<none>               <none>     fd4916f52ff6  7 days ago   206MB
mociojaziel/myimage    latest     79e1855d745d  7 days ago   206MB
<none>               <none>     86493f284b09  7 days ago   206MB
<none>               <none>     1e45a400d9db  7 days ago   206MB
mociojaziel/myapp      1.1        6411ab568f05  9 days ago   205MB
mociojaziel/myapp      1.0        885fc6e2fbcf  9 days ago   205MB
dbaontap/node-divide-app latest     a6301128b6ea  2 weeks ago  165MB
dbaontap/node-multiply-app latest    56b756f6d340  2 weeks ago  165MB
dbaontap/node-subtract-app latest    104488c48925  2 weeks ago  165MB
dbaontap/node-add-app   latest     bc00e1413ed6  2 weeks ago  165MB
oraclelinux            7-slim     d94f4e9e5c13  2 weeks ago  118MB
my-ubuntu              dev        13c9f1285025  3 weeks ago  119MB
ubuntu                 16.04     13c9f1285025  3 weeks ago  119MB
tomcat                 latest     5377fd853c3  3 weeks ago  506MB
ubuntu                 14.04     2c5e00d77a67  8 weeks ago  188MB
dbaontap/node-add-app <none>     dc126322ee0c  21 months ago 202MB
java                   8          d23bdf5b1b1b  2 years ago  643MB
root@Nx:/home/mucio/novaimg#
```

4. Você deve ter o `ubuntu:16.04` e `my-ubuntu:dev` listados, mas eles devem ter o mesmo hash no ID da imagem, já que eles são na verdade a mesma imagem.

```
root@Nx:/home/mucio/novaimg# docker image ls
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
mociojaziel/javahelloworld  1.0        7b2babcc2cc43  3 days ago   643MB
mociojaziel/myimage    1.2        ad484e5655b7   4 days ago   206MB
mociojaziel/myimage    1.3        ad484e5655b7   4 days ago   206MB
mociojaziel/myimage    1.0        bd1f4ef5eb0   7 days ago   206MB
<none>               <none>     fd4916f52ff6  7 days ago   206MB
mociojaziel/myimage    latest     79e1855d745d  7 days ago   206MB
<none>               <none>     86493f284b09  7 days ago   206MB
<none>               <none>     1e45a400d9db  7 days ago   206MB
mociojaziel/myapp      1.1        6411ab568f05  9 days ago   205MB
mociojaziel/myapp      1.0        885fc6e2fbcf  9 days ago   205MB
dbaontap/node-divide-app latest     a6301128b6ea  2 weeks ago  165MB
dbaontap/node-multiply-app latest    56b756f6d340  2 weeks ago  165MB
dbaontap/node-subtract-app latest    104488c48925  2 weeks ago  165MB
dbaontap/node-add-app   latest     bc00e1413ed6  2 weeks ago  165MB
oraclelinux            7-slim     d94f4e9e5c13  2 weeks ago  118MB
my-ubuntu              dev        13c9f1285025  3 weeks ago  119MB
ubuntu                 16.04     13c9f1285025  3 weeks ago  119MB
tomcat                 latest     5377fd853c3  3 weeks ago  506MB
ubuntu                 14.04     2c5e00d77a67  8 weeks ago  188MB
dbaontap/node-add-app <none>     dc126322ee0c  21 months ago 202MB
java                   8          d23bdf5b1b1b  2 years ago  643MB
root@Nx:/home/mucio/novaimg#
```

## 9.2 Compartilhando Imagens na Docker Store

1. Em seguida, vamos compartilhar nossa imagem na Docker Store. Se você ainda não tem uma conta, acesse [store.docker.com](https://store.docker.com) e faça uma.

2. Faça um Envio [Push] da sua imagem to Docker Store:

```
$ docker image push my-ubuntu:dev
```

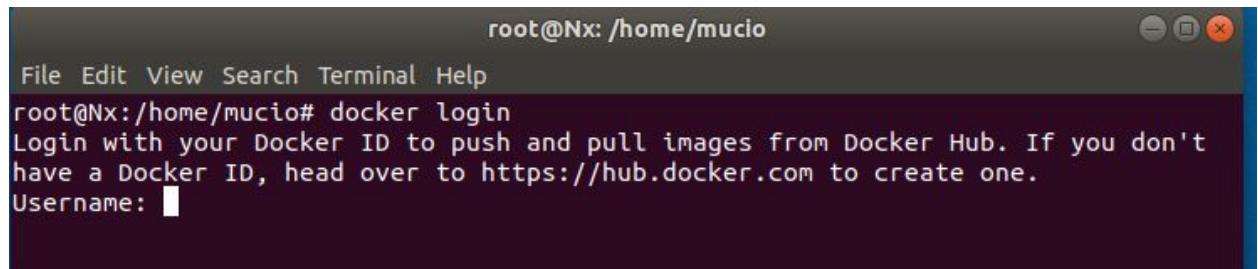
Você deve receber um erro de autenticação obrigatório, algo assim: `error authentication required`.

```
root@Nx:/home/mucio/novaimg# docker image push my-ubuntu:dev
The push refers to repository [docker.io/library/my-ubuntu]
92d3f22d44f3: Preparing
10e46f329a25: Preparing
24ab7de5faec: Preparing
1ea5a27b0484: Preparing
denied: requested access to the resource is denied
root@Nx:/home/mucio/novaimg#
```

3. Faça login no docker login, e tente enviar novamente. O envio [Push] falha novamente porque não temos o namespace de nossa imagem corretamente para distribuição na Docker Store; todas as imagens que você deseja compartilhar no Docker Store devem ser nomeadas como <your Docker Store username>/<repo name>[:<optional tag>].

4. Renomeie a tag da sua imagem para ser namespaced corretamente e envie novamente:

Obs: É preciso criar uma conta no Docker Hub. E fazer o login no terminal como abaixo:



```
root@Nx:/home/mucio/novaimg# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: muciojaziel
Password:
Error response from daemon: Get https://registry-1.docker.io/v2/: unauthorized:
incorrect username or password
root@Nx:/home/mucio/novaimg# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: muciojaziel
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
root@Nx:/home/mucio/novaimg#
```

Insira o nome do usuário, e após sua senha [ a mesma usada no site]. Em caso de erro, você pode tentar:

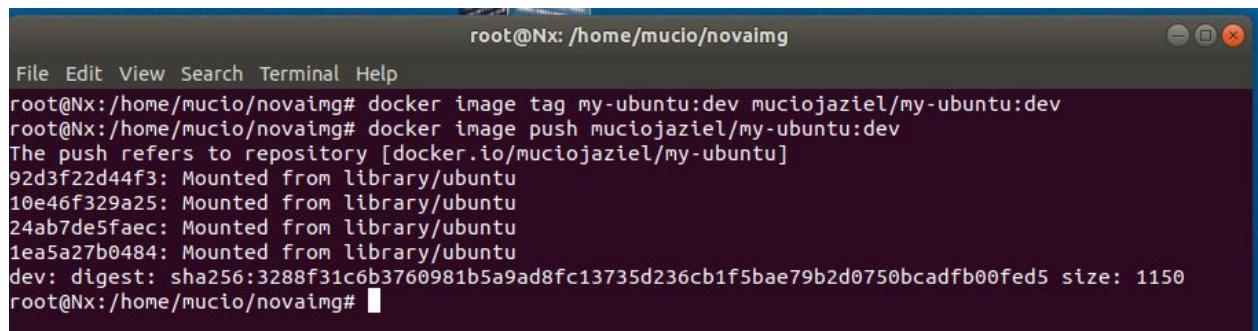
```
$ sudo apt install gnupg2 pass
```

Caso, o erro persista, você pode analisar esta discussão, e tentar o passo a passo por ela que irá conseguir.

<https://stackoverflow.com/questions/50151833/cannot-login-to-docker-account>

**Após isso, use os comandos seguintes:**

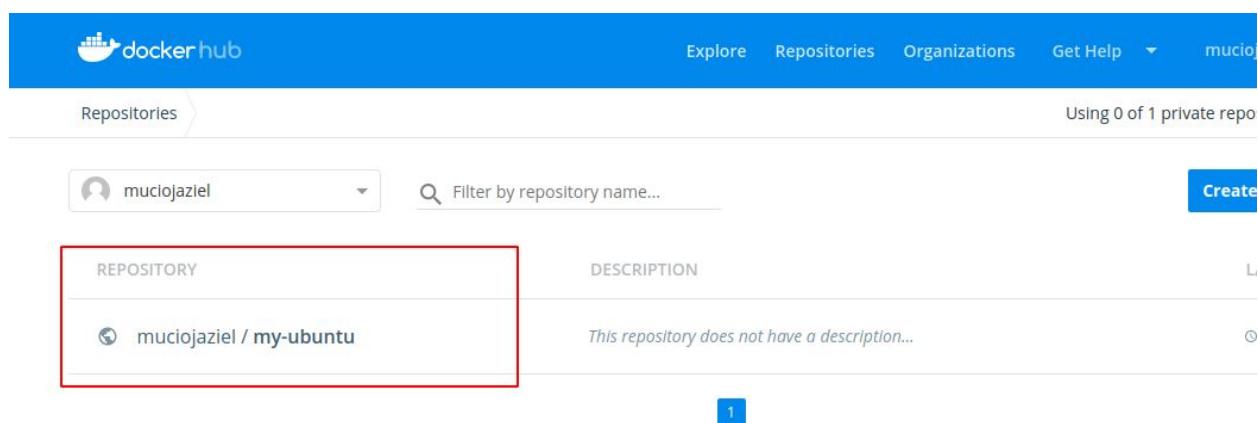
```
$ docker image tag my-ubuntu:dev <docker store username>/my-ubuntu:dev  
$ docker image push <docker store username>/my-ubuntu:dev
```



A terminal window titled 'root@Nx: /home/mucio/novaimg' showing the execution of Docker commands. The commands are:  
\$ docker image tag my-ubuntu:dev <docker store username>/my-ubuntu:dev  
\$ docker image push <docker store username>/my-ubuntu:dev

The output shows the repository being pushed:  
root@Nx:/home/mucio/novaimg# docker image tag my-ubuntu:dev muciojaziel/my-ubuntu:dev  
root@Nx:/home/mucio/novaimg# docker image push muciojaziel/my-ubuntu:dev  
The push refers to repository [docker.io/muciojaziel/my-ubuntu]  
92d3f22d44f3: Mounted from library/ubuntu  
10e46f329a25: Mounted from library/ubuntu  
24ab7de5faec: Mounted from library/ubuntu  
1ea5a27b0484: Mounted from library/ubuntu  
dev: digest: sha256:3288f31c6b3760981b5a9ad8fc13735d236cb1f5bae79b2d0750bcadfb00fed5 size: 1150  
root@Nx:/home/mucio/novaimg#

5. Visite a sua página do Docker Store, encontre o seu novo repositório my-ubuntu e confirme que você pode ver a tag :dev nele. Explore seu novo repo e preencha a descrição e outros campos que você encontrar lá.



Screenshot of the Docker Hub website showing the user's repositories page. The user is logged in as 'muciojaziel'. The page displays one repository: 'muciojaziel / my-ubuntu'. The repository card is highlighted with a red border. The repository details are as follows:

REPOSITORY	DESCRIPTION
muciojaziel / my-ubuntu	This repository does not have a description...

At the bottom of the page, there is a blue button labeled '1'.

The screenshot shows a Docker repository page for 'muciojaziel / my-ubuntu'. At the top, there's a navigation bar with 'Repositories' and the repository name 'muciojaziel / my-ubuntu'. On the right, it says 'Using 0 of 1 private repository'. Below the navigation, there are tabs for 'General', 'Tags', 'Builds', 'Timeline', 'Collaborators', 'Webhooks', and 'Settings'. The 'General' tab is selected.

In the main content area, there's a section for 'Docker commands' with a button labeled 'Public'. Below that, it says 'To push a new tag to this repository,' followed by a command box containing 'docker push muciojaziel/my-ubuntu:tagname'.

Under the 'Tags' heading, it says 'This repository contains 1 tag(s)'. A single tag named 'dev' is listed, with a timestamp of '2 minutes ago'. There's also a link 'See all'.

## Aqui vou deixar como desafio os passos 6, 7, 8 e 9 uma vez que você já sabe vários caminhos!

6. Em seguida, escreva um Dockerfile que use <docker store username>/my-ubuntu:dev como sua imagem de base e instale qualquer aplicativo que você goste além disso. Faça o build da imagem e, simultaneamente, renomeie a tag como: 1.0:

```
$ docker image build -t <docker store username>/my-ubuntu:1.0 .
```

7. Envie sua tag: 1.0 para o Docker Store e confirme que você podevê-la no repositório apropriado.

8. Por fim, liste as imagens atualmente em seu nó com `docker image ls`. Você ainda deve ter a versão da sua imagem que não tenha sido nomeada com o nome de usuário do Docker Store; exclua isso usando `docker image rm`:

```
$ docker image rm my-ubuntu:dev
```

### 9.3 Imagens Privadas

1. Explore a interface do usuário [UI] em seu repositório do Docker Store <username>/my-ubuntu, e localize a opção para tornar esse repositório privado.

2. Junte-se a alguém sentado ao seu lado e tente puxar a imagem dele:

```
$ docker image pull <partners docker store name>/my-ubuntu:1.0
```

Seu repositório privado deve ser invisível para você neste momento.

3. Adicionar uns aos outros como colaboradores para o seu repositório my-ubuntu, e puxar novamente; se tudo

tiver corrido bem, agora você deve conseguir fazer o repo. Além disso, verifique se você pode ver o repo do outro na guia "Repositórios" do seu perfil da Docker Store.

## 9.4 Conclusão

Neste exercício, vimos como nomear e marcar imagens com docker image tag; explorou as convenções de nomenclatura corretas necessárias para enviar imagens para sua conta do Docker Store com docker image push; Aprendi como remover imagens antigas com a docker image rm; e aprendi como tornar o repositório privado e compartilhá-lo com os colaboradores na Docker Store.

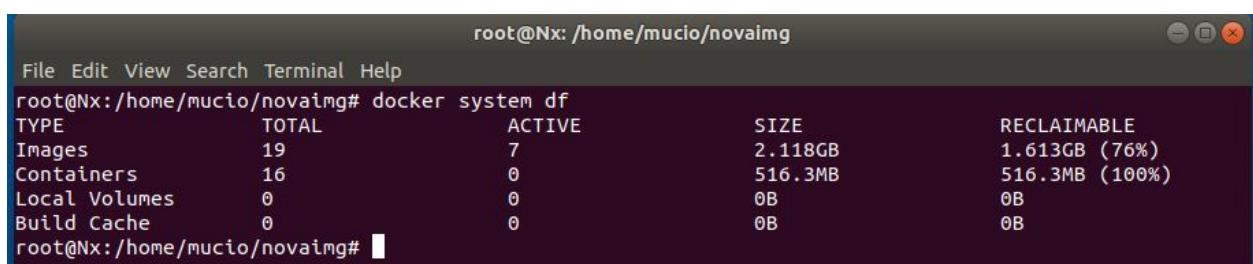
## 10 Comandos de limpeza

Quando começarmos a usar o Docker muito em nosso sistema, queremos entender quais recursos o mecanismo do Docker está usando. Também queremos maneiras de limpar e liberar recursos não utilizados. Por tudo isso, podemos usar os comandos do docker system.

1. Descubra quanto de memória o Docker está usando executando:

```
$ docker system df
```

A saída mostrará quanto espaço as imagens, os contêineres e os volumes (locais) estão ocupando e quanto desse espaço pode ser recuperado.



The screenshot shows a terminal window with the title 'root@Nx:/home/mucio/novaimg'. The window contains the following text:

```
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# docker system df
TYPE          TOTAL        ACTIVE      SIZE      RECLAIMABLE
Images         19           7       2.118GB   1.613GB (76%)
Containers     16           0       516.3MB   516.3MB (100%)
Local Volumes   0            0          0B          0B
Build Cache    0            0          0B          0B
root@Nx:/home/mucio/novaimg#
```

2. Recupere todo o espaço recuperável usando o seguinte comando:

```
$ docker system prune
```

Responda com y quando for perguntado se realmente queremos remover todas as redes, contêineres, imagens e volumes não utilizados.

```

root@Nx: /home/mucio/novaimg#
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# docker system df
TYPE          TOTAL        ACTIVE        SIZE      RECLAIMABLE
Images         19           7       2.118GB   1.613GB (76%)
Containers     16           0       516.3MB   516.3MB (100%)
Local Volumes  0            0          0B        0B
Build Cache    0            0          0B        0B
root@Nx:/home/mucio/novaimg# docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache
Are you sure you want to continue? [y/N] y
Deleted Containers:
56513f1dfde4267077963be573cb3e1d22b627e8fb80360fcfd6e5bac47e92df8
3e0e1a1b9bc7e219f5d9faf3037601d76498c2073d3d5d91de94842b17c7df34
129f6496ae91167fb0a5f7b238dd2ff8a7bee4c588944192c2b4129a9831ef33
dcfdff394320326a421705321b835cd9b36ab321085c121005cced9b45c5e624
077de1d1e5074694af8c6174b597171cf0d450680054fd7e2d553ac1c99b8708
4b81c4b273fb2518c0321c6d3c11e1691cdd1d58272f89d7c626f0a7c1d2cb44
72fd132cb41398b45ae1797c26ec73a773e85adcb836b338c1b46cee37986bfb
8d546069f37c25047199e2e51d22763567e43eae93c76f063158e5101adc4ca4
60cb167fe935bf7a1578a958542133cab25ff56caa666477aab4ab7bf21d3f59
8a78ff26d05a7394c186c3be61299d4fa57e8d5a7e5ddfb0e93da09202745e7c

```

```

root@Nx: /home/mucio/novaimg#
File Edit View Search Terminal Help
deleted: sha256:6a49f8db254023e6ab6deffe28f99b74276c68e7c10d047a21880979ae0fbc4f
deleted: sha256:05b70822e55ce74425f96bfdf6f2cac6880bbeaa51205814d78e57901190a93c
deleted: sha256:94b4ffd5c055493130f90aa1740aa20d3e2abe20f682eda746e84f9f31d88a8d
deleted: sha256:fd4916f52ff64877891b1ddec98727d266f20923ca9bf1468ab9b9520859bbca
untagged: dbaontap/node-add-app@sha256:be9ee774c074159b74480c8ad27eb83738d066348ef8e92e234e5cce9d70
ea4b
deleted: sha256:dc126322ee0c5f2e2cfac86094795bf05ae3489c94acd91c33bbc8e6b4cb90cf
deleted: sha256:e460b2d0e4eb61bdcf6b26d8a70b8e2c885b245c30af1112e918348725ebda33
deleted: sha256:9f37c9881d6010ac8d5a1f6fdc132202eb2c5d0dfe0bd0b735f8221f000c5142
deleted: sha256:68bfe783f368236ee9324a30b5a48fc3b243677281e6305ea336991a4a69b1dd
deleted: sha256:1e722c3cd0284a2d96fe49a0abb19a341dbccea3f094e7ff836a6880b3f33729
deleted: sha256:364066891e3322a86cd8453a205dc726aa4d44c3f0552ee64ea05535d46a258f
deleted: sha256:5f0402380aa4bb517a1d9bfae6d81aa6aea1a13381f8875c5e8c0df00dd8b4f3
deleted: sha256:2841c5abec155d2bddd6e9a580d5de4efb4add1baa38c7eaf6924f22aae4e9b
deleted: sha256:0c81b9638f05f84286f04e0fce636e52701e271321ad5016e9538cc3935c1e8c
deleted: sha256:64d539e91bea5dbdddb784db5d064092354bec7d375279b9b7ffffb412275ad2
deleted: sha256:b949b431c79e8c36b35e5a4811d15c15f500282b5699ae26fb632c58ef40c907
deleted: sha256:86493f284b090b3cc7284db954d6ba85c07a8520d477bf7085ab8eab372dfa65
deleted: sha256:f4bd4838514dc8e91c618fc88ad5262d14d71108fec61d4d6706f18d4ed4fed
deleted: sha256:24b042451c4510b85bf175c99258197580084ade995359b7c4d228370fc6f86a
deleted: sha256:2ac82f8c2e6cc666d91f5069ccf6157f9cc69e19ba982d7bbe47605c12ec25e2

Total reclaimed space: 841.6MB
root@Nx: /home/mucio/novaimg#

```

3. Essa é a maneira mais radical de manter nosso sistema limpo. Se queremos ser mais focados e apenas limpar os recursos de um determinado tipo, podemos usar os seguintes comandos:

```

$ docker image prune
$ docker container prune
$ docker volume prune
$ docker network prune

```

```
root@Nx:/home/mucio/novaimg
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
root@Nx:/home/mucio/novaimg# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
root@Nx:/home/mucio/novaimg# docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
root@Nx:/home/mucio/novaimg# docker network prune
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N] y
root@Nx:/home/mucio/novaimg#
```

Experimente cada um deles. Se você estiver cansado de responder `y` a cada vez, adicione a flag `-f` ou `--force` ao comando, por exemplo:

```
$ docker system prune --force
```

**Isso é especialmente útil se você deseja executar os comandos acima como parte ou um script de automação.**

## 11 Inspecionando Comandos

### 11.1 Informações do sistema

1. Podemos encontrar o comando `info` no sistema. Execute:

```
$ docker system info
```

```
root@Nx: /home/mucio/novaimg
File Edit View Search Terminal Help

root@Nx:/home/mucio/novaimg# docker system info
Containers: 0
Running: 0
Paused: 0
Stopped: 0
Images: 30
Server Version: 18.09.7
Storage Driver: overlay2
Backing Filesystem: extfs
Supports d_type: true
Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
Volume: local
Network: bridge host macvlan null overlay
Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 894b81a4b802e4eb2a91d1ce216b8817763c29fb
runc version: 425e105d5a03fabd737a126ad93d62a9eeede87f
```

2. A partir da saída do último comando, identifique:

- Quantas imagens são armazenadas em cache na sua máquina?
- Quantos contêineres estão sendo executados ou interrompidos?
- Qual versão do container você está executando?
- Se o Docker está sendo executado no modo swarm?

## 11.2 Eventos de sistema

1. Existe outro poderoso comando do sistema que nos permite monitorar o que está acontecendo no host do Docker. Execute o seguinte comando:

```
$ docker system events
```

Por favor, note que parece que o sistema está pendurado, mas esse não é o caso. O sistema está apenas esperando que alguns eventos aconteçam.

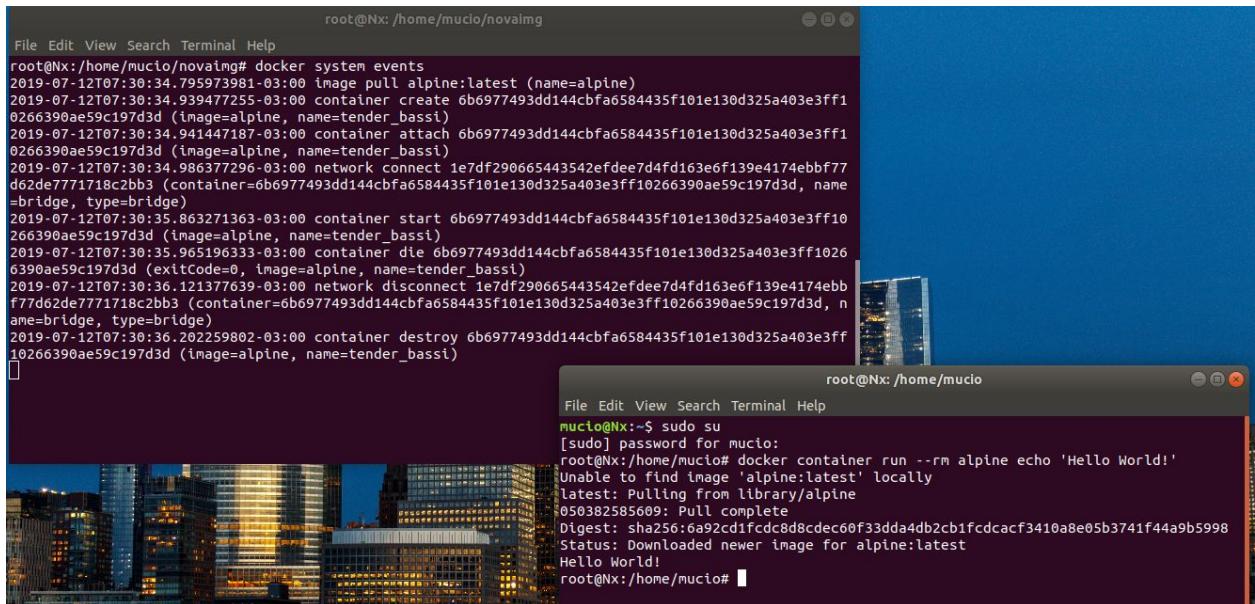
```
root@Nx: /home/mucio/novaimg
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# docker system events
■
```

2. Abra um segundo terminal e execute o seguinte comando:

```
$ docker container run --rm alpine echo 'Hello World!'
```

e observe a saída gerada no primeiro terminal. Deve ser semelhante a isto:

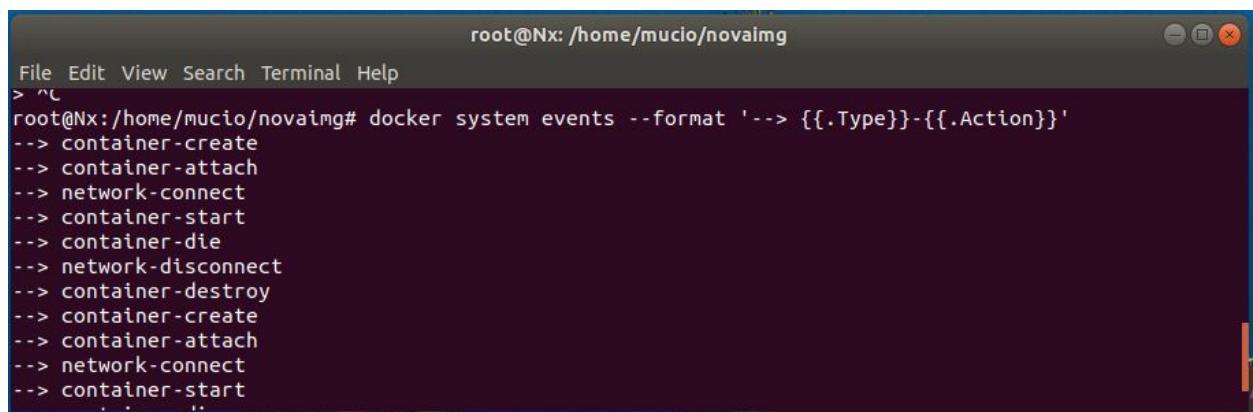
```
2017-01-25T16:57:48.553596179-06:00 container create 30eb63 ...
2017-01-25T16:57:48.556718161-06:00 container attach 30eb63 ...
2017-01-25T16:57:48.698190608-06:00 network connect de1b2b ...
2017-01-25T16:57:49.062631155-06:00 container start 30eb63 ...
2017-01-25T16:57:49.065552570-06:00 container resize 30eb63 ...
2017-01-25T16:57:49.164526268-06:00 container die 30eb63 ...
2017-01-25T16:57:49.613422740-06:00 network disconnect de1b2b ...
2017-01-25T16:57:49.815845051-06:00 container destroy 30eb63 ...
```



3. Se você não gostar do formato da saída, podemos usar o parâmetro `--format` para definir nosso próprio formato na forma de um modelo Go. Pare o relógio de eventos no seu primeiro terminal com `CTRL+C` e tente isto:

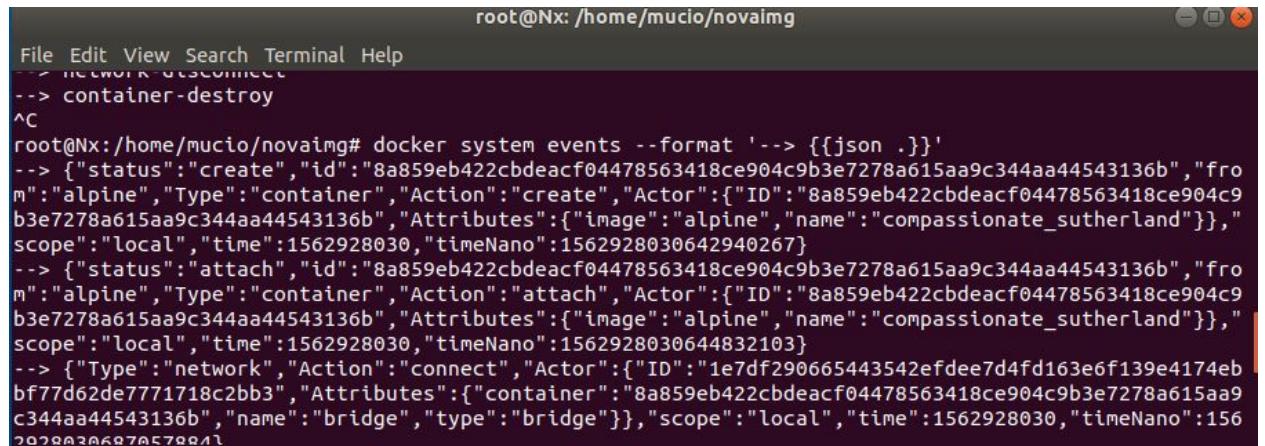
```
$ docker system events --format '---> {{.Type}}-{{.Action}}'
```

agora a saída parece um pouco menos desordenada quando rodamos nosso contêiner alpine no segundo terminal como acima.



4. Finalmente, podemos descobrir como é a estrutura do evento, emitindo os eventos no formato json (mais uma vez, depois de matar o inspetor de eventos no primeiro terminal e reiniciá-lo com):

```
$ docker events --format '{{json .}}'
```



A screenshot of a terminal window titled "root@Nx:/home/mucio/novaimg". The window shows a series of JSON event objects. The first few lines are: "File Edit View Search Terminal Help", "--> network-disconnect", "--> container-destroy", and a command '^C'. Below these, several event objects are listed, each starting with "--> {"status": "create", "id": ...". The objects describe the creation of a container named "sleepy\_roentgen" from the "alpine" image. The terminal window has a dark background with light-colored text.

```
root@Nx:/home/mucio/novaimg# docker system events --format '--> {{json .}}'
--> {"status":"create","id":"8a859eb422cbdeacf04478563418ce904c9b3e7278a615aa9c344aa44543136b","from":"alpine","Type":"container","Action":"create","Actor":{"ID":"8a859eb422cbdeacf04478563418ce904c9b3e7278a615aa9c344aa44543136b","Attributes":{"image":"alpine","name":"compassionate_sutherland"}}, "scope":"local","time":1562928030,"timeNano":1562928030642940267}
--> {"status":"attach","id":"8a859eb422cbdeacf04478563418ce904c9b3e7278a615aa9c344aa44543136b","from":"alpine","Type":"container","Action":"attach","Actor":{"ID":"8a859eb422cbdeacf04478563418ce904c9b3e7278a615aa9c344aa44543136b","Attributes":{"image":"alpine","name":"compassionate_sutherland"}}, "scope":"local","time":1562928030,"timeNano":1562928030644832103}
--> {"Type":"network","Action":"connect","Actor":{"ID":"1e7df290665443542efdee7d4fd163e6f139e4174ebbf77d62de7771718c2bb3","Attributes":{"container":"8a859eb422cbdeacf04478563418ce904c9b3e7278a615aa9c344aa44543136b","name":"bridge","type":"bridge"}}, "scope":"local","time":1562928030,"timeNano":1562928030644870579941}
```

que deve nos dar para o primeiro evento da série depois de re-executar o nosso contêiner alpino no segundo nó algo como isto (note que a saída ficou mais legível):

```
{
  "status": "create",
  "id": "95ddb6ed4c87d67fa98c3e63397e573a23786046e00c2c68a5bcb9df4c17635c",
  "from": "alpine", "Type": "container", "Action": "create", "Actor": {
    "ID": "95ddb6ed4c87d67fa98c3e63397e573a23786046e00c2c68a5bcb9df4c17635c",
    "Attributes": {
      "image": "alpine", "name": "sleepy_roentgen" } }, "time": 1485385702,
  "timeNano": 1485385702748011034
}
```

## 11.3 Resumo

Neste exercício, aprendemos a inspecionar as propriedades de todo o sistema do host do Docker usando o comando `docker system inspect`. Também usamos o comando de eventos do sistema do docker, comando: `docker system events`, para inspecionar ainda mais a atividade no sistema quando contêineres e outros recursos são criados, usados e destruídos.

# 12 Criando e Montando Volumes

## 12.1 Criando um Volume

1. Crie um volume chamado test1:

```
$ docker volume create --name test1
```

The screenshot shows a terminal window with a dark background and white text. At the top, it says "root@Nx: /home/mucio/novaimg". Below that is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal shows the command "root@Nx:/home/mucio/novaimg# docker volume create --name test1" followed by the output "test1". The prompt "root@Nx:/home/mucio/novaimg#" is visible at the bottom.

```
root@Nx: /home/mucio/novaimg
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# docker volume create --name test1
test1
root@Nx:/home/mucio/novaimg#
```

2. Execute docker volume ls e verifique se você pode ver o volume test1.

```
root@Nx:/home/mucio/novaimg# docker volume ls
DRIVER          VOLUME NAME
local           test1
root@Nx:/home/mucio/novaimg#
```

3. Execute um novo contêiner ubuntu e monte o volume test1. Mapeie para o caminho /www/website e execute bash como seu processo:

```
$ docker container run -it -v test1:/www/website ubuntu:16.04 bash
```

The screenshot shows a terminal window with a dark background and white text. At the top, it says "root@8426af343d4d: /". Below that is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal shows the command "root@Nx:/home/mucio/novaimg# docker container run -it -v test1:/www/website ubuntu:16.04 bash" followed by the output "root@8426af343d4d: #". The prompt "root@8426af343d4d: #" is visible at the bottom.

```
root@8426af343d4d: /
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# docker container run -it -v test1:/www/website ubuntu:16.04 bash
root@8426af343d4d: #
```

4. Dentro do contêiner, verifique se você pode acessar /www/website:

```
$ cd /www/website
```

5. Crie um arquivo chamado test.txt insira-o na pasta /www/website:::

```
$ touch test.txt
```

6. Saia do container, sem pará-lo, teclando CTRL + P + Q.

Obs: A imagem contém os passos 4, 5 e 6;

```
root@Nx: /home/mucio/novaimg
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# docker container run -it -v test1:/www/website ubuntu:16.04 bash
root@8426af343d4d:# cd /www/website
root@8426af343d4d:/www/website# touch test.txt
root@8426af343d4d:/www/website# ls
test.txt
root@8426af343d4d:/www/website# root@Nx:/home/mucio/novaimg#
```

## 12.2 Volumes durante a criação de imagens

1. Confirme o contêiner atualizado como uma nova imagem chamada test e insira a tag como 1.0:

```
$ docker container commit <container ID> test:1.0
```

2. Execute um novo container com sua imagem de teste e entre em seu shell bash:

```
$ docker container run -it test:1.0 bash
```

3. Verifique que a pasta existe /www/website. Existem arquivos dentro dele? Saia deste contêiner.

4. Execute docker container ls para garantir que seu primeiro contêiner ainda esteja em execução para a próxima etapa.

Nesta seção, vimos como criar e montar um volume e adicionar dados a ele de dentro de um contêiner. Também vimos que criar uma imagem a partir de um contêiner em execução por meio de docker container commit não captura nenhuma informação de volumes montados dentro do contêiner; volumes e imagens são criados e atualizados de forma completamente independente.

## 12.3 Encontrando o Ponto de Montagem do Host

1. Execute docker volume inspect no volume test1 para descobrir onde ele é montado na máquina host (veja o campo "Mountpoint"):

```
$ docker volume inspect test1
```

```
root@Nx:/home/mucio/novaimg
File Edit View Search Terminal Help
root@8426af343d4d:/www/website# ls
test.txt
root@8426af343d4d:/www/website# root@Nx:/home/mucio/novaimg# docker volume inspect test1
[
  {
    "CreatedAt": "2019-07-12T07:44:36-03:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/test1/_data",
    "Name": "test1",
    "Options": {},
    "Scope": "local"
  }
]
root@Nx:/home/mucio/novaimg#
```

2. Eleve seus privilégios de usuário ao root:

```
$ sudo su
```

3. Altere o diretório para o caminho do volume encontrado na etapa 1:

```
$ cd /var/lib/docker/volumes/test1/_data
```

```
root@Nx: /var/lib/docker/volumes/test1/_data
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# cd /var/lib/docker/volumes/test1/_data
root@Nx:/var/lib/docker/volumes/test1/_data#
```

4. Execute `ls` e verifique se você pode ver o arquivo `test.txt` que você criou dentro do seu contêiner original.

```
root@Nx: /var/lib/docker/volumes/test1/_data
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# cd /var/lib/docker/volumes/test1/_data
root@Nx:/var/lib/docker/volumes/test1/_data# ls
test.txt
root@Nx:/var/lib/docker/volumes/test1/_data#
```

5. Crie outro arquivo chamado `test2.txt` dentro deste diretório:

```
$ touch test2.txt
```

```
root@Nx: /var/lib/docker/volumes/test1/_data
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# cd /var/lib/docker/volumes/test1/_data
root@Nx:/var/lib/docker/volumes/test1/_data# ls
test.txt
root@Nx:/var/lib/docker/volumes/test1/_data# touch test2.txt
root@Nx:/var/lib/docker/volumes/test1/_data# ls
test2.txt  test.txt
root@Nx:/var/lib/docker/volumes/test1/_data#
```

6. Saia da conta de superusuário:

```
$ exit
```

7. Use `docker container exec` para entrar novamente no shell do seu contêiner `ubuntu` que ainda está em execução:

```
$ docker container exec -it <container name> bash
```

8. Altere o diretório para a pasta `/www/website` e verifique se você pode ver os arquivos `test.txt` e `test2.txt`. Os arquivos gravados no volume no host listado pelo `docker volume inspect` da janela de encaixe aparecem no sistema de arquivos de cada contêiner que o monta.

```
root@8426af343d4d: /www/website
File Edit View Search Terminal Help
root@Nx:/var/lib/docker/volumes/test1/_data# docker container exec -it 8426af343d4d bash
root@8426af343d4d:/# cd /www/website
root@8426af343d4d:/www/website# ls
test.txt test2.txt
root@8426af343d4d:/www/website#
```

## 12.4 Deletando Volumes

1. Depois de sair do seu contêiner e retornar ao prompt bash do host, tente excluir o volume test1:

```
$ docker volume rm test1
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker volume rm test1
```

2. Exclua o recipiente restante sem usar nenhuma opção:

```
$ docker container rm -f <container ID>
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container rm -f 8426af343d4d9
```

3. Execute docker volume ls e verifique o resultado; observe que nosso volume test1 ainda está presente, já que a remoção de contêineres não afeta os contêineres montados.

```
root@Nx:/home/mucio/novaimg# docker volume ls
DRIVER      VOLUME NAME
local       test1
```

4. Verifique se os arquivos test.txt e test2.txt também ainda estão presentes em seu volume no host:

```
$ sudo ls /var/lib/docker/volumes/test1/_data
```

```
root@Nx: /var/lib/docker/volumes/test1/_data
File Edit View Search Terminal Help
root@Nx:/home/mucio/novaimg# cd /var/lib/docker/volumes/test1/_data
root@Nx:/var/lib/docker/volumes/test1/_data# ls
test.txt
root@Nx:/var/lib/docker/volumes/test1/_data# touch test2.txt
root@Nx:/var/lib/docker/volumes/test1/_data# ls
test2.txt test.txt
root@Nx:/var/lib/docker/volumes/test1/_data# 
```

5. Delete o volume test1:

```
$ docker volume rm test1
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker volume rm test1
test1
root@Nx:/home/mucio# 
```

6. Execute docker volume ls e certifique-se de que o volume test1 foi de fato deletado..

```
root@Nx:/home/mucio# docker volume ls
DRIVER      VOLUME NAME
root@Nx:/home/mucio# 
```

## 12.5 Conclusão

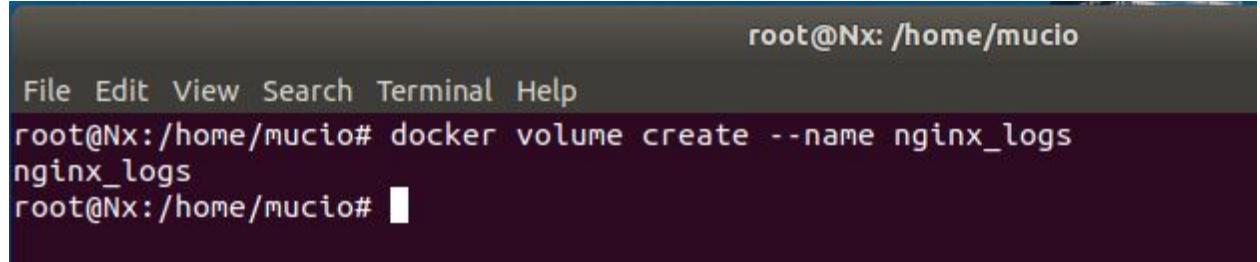
Os volumes são a ferramenta preferida do Docker para persistir dados além do tempo de vida de um contêiner. Neste exercício, vimos como criar e destruir volumes; como montar volumes ao executar um contêiner; como localizar seus locais no host (em /var/lib/docker/volumes) e no contêiner usando docker volume inspect e docker container inspect; e como eles interagem com o sistema de arquivos de união do contêiner.

## 13 Volumes Caso de uso: registros de gravação

### 13.1 Configurar um aplicativo com registro [ logging ]

1. Crie um volume chamado nginx\_logs:

```
$ docker volume create --name nginx_logs
```

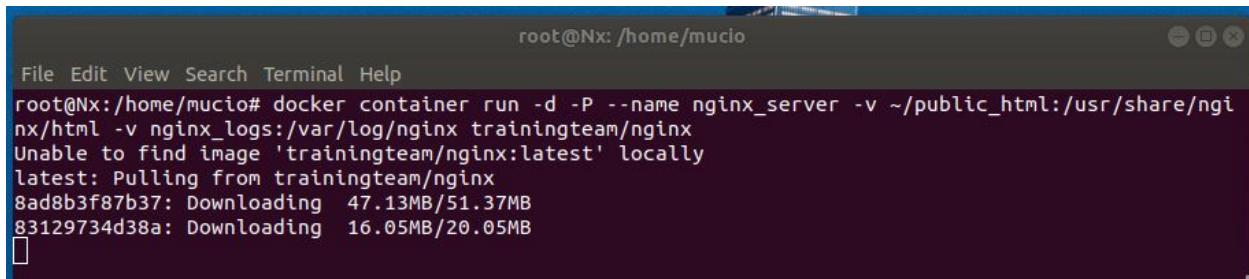


```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker volume create --name nginx_logs
nginx_logs
root@Nx:/home/mucio#
```

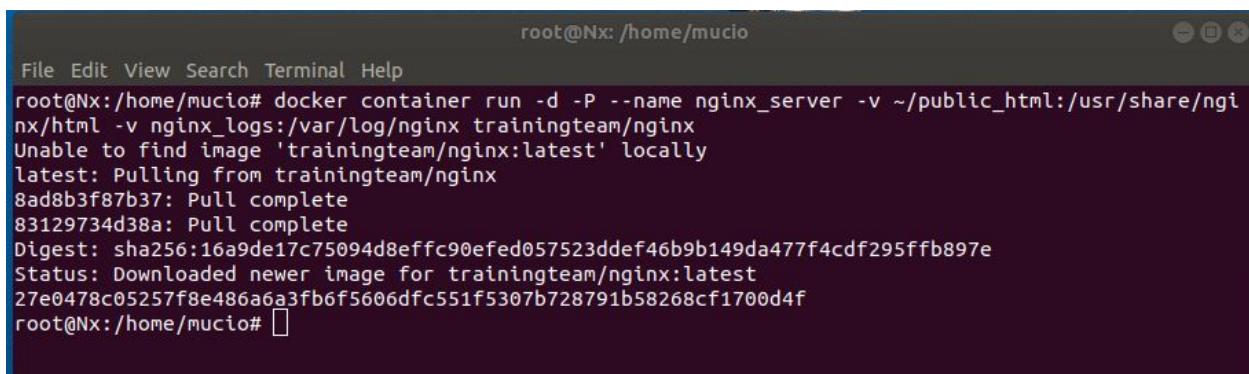
2. Execute o container `trainingteam/nginx` personalizado e mapeie sua pasta host `public_html` para um diretório em `/usr/share/nginx/html`. Monte também o seu volume `nginx_logs` na pasta `/var/log/nginx`. Nomeie o contêiner `nginx_server`:

```
$ docker container run -d -P --name nginx_server \
-v ~/public_html:/usr/share/nginx/html \
-v nginx_logs:/var/log/nginx \
trainingteam/nginx
```

Obs: Você pode escrever direto, ou fazer uso das barras invertidas \ . **Escrevi direto conforme imagem.**



```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container run -d -P --name nginx_server -v ~/public_html:/usr/share/nginx/html -v nginx_logs:/var/log/nginx trainingteam/nginx
Unable to find image 'trainingteam/nginx:latest' locally
latest: Pulling from trainingteam/nginx
8ad8b3f87b37: Downloading 47.13MB/51.37MB
83129734d38a: Downloading 16.05MB/20.05MB
[]
```



```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container run -d -P --name nginx_server -v ~/public_html:/usr/share/nginx/html -v nginx_logs:/var/log/nginx trainingteam/nginx
Unable to find image 'trainingteam/nginx:latest' locally
latest: Pulling from trainingteam/nginx
8ad8b3f87b37: Pull complete
83129734d38a: Pull complete
Digest: sha256:16a9de17c75094d8effc90efed057523ddef46b9b149da477f4cdf295ffb897e
Status: Downloaded newer image for trainingteam/nginx:latest
27e0478c05257f8e486a6a3fb6f5606dfc551f5307b728791b58268cf1700d4f
root@Nx:/home/mucio#
```

3. Obtenha acesso ao terminal para o seu contêiner:

```
$ docker container exec -it nginx_server bash
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container exec -it nginx_server bash
root@27e0478c0525:# 
```

4. Coloque um texto em /usr/share/nginx/html/index.html e saia do terminal.

Obs: Foi preciso atualizar a imagem, com :

```
$ apt-get update
```

E instalar o vim pra poder alterar o index.html:

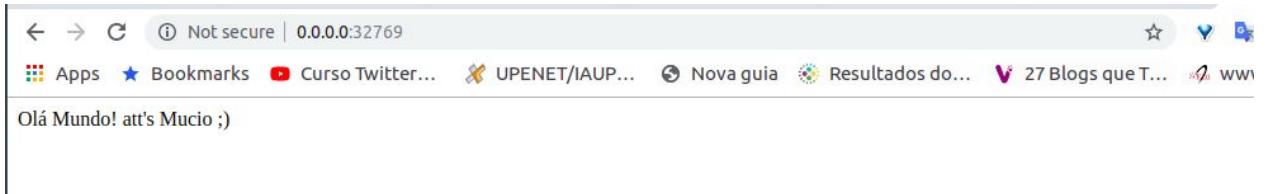
```
$ apt-get install vim
```

```
root@Nx: /ho
File Edit View Search Terminal Help
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <title>Título da página</title>
    <meta charset="utf-8">
  </head>
  <body>
    Olá Mundo! att's Mucio ;)
  </body>
</html>
~
~
~
~
:wq 
```

5. Execute docker container ls para localizar a porta do host que é mapeada para a porta 80 no contêiner. Em seu navegador, accese a URL e a porta em que o nginx está exposto.

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container ls
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          NAMES
           PORTS
27e0478c0525        trainingteam/nginx   "nginx -g 'daemon of..."   17 minutes ago   Up 17 minutes   nginx_server
root@Nx:/home/mucio# 
```

6. Verifique se você pode ver o conteúdo do seu arquivo `index.html` da sua pasta `public_html` no seu host.



## 13.2 Inspeccione os Logs no seu Host

1. Obtenha o acesso do terminal ao seu contêiner novamente:

```
$ docker container exec -it nginx_server bash
```

A screenshot of a terminal window titled "root@Nx: /home/mucio". It shows the command `docker container ls` output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
27e0478c0525	trainingteam/nginx	"nginx -g 'daemon off...'	17 minutes ago	Up 17 minutes
		0.0.0.0:32769->80/tcp, 0.0.0.0:32768->443/tcp		
		nginx_server		

The command `root@27e0478c0525:/# docker container exec -it nginx_server bash` is entered and the prompt changes to `root@27e0478c0525:/#`.

2. Altere o diretório para `/var/log/nginx`:

```
$ cd /var/log/nginx
```

```
root@27e0478c0525:/# cd /var/log/nginx
root@27e0478c0525:/var/log/nginx#
```

3. Verifique se você pode ver os arquivos `access.log` e `error.log`.

```
root@27e0478c0525:/var/log/nginx# ls
access.log  error.log
root@27e0478c0525:/var/log/nginx#
```

4. Execute `tail -f access.log`, atualize seu navegador algumas vezes e observe as entradas de log sendo gravadas no arquivo. **Saia do terminal** de contêineres depois de ver algumas entradas de registro ao vivo.

```
root@27e0478c0525:/var/log/nginx# tail -f access.log
172.17.0.1 - - [12/Jul/2019:12:02:40 +0000] "GET / HTTP/1.1" 200 178 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36"
172.17.0.1 - - [12/Jul/2019:12:02:41 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "http://0.0.0.0:32769/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36"
172.17.0.1 - - [12/Jul/2019:12:02:54 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36"
```

```
root@27e0478c0525:/var/log/nginx# tail -f access.log
172.17.0.1 - - [12/Jul/2019:12:02:40 +0000] "GET / HTTP/1.1" 200 178 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:02:41 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "http://0.0.0.0:32769/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:02:54 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:26 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:27 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:28 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:29 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:30 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"

[REDACTED]
```

5. Execute `$ docker volume inspect nginx_logs` e copie o caminho indicando o campo “Mountpoint”; o caminho deve ser `/var/lib/docker/volumes/nginx_logs/_data`.

```
root@Nx:/home/mucio# docker volume inspect nginx_logs
[
  {
    "CreatedAt": "2019-07-12T08:31:58-03:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/nginx_logs/_data",
    "Name": "nginx_logs",
    "Options": {},
    "Scope": "local"
  }
]
root@Nx:/home/mucio#
```

6. Verifique a presença dos arquivos `access.log` e `error.log`, então siga o final do `access.log`:

```
$ sudo ls /var/lib/docker/volumes/nginx_logs/_data
```

```
root@Nx:/home/mucio# sudo ls /var/lib/docker/volumes/nginx_logs/_data
access.log  error.log
```

```
$ sudo tail -f /var/lib/docker/volumes/nginx_logs/_data/access.log
```

```
root@Nx:/home/mucio# sudo tail -f /var/lib/docker/volumes/nginx_logs/_data/access.log
172.17.0.1 - - [12/Jul/2019:12:02:40 +0000] "GET / HTTP/1.1" 200 178 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:02:41 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "http://0.0.0.0:32769/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:02:54 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:26 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:27 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:28 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:29 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:30 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"

[REDACTED]
```

7. Atualize seu navegador algumas vezes para fazer algumas solicitações ao servidor NGINX; observe as entradas de log sendo gravadas no arquivo `access.log`, disponível no volume `nginx_logs` em sua máquina host.

```

root@Nx: /home/mucio
File Edit View Search Terminal Help
172.17.0.1 - - [12/Jul/2019:12:02:40 +0000] "GET / HTTP/1.1" 200 178 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:02:41 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "http://0.0.0.0:32769/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:02:54 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:26 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:27 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:28 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:29 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:07:30 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:13:10 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"
172.17.0.1 - - [12/Jul/2019:12:13:10 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36" "-"

```

### 13.3 Compartilhando Volumes

- Execute `$ docker container ls` e certifique-se de que seu contêiner `nginx_server` da última etapa ainda esteja em execução; se não, reinicie-o.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
27e0478c0525	trainingteam/nginx	"nginx -g 'daemon off...'	42 minutes ago	Up 42 minutes	0.0.0.0:32769->80/tcp, 0.0.0.0:32768->443/tcp

- Execute um novo container `ubuntu` e monte um volume `nginx_logs` na pasta `/data/mylogs` como somente leitura, com bash como seu processo:

```

$ docker container run -it \
-v nginx_logs:/data/mylogs:ro \
ubuntu:14.04 bash

```

```

root@960b8c739181: /
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container run -it -v nginx_logs:/data/mylogs:ro ubuntu:14.04 bash
root@960b8c739181:#

```

- No terminal do seu novo contêiner, altere o diretório para `/data/mylogs`
- Confirme que você pode ver os arquivos `access.log` e `error.log`.
- Experimente e crie um novo arquivo chamado `text.txt`

```
$ touch test.txt
```

Observe como ele falha porque montamos o volume como somente leitura.

```
root@960b8c739181:/# cd /data/mylogs
root@960b8c739181:/data/mylogs# ls
access.log  error.log
root@960b8c739181:/data/mylogs# touch text.txt
touch: cannot touch 'text.txt': Read-only file system ←
root@960b8c739181:/data/mylogs#
```

Obs: A imagem contém os passos de 3 a 5;

## 13.4 Conclusão

Neste exercício, você explorou como os volumes de montagem fazem com que dados em tempo real sejam gerados em um contêiner disponível para outros contêineres e para o mundo externo; as informações que o nginx estava gravando no volume de registro podem ser consumidas em tempo real por aplicativos de monitoramento independentes que sobreviveriam à falha ou exclusão do contêiner nginx.

Também usamos a flag `:ro` para montar um volume no modo somente leitura **[Read Only]**, para que o contêiner correspondente não danifique nenhum dado no volume. Essa também é uma prática recomendada de segurança importante, já que o compartilhamento de volumes entre contêineres interrompe o isolamento entre contêineres; Ao permitir o acesso somente leitura, impedimos que o contêiner injete quaisquer dados mal-intencionados no volume que apareceria em todos os outros contêineres que usam esse volume, bem como no host.

## 14 Demonstração: Docker Plugins

Plugins são usados para estender os recursos do Docker Engine. Qualquer um, não apenas o Docker, pode implementar plugins. Atualmente, somente os plug-ins de volume e driver de rede são suportados, mas no futuro, o suporte a mais tipos será adicionado.

### 14.1 Installing a Plugin

1. Os plug-ins podem ser hospedados no Docker Store ou em qualquer outro repositório (privado). Vamos começar com a Docker Store. Navegue até <https://store.docker.com>, digite `tiborvass/rexray-plugin` na caixa de pesquisa e clique em ‘Community’ em Filters->Type à esquerda para ver os plug-ins feitos pelos community members. O resultado deve mostrar o plug-in com o qual vamos trabalhar.

---

2. Instale um plug-in que armazene volumes no Amazon EBS em nosso Docker Engine:

```
$ docker plugin install tiborvass/rexray-plugin
```

Seu sistema deve pedir permissão para usar privilégios; respondendo com `y`.

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker plugin install tiborvass/rexray-plugin
Plugin "tiborvass/rexray-plugin" is requesting the following privileges:
- network: [host]
- mount: [/dev]
- allow-all-devices: [true]
- capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
latest: Pulling from tiborvass/rexray-plugin
abd92ccfc91: Download complete
Digest: sha256:df5dbbb15a797102741d1991ecf4b8015b118e563edcba1b4274552fe2481be7
Status: Downloaded newer image for tiborvass/rexray-plugin:latest
Installed plugin tiborvass/rexray-plugin
root@Nx:/home/mucio#
```

3. Uma vez que tenhamos instalado alguns plugins com sucesso, podemos usar o comando `ls` para ver o status de cada um dos plugins instalados. Execute:

```
$ docker plugin ls
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker plugin ls
ID          NAME                  DESCRIPTION          ENABLED
1affadbc5f0b  tiborvass/rexray-plugin:latest  A rexray volume plugin for Docker  true
root@Nx:/home/mucio#
```

## 14.2 Inspecionando, habilitando e desabilitando um plug-in

1. Podemos inspecionar um plugin via:

```
$ docker plugin inspect tiborvass/rexray-plugin
```

Observe que o acesso e as chaves secretas não estão definidas no momento.

```
tiborvass/rexray-plugin-test - A rexray volume plugin for Docker
root@Nx:/home/mucio# docker plugin inspect tiborvass/rexray-plugin
[
  {
    "Config": {
      "Args": [
        "Description": "",
        "Name": "",
        "Settable": null,
        "Value": null
      ],
      "Description": "A rexray volume plugin for Docker",
      "Documentation": "https://docs.docker.com/engine/extend/plugins/",
      "Entrypoint": [
        "/usr/bin/rexray",
        "service",
        "start",
        "-f"
      ],
      "Env": [
        {
          "Name": "REXRAY_PLUGIN_NAME",
          "Value": "tiborvass/rexray-plugin"
        }
      ]
    }
  }
]
```

2. Depois que um plugin é instalado, ele é ativado por padrão. Podemos desabilitá-lo usando este comando:

```
$ docker plugin disable tiborvass/rexray-plugin
```

somente quando um plugin é desativado, certas operações nele podem ser executadas, como alterar as configurações.

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker plugin disable tiborvass/rexray-plugin
tiborvass/rexray-plugin
root@Nx:/home/mucio#
```

3. Defina o token de acesso necessário para que este plug-in possa gravar em:

```
$ AWS_ACCESSKEY=XXX
```

```
$ AWS_SECRETKEY=YYY
```

Obs: Aqui deve ser as credenciais usadas no console da Amazon.

```
root@Nx:/home/mucio# AWS_ACCESSKEY=XXX && AWS_SECRETKEY=YYY
root@Nx:/home/mucio#
```

```
$ docker plugin set tiborvass/rexray-plugin \
```

```
EBS_ACCESSKEY=$AWS_ACCESSKEY EBS_SECRETKEY=$AWS_SECRETKEY
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker plugin set tiborvass/rexray-plugin EBS_ACCESSKEY=$AWS_ACCESSKEY EBS_SECRETKEY=$AWS_SECRETKEY
root@Nx:/home/mucio#
```

4. O plugin pode ser (re) ativado usando este comando:

```
$ docker plugin enable tiborvass/rexray-plugin  
root@Nx:/home/mucio# docker plugin enable tiborvass/rexray-plugin  
tiborvass/rexray-plugin  
root@Nx:/home/mucio#
```

### 14.3 Usando o Plugin

1. Para usar o plugin agora criamos um Docker volume:

```
$ docker volume create -d tiborvass/rexray-plugin my-ebs-volume
```

2. Agora podemos usar esse volume para acessar a pasta remota e trabalhar com ela da seguinte maneira. Execute o seguinte comando para executar um contêiner alpine que tenha acesso ao volume remoto:

```
$ docker container run -v my-ebs-volume:/volume-demo -it ubuntu:16.04 /bin/bash  
root@Nx:/home/mucio# docker container run -v my-ebs-volume:/volume-demo -it ubuntu:16.04 /bin/bash  
root@4b2f43629d66:#
```

3. Dentro do contêiner, navegue até a pasta/volume-demo e crie um novo arquivo:

```
$ cd /volume-demo  
  
$ touch data.dat
```

```
root@4b2f43629d66:/volume-demo  
File Edit View Search Terminal Help  
root@Nx:/home/mucio# docker volume create -d tiborvass/rexray-plugin myVolume  
Error response from daemon: create myVolume: VolumeDriver.Create: {"Error": "error getting volume"}  
root@Nx:/home/mucio# docker container run -v my-ebs-volume:/volume-demo -it ubuntu:16.04 /bin/bash  
root@4b2f43629d66:# cd /volume-demo  
root@4b2f43629d66:/volume-demo# touch data.dat  
root@4b2f43629d66:/volume-demo#
```

4. Dê uma olhada no Amazon EC2 console - seu EBS volume chamado my-ebs-volume deve estar presente.

**Siga este guia, para poder acessar o console Amazon.**

[https://docs.aws.amazon.com/pt\\_br/AWSEC2/latest/UserGuide/ebs-describing-volumes.html](https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/ebs-describing-volumes.html)

### 14.4 Removendo um Plugin

1. Se não quisermos ou precisarmos mais deste plug-in, podemos removê-lo usando o comando:

```
$ docker plugin disable tiborvass/rexray-plugin  
$ docker plugin rm tiborvass/rexray-plugin
```

Observe como primeiro precisamos desabilitar o plug-in antes que possamos removê-lo.

```
root@Nx:/home/mucio# docker plugin disable tiborvass/rexray-plugin
tiborvass/rexray-plugin
root@Nx:/home/mucio# docker plugin rm tiborvass/rexray-plugin
tiborvass/rexray-plugin
root@Nx:/home/mucio# █
```

## 14.5 Summary

Nesta tarefa, aprendemos a instalar, inspecionar e remover plug-ins em ou a partir de um mecanismo Docker. Instalamos especificamente um plug-in que nos permite acessar um volume remoto no Amazon EBS. Em seguida, criamos um Docker volume que usa esse plug-in e demonstrou seu uso.

# 15 Introdução à rede de contêineres

## 15.1 O padrão Bridge Network

1. Primeiro, vamos investigar a Bridge do Linux que o Docker fornece por padrão. Comece instalando os utilitários de Bridge:

```
$ apt-get install bridge-utils
```

```
root@Nx: /home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# apt-get install bridge-utils
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libjavascriptcoregtk-1.0-0 libllvm7 libllvm7:i386 libmypaint libwebkitgtk-1.0-0 mypaint-brushes
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  bridge-utils
0 upgraded, 1 newly installed, 0 to remove and 19 not upgraded.
Need to get 30,1 kB of archives.
After this operation, 102 kB of additional disk space will be used.
Get:1 http://br.archive.ubuntu.com/ubuntu bionic/main amd64 bridge-utils amd64 1.5-15ubuntu1 [30,1 kB]
Fetched 30,1 kB in 0s (124 kB/s)
Selecting previously unselected package bridge-utils.
(Reading database ... 195606 files and directories currently installed.)
Preparing to unpack .../bridge-utils_1.5-15ubuntu1_amd64.deb ...
Unpacking bridge-utils (1.5-15ubuntu1) ...
Setting up bridge-utils (1.5-15ubuntu1)
```

2. Peça informações sobre o padrão Docker linux bridge, Docker0:

```
$ brctl show docker0
```

```
root@Nx:/home/mucio# brctl show docker0
bridge name      bridge id          STP enabled    interfaces
docker0          8000.0242a911e701    no            vethc5c9941
root@Nx:/home/mucio#
```

3. Inicie alguns contêineres nomeados e verifique novamente:

```
$ docker container run --name=u1 -dt ubuntu:14.04
$ docker container run --name=u2 -dt ubuntu:14.04
$ brctl show docker0
```

Você deverá ver duas novas conexões de ethernet virtual (veth) para a ponte, uma para cada contêiner. As conexões veth são um recurso do Linux para criar um ponto de acesso para um namespace de rede em área restrita.

```
root@Nx:/home/mucio# docker container run --name=u1 -dt ubuntu:14.04
8ae38780dd3c467637c9207b04e7623a77aa97f665258de6739d0d0449a5c026
root@Nx:/home/mucio# docker container run --name=u2 -dt ubuntu:14.04
64f1572d7f092d1439bcea1f30c74c87e9061a794f8dc1880c1d3b14525a8cc9
root@Nx:/home/mucio# brctl show docker0
bridge name      bridge id          STP enabled    interfaces
docker0          8000.0242a911e701    no            veth1e8fc02
                                         vethc5c9941
                                         vetheaabee4
root@Nx:/home/mucio#
```

4. O comando **docker network inspect** rende informações de rede sobre quais contêineres estão conectados à rede especificada; a rede padrão é sempre chamada de bridge, então execute:

```
$ docker network inspect bridge
```

e encontre o IP do seu contêiner u1.

```
root@Nx:/home/mucio# docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "1e7df290665443542efdee7d4fd163e6f139e4174ebbf77d62de7771718c2bb3",
    "Created": "2019-07-12T05:41:01.077Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Containers": {}
  }
]
```

5. Conecte-se ao contêiner u2 de seus contêineres usando

```
$ docker container exec -it u2 /bin/bash.
```

```
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container exec -it u2 /bin/bash
root@64f1572d7f09:/#
```

6. De dentro u2, tente pingando o container u1 pelo endereço IP que você encontrou na etapa anterior; então tente pingando u1 pelo nome do contêiner, ping u1 - Observe que a pesquisa funciona com o IP, mas não com o nome do contêiner nesse caso.

```
root@Nx:/home/mucio# docker container exec -it u2 /bin/bash
root@64f1572d7f09:/# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.238 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.059 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.109 ms
```

7. Ainda dentro do contêiner u2 execute ip a para ver algumas informações sobre como é a conexão de rede de dentro do contêiner. Encontre a entrada eth0 e confirme se o endereço MAC e o IP atribuídos são os mesmos (o Docker sempre atribui os pares MAC e IP dessa maneira, para evitar colisões).

```
File Edit View Search Terminal Help
root@64f1572d7f09:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
41: eth0@if42: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
root@64f1572d7f09:/#
```

8. Por fim, de volta ao host, execute docker container inspect u2 e procure a chave NetworkSettings para ver como essa conexão se parece de fora do namespace de rede do contêiner.

```
root@Nx:/home/mucio
File Edit View Search Terminal Help
Labels : {}

},
"NetworkSettings": {
    "Bridge": "",
    "SandboxID": "b833a49dbbf5a6df9bad257561ea6fa92bcfa1f21e906a4c0cdc072ed1a23f1d",
    "HairpinMode": false,
    "LinkLocalIPv6Address": "",
    "LinkLocalIPv6PrefixLen": 0,
    "Ports": {},
    "SandboxKey": "/var/run/docker/netns/b833a49dbbf5",
    "SecondaryIPAddresses": null,
    "SecondaryIPv6Addresses": null,
    "EndpointID": "d8087ad67ed78169de0a553486db661bd0d2557a451de2335413f651d50165dc",
    "Gateway": "172.17.0.1",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "172.17.0.4",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "MacAddress": "02:42:ac:11:00:04",
    "Networks": {
        "bridge": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": null,
            "NetworkID": "1e7df290665443542efdee7d4fd163e6f139e4174ebbf77d62de7771718c2bb3",
            "EndpointID": "d8087ad67ed78169de0a553486db661bd0d2557a451de2335413f651d50165dc",
            "Gateway": "172.17.0.1",
            "IPAddress": "172.17.0.4",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:11:00:04",
            "DriverOpts": null
        }
    }
}
```

## 15.2 Redes Bridge definidas pelo usuário

Na última etapa, investigamos a rede de ponte padrão; Agora vamos tentar fazer o nosso. As redes de pontes definidas pelo usuário funcionam exatamente da mesma forma que as padrão, mas fornecem a pesquisa de DNS pelo nome do contêiner e são protegidas por firewall de outras redes por padrão.

1. Criamos uma rede bridge usando o bridge driver com docker network create:

```
$ docker network create --driver bridge my_bridge
```

2. Examine quais redes estão disponíveis em seu host:

```
$ docker network ls
```

Você deveria ver `my_bridge` e `bridge`, as duas redes de bridge, bem como `none` e `host` - são outras duas redes padrão que não fornecem pilha de rede nem se conectam à pilha de rede do host, respectivamente.

```
root@Nx:/home/mucio# File Edit View Search Terminal Help
root@Nx:/home/mucio# docker network create --driver bridge my_bridge
ce92f7fdef6ccac382cabfeb85a4ddc6831dd79d33d44f92ea89f1d511e89da7
root@Nx:/home/mucio# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
1e7df2906654   bridge    bridge      local
009a14763e99   host      host       local
ce92f7fdef6c   my_bridge bridge      local
3fe8e1b04bb3   none     null       local
root@Nx:/home/mucio#
```

3. Inicie um container conectado à sua nova rede através do `--network` flag:

```
$ docker container run --name=u3 --network=my_bridge -dt ubuntu:14.04
```

```
root@Nx:/home/mucio# File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container run --name=u3 --network=my_bridge -dt ubuntu:14.04
87a751cd43469416c263ab5bf83cbcdaee40118fc086887954f4973e4d712d
root@Nx:/home/mucio#
```

4. Use o comando `inspect` investigar as configurações de rede deste contêiner:

```
$ docker container inspect u3
```

`my_bridge` deve ser listado sob a chave da rede, `Networks` key.

```

Edit View Search Terminal Help
    "SecondaryIPv6Addresses": null,
    "EndpointID": "",
    "Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "",
    "IPPrefixLen": 0,
    "IPv6Gateway": "",
    "MacAddress": "",
    "Networks": {
        "my_bridge": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": [
                "87a751cdae43"
            ],
            "NetworkID": "ce92f7fdef6ccac382cabfeb85a4ddc6831dd79d33d44f92ea89f1d511e89da7",
            "EndpointID": "9ecf77be58d23dfded65e32b7a54fdc976a0e994d994a24517531d60d5cf4506",
            "Gateway": "172.18.0.1",
            "IPAddress": "172.18.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:12:00:02",
            "DriverOpts": null
        }
    }
}

```

5. Lançar outro container, desta vez interativamente:

```
$ docker container run --name=u4 --network=my_bridge -it ubuntu:14.04
```

```

root@63cf8e1492c9: /
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container run --name=u4 --network=my_bridge -it ubuntu:14.04
root@63cf8e1492c9:/# []

```

Desafio! Os conceitos até aqui apresentados o farão desenvolver os passos 6 e 7.

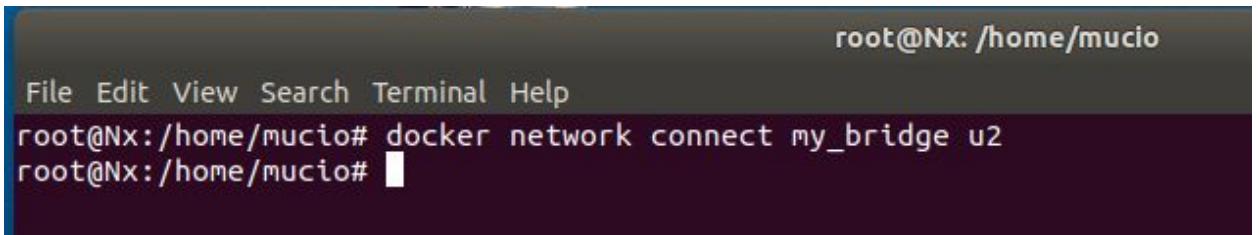
6. De dentro do container u4, ping u3 pelo nome: ping u3. Lembre-se de que isso não funcionou na rede de ponte padrão entre u1 e u2; A pesquisa de DNS pelo nome do contêiner é ativada somente para redes criadas explicitamente.

7. Finalmente, tente pingando u1 pelo IP ou o nome do container como você fez no passo anterior, desta vez do container u4. u1 (e u2) não são alcançáveis a partir de u4 (ou u3), desde que residem em redes diferentes; Todas as redes Docker são protegidas por firewalls umas das outras por padrão.

### 15.3 Comunicação Inter-Container

1. Lembre-se de seu container u2 está atualmente conectado apenas ao padrão de rede bridge; confirme isso usando docker container inspect u2. Conectar u2 para rede my\_bridge:

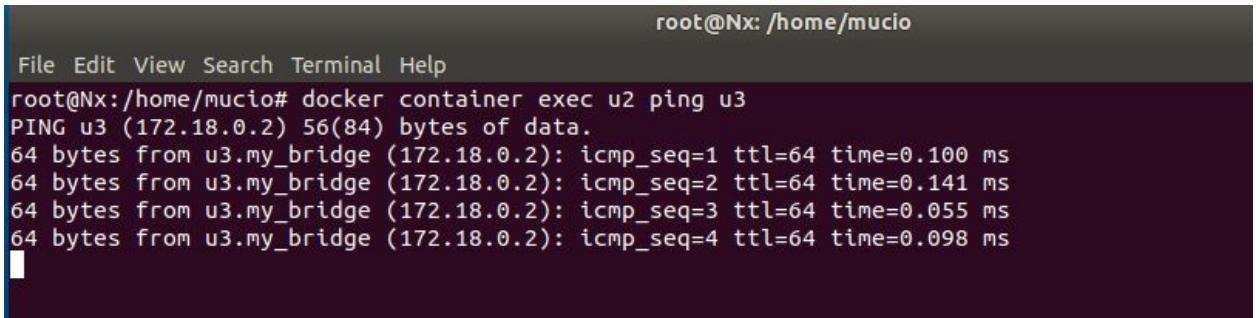
```
$ docker network connect my_bridge u2
```



```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker network connect my_bridge u2
root@Nx:/home/mucio#
```

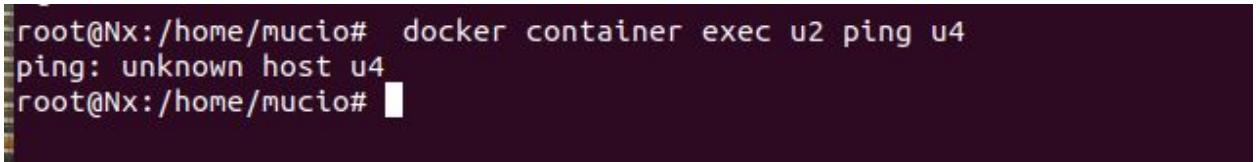
2. Verifique se você pode executar ping nos contêineres u3 e u4 da u2:

```
$ docker container exec u2 ping u3
```



```
root@Nx:/home/mucio
File Edit View Search Terminal Help
root@Nx:/home/mucio# docker container exec u2 ping u3
PING u3 (172.18.0.2) 56(84) bytes of data.
64 bytes from u3.my_bridge (172.18.0.2): icmp_seq=1 ttl=64 time=0.100 ms
64 bytes from u3.my_bridge (172.18.0.2): icmp_seq=2 ttl=64 time=0.141 ms
64 bytes from u3.my_bridge (172.18.0.2): icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from u3.my_bridge (172.18.0.2): icmp_seq=4 ttl=64 time=0.098 ms
```

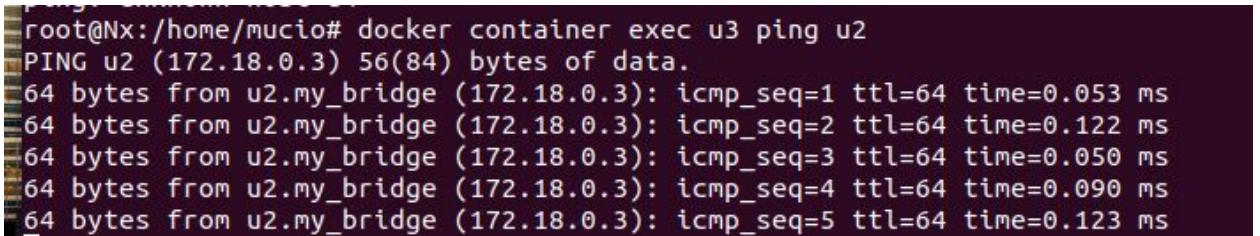
```
$ docker container exec u2 ping u4
```



```
root@Nx:/home/mucio# docker container exec u2 ping u4
ping: unknown host u4
root@Nx:/home/mucio#
```

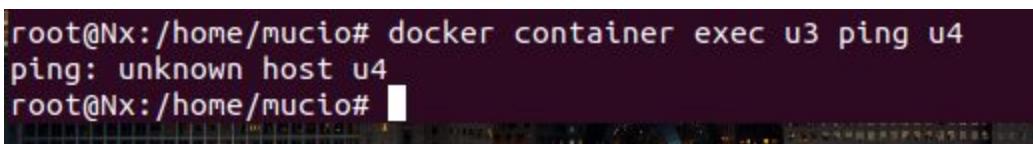
3. Verifique se você pode executar ping no contêiner u2 e u4 de u3

```
$ docker container exec u3 ping u2
```



```
root@Nx:/home/mucio# docker container exec u3 ping u2
PING u2 (172.18.0.3) 56(84) bytes of data.
64 bytes from u2.my_bridge (172.18.0.3): icmp_seq=1 ttl=64 time=0.053 ms
64 bytes from u2.my_bridge (172.18.0.3): icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from u2.my_bridge (172.18.0.3): icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from u2.my_bridge (172.18.0.3): icmp_seq=4 ttl=64 time=0.090 ms
64 bytes from u2.my_bridge (172.18.0.3): icmp_seq=5 ttl=64 time=0.123 ms
```

```
$ docker container exec u3 ping u4
```



```
root@Nx:/home/mucio# docker container exec u3 ping u4
ping: unknown host u4
root@Nx:/home/mucio#
```

4. Nota u1 ainda não pode alcançar u3 e u4:

```
$ docker container exec u1 ping u3  
$ docker container exec u1 ping u4  
  
root@Nx:/home/mucio# docker container exec u1 ping u3  
ping: unknown host u3  
root@Nx:/home/mucio# docker container exec u1 ping u4  
ping: unknown host u4  
root@Nx:/home/mucio#
```

## 15.4 Conclusion

Neste exercício, você explorou os fundamentos da rede de contêineres. A chave é que os contêineres em redes separadas são protegidos por firewall uns dos outros por padrão. Isso deve ser aproveitado o máximo possível para proteger seus aplicativos; Se dois contêineres não precisarem se comunicar, coloque-os em redes separadas.

Você também explorou vários objetos da API:

- docker network ls lista todas as redes no host
- docker network inspect <network name> fornece informações mais detalhadas sobre a rede nomeada
- docker network create --driver <driver> <network name> cria uma nova rede usando o driver especificado; Até agora, só vimos o driver de bridge, para criar uma rede baseada em bridge Linux.
- docker network connect <network name> <container name or id> conecta o contêiner especificado à rede especificada após a execução do contêiner; a flag --network em docker container run alcança o mesmo resultado no lançamento do contêiner.
- docker container inspect <container name or id> gera, entre outras coisas, informações sobre as redes às quais o contêiner especificado está conectado.

## 16 Mapeando Portas de Containeres

### 16.1 Mapeamento de Portas em Tempo de Execução

1. Execute um contêiner nginx sem mapeamentos de portas especiais:

```
$ docker container run -d nginx
```

```
root@Nx:/home/mucio# docker container run -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fc7181108d40: Downloading 228.1kB/22.49MB
d2e987ca2267: Downloading 1.846MB/22.3MB
0b760b431b11: Download complete
```

nginx levanta uma página de destino em <ip>: 80; tente visitar isso no IP do seu host ou do contêiner e ele não estará visível. nenhum tráfego externo pode passar pelo firewalls da brigde do Linux para o contêiner nginx.

2. Agora, execute um contêiner nginx e mapeie a porta 80 no contêiner para a porta 5000 em seu host usando a flag -p; mapeie também a porta 8080 no contêiner para a porta 9000 no host:

```
$ docker container run -d -p 5000:80 -p 9000:8080 nginx
```

Note que a sintaxe é: -p [host-port]:[container-port].

```
root@Nx:/home/mucio# docker container run -d -p 5000:80 -p 9000:8080 nginx
b689e175ff13e520396b9c4b51da167b40783da7753ff98860dd7b7e67e015f3
root@Nx:/home/mucio#
```

3. Verifique os mapeamentos de porta com o comando docker container port:

```
$ docker container port <container id>
```

```
root@Nx:/home/mucio# docker container port b689e175ff13e52
80/tcp -> 0.0.0.0:5000
8080/tcp -> 0.0.0.0:9000
root@Nx:/home/mucio#
```

4. Visite sua página de destino do nginx em <host\_ip>:5000.



## 16.2 Expondo Portas do Dockerfile

1. Além do mapeamento de porta manual, podemos expor algumas portas em um Dockerfile para mapeamento automático de portas inicialização de contêineres. Em um diretório novo, crie um Dockerfile:

```
$FROM nginx  
$EXPOSE 80 8080
```

A screenshot of a terminal window. The title bar says 'root'. The menu bar includes 'File Edit View Search Terminal Help'. The main area contains a Dockerfile with the following content:

```
FROM nginx  
EXPOSE 80 8080
```

2. Construa sua imagem com `docker image build -t my_nginx ..`. Use a flag `-P` ao correr para mapear todas as portas mencionadas na diretiva `EXPOSE`:

```
root@Nx:/home/mucio/novoNginx# docker image build -t my_nginx .  
Sending build context to Docker daemon 2.048kB  
Step 1/2 : FROM nginx  
--> f68d6e55e065  
Step 2/2 : EXPOSE 80 8080  
--> Running in a9690656ab37  
Removing intermediate container a9690656ab37  
--> 418f35613b9f  
Successfully built 418f35613b9f  
Successfully tagged my_nginx:latest  
root@Nx:/home/mucio/novoNginx#
```

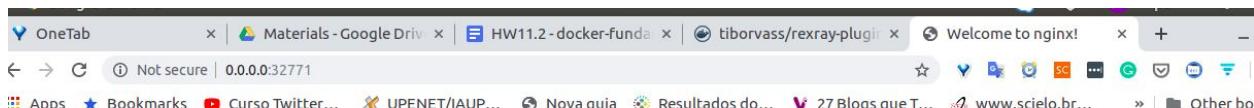
```
$ docker container run -d -P my_nginx
```

```
root@Nx:/home/mucio/novoNginx
File Edit View Search Terminal Help
root@Nx:/home/mucio/novoNginx# docker container run -d -P my_nginx
55c766c7ada7e065c0b5ae97816b69ce34d058c13c97f8eb54d6de5bdb12f06c
root@Nx:/home/mucio/novoNginx#
```

3. Use docker container ls ou docker container port para descobrir quais portas do host foram usadas e visitar sua página de destino do nginx no ip/port apropriado.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
55c766c7ada7	my_nginx	"nginx -g 'daemon off;'	14 seconds ago	Up 13 seconds	0.0.0.0:32771->80/tcp, 0.0.0.0:32770->8080/tcp	glifted_kapitsa
be89e175ff13	nginx	"nginx -g 'daemon off;'	6 minutes ago	Up 6 minutes	0.0.0.0:5000->80/tcp, 0.0.0.0:9000->8080/tcp	romantic_hawking
5b299489bed3	nginx	"nginx -g 'daemon off;'	7 minutes ago	Up 7 minutes	80/tcp	suspicious_goldberg
87a751cd4e43	ubuntu:14.04	"/bin/bash"	18 minutes ago	Up 18 minutes		u3
64f1572d7f09	ubuntu:14.04	"/bin/bash"	25 minutes ago	Up 25 minutes		u2
8ae38780dd3c	ubuntu:14.04	"/bin/bash"	25 minutes ago	Up 25 minutes		u1
27e0478c0525	trainingteam/nginx	"nginx -g 'daemon off;'	2 hours ago	Up 2 hours	0.0.0.0:32769->80/tcp, 0.0.0.0:32768->443/tcp	nginx_server

E o resultado através do navegador.



## Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org). Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

## 16.3 Conclusão

Neste exercício, vimos como mapear explicitamente as portas da pilha de rede de nosso contêiner para as portas de nosso host em tempo de execução com a opção -p para docker container run, ou de forma mais flexível em nosso Dockerfile com EXPOSE, que resultará nas portas listadas em nosso contêiner sendo mapeadas para portas aleatórias disponíveis em nosso host.

## 17 Iniciando um Compose App

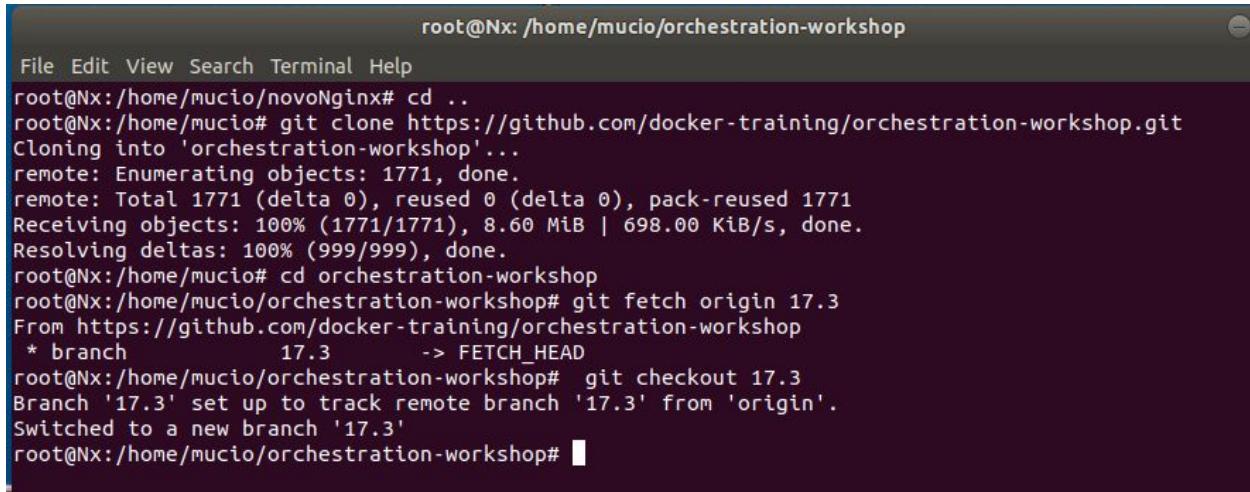
Em um padrão de projeto orientado por microsserviço, o trabalho é dividido entre serviços modulares independentes,

muitos dos quais cooperam para formar uma aplicação completa. Imagens e contêineres do Docker naturalmente permitem esse paradigma usando imagens para definir serviços e contêineres para corresponder às instâncias desses serviços. Para ter sucesso, cada contêiner em execução precisará interagir; O Docker Compose facilita essas interações em um único host. Neste exemplo, exploraremos um exemplo de brinquedo desse aplicativo orquestrado pelo Docker Compose.

## 17.1 Prepare imagens de serviço

1. Baixe o app Dockercoins do github:

```
$ git clone https://github.com/docker-training/orchestration-workshop.git $  
cd orchestration-workshop  
  
$ git fetch origin 17.3  
  
$ git checkout 17.3
```



```
root@Nx:/home/mucio/orchestration-workshop  
File Edit View Search Terminal Help  
root@Nx:/home/mucio/novoNginx# cd ..  
root@Nx:/home/mucio# git clone https://github.com/docker-training/orchestration-workshop.git  
Cloning into 'orchestration-workshop'...  
remote: Enumerating objects: 1771, done.  
remote: Total 1771 (delta 0), reused 0 (delta 0), pack-reused 1771  
Receiving objects: 100% (1771/1771), 8.60 MiB | 698.00 KiB/s, done.  
Resolving deltas: 100% (999/999), done.  
root@Nx:/home/mucio# cd orchestration-workshop  
root@Nx:/home/mucio/orchestration-workshop# git fetch origin 17.3  
From https://github.com/docker-training/orchestration-workshop  
 * branch            17.3      -> FETCH_HEAD  
root@Nx:/home/mucio/orchestration-workshop# git checkout 17.3  
Branch '17.3' set up to track remote branch '17.3' from 'origin'.  
Switched to a new branch '17.3'  
root@Nx:/home/mucio/orchestration-workshop# █
```

Este aplicativo consiste em 5 serviços: um gerador de números aleatórios, um hasher, um backend worker, uma fila de redis e uma interface web. Cada serviço tem uma imagem correspondente, que será construída e enviada para a Docker Store para uso posterior (se você não tiver uma conta do Docker Store, crie uma gratuita primeiro em <https://store.docker.com>).

2. Faça login na sua conta do Docker Store na linha de comando:

```
$ docker login
```

3. Construa e empurre as imagens correspondentes ao rng, hasher, worker e web services. Para hasher1, parece que (da pasta orchestration-workshop/dockercoins 'você acabou de clonar do GitHub'):

Obs: o caminho de onde vc executa o comando, se não retorna erro.

```
$ docker image build -t <username>/dockercoins_hasher:1.0 hasher
```

```
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker image build -t muciojaziel/dockercoins_hasher:1.0 hasher
Sending build context to Docker daemon 3.072kB
Step 1/7 : FROM ruby:alpine
alpine: Pulling from library/ruby
050382585609: Already exists
cb9e14f894ff: Pull complete
78b911433595: Pull complete
abf8325464c1: Downloading 19.9MB/22.11MB
b8311c70979d: Download complete
[
```

```
root@Nx:/home/mucio/orchestration-workshop/dockercoins# ls
docker-compose.yml      docker-compose.yml-logging  rng    worker
docker-compose.yml-images  hasher                  webui
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker image build -t muciojaziel/dockercoins_hasher:1.0 webui
Sending build context to Docker daemon 341kB
Step 1/7 : FROM node:4-slim
4-slim: Pulling from library/node
3d77ce4481b1: Downloading 2.702MB/54.26MB
534514c83d69: Downloading 3.443MB/17.58MB
02d86970c404: Download complete
de288f5266ab: Download complete
85bd110f6aff: Downloading 264.9kB/12.65MB
e8f17c7701dc: Waiting
[
```

```
Removing intermediate container 2d508107c7e9
--> dfde43f16433
Step 7/7 : EXPOSE 80
--> Running in 9a001938fc52
Removing intermediate container 9a001938fc52
--> a1efa4230f41
Successfully built a1efa4230f41
Successfully tagged muciojaziel/dockercoins_hasher:1.0
root@Nx:/home/mucio/orchestration-workshop/dockercoins# [
```

```
$ docker image push <username>/dockercoins_hasher:1.0
```

```
Successfully installed attr 1.7.2
3 gems installed
Removing intermediate container 306b1f94aba2
--> 5645fa92d79c
Step 5/7 : ADD hasher.rb /
--> a15497738ced
Step 6/7 : CMD ["ruby", "hasher.rb"]
--> Running in 2b3d81d7c7e9
Removing intermediate container 2b3d81d7c7e9
--> dfde43f16433
Step 7/7 : EXPOSE 80
--> Running in 9a001938fc52
Removing intermediate container 9a001938fc52
--> a1efa4230f41
Successfully built a1efa4230f41
Successfully tagged muciojaziel/dockercoins_hasher:1.0
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker image push muciojaziel/dockercoins_hasher:1.0
The push refers to repository [docker.io/muciojaziel/dockercoins_hasher]
d3b870418c46: Pushed
be76a692681c: Pushed
b6b45e5c5cc6: Pushed
d73ab5221b27: Pushed
e024e848c970: Mounted from library/ruby
26ed01919d13: Mounted from library/ruby
74f1476b5d78: Mounted from library/ruby
9ecf15f06572: Mounted from library/ruby
1bfeebd65323: Mounted from library/ruby
1.0: digest: sha256:d39f298b96496d0337b542327729a66685456ef8b54f50afe5f652b63b1806e7 size: 2206
```

4. Olhar dentro `docker-compose.yml`, e altere todos os valores de `image` para ter seu ID do Docker Store em vez do usuário; Agora você poderá usar este arquivo Compose para configurar seu aplicativo em qualquer máquina que possa acessar o Docker Store.

```
1.0: digest: sha256:d39f298b96496d0337b542327729a66685456ef8b54f50afe5f652b63b1806e7 size
root@Nx:/home/mucio/orchestration-workshop/dockercoins# ls
docker-compose.yml      docker-compose.yml-logging  rng    worker
docker-compose.yml-images  hasher                  webui
root@Nx:/home/mucio/orchestration-workshop/dockercoins#
```

```
root@Nx: /home/mucio/orchestration-workshop/dockercoins
File Edit View Search Terminal Help
version: "3.1"

services:
  rng:
    image: user/dockercoins_rng:1.0
    networks:
      - dockercoins
    ports:
      - "8001:80"

  hasher:
    image: user/dockercoins_hasher:1.0
    networks:
      - dockercoins
    ports:
      - "8002:80"

  webui:
    image: user/dockercoins_webui:1.0
    networks:
      - dockercoins
    ports:
      - "8000:80"

  redis:
    image: redis
    networks:
      - dockercoins
```

Você terá algo do tipo. e fará isso

```
root@Nx: /home/mucio/orchestration-workshop

File Edit View Search Terminal Help
version: "3.1"

services:
  rng:
    image: muciojaziel/dockercoins_rng:1.0
    networks:
      - dockercoins
    ports:
      - "8001:80"

  hasher:
    image: muciojaziel/dockercoins_hasher:1.0
    networks:
      - dockercoins
    ports:
      - "8002:80"

  webui:
    image: muciojaziel/dockercoins_webui:1.0
    networks:
      - dockercoins
    ports:
      - "8000:80"

  redis:
    image: redis
    networks:
      - dockercoins
```

## 17.2 Inicie o aplicativo

1. Levante o aplicativo (talvez seja necessário instalar primeiro o Docker Compose, se isso não vier pré-instalado em sua máquina; consulte as instruções em <https://docs.docker.com/compose/install/>):

```
$ docker-compose up
```

```

root@Nx: /home/mucio/orchestration-workshop/dockercoins
File Edit View Search Terminal Help
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:04] "GET /32 HTTP/1.1" 200 -
worker_1 | INFO:__main__:5 units of work done, updating hash counter
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:05] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:05] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:05] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:05] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:05] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:05] "GET /32 HTTP/1.1" 200 -
worker_1 | INFO:__main__:5 units of work done, updating hash counter
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:06] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:06] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:06] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:06] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:06] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:07] "GET /32 HTTP/1.1" 200 -
worker_1 | INFO:__main__:5 units of work done, updating hash counter
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:07] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:07] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:07] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:07] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:07] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:08] "GET /32 HTTP/1.1" 200 -
worker_1 | INFO:__main__:5 units of work done, updating hash counter
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:08] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:08] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:08] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:08] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:08] "GET /32 HTTP/1.1" 200 -
rng_1 | 172.19.0.5 - - [12/Jul/2019 14:18:09] "GET /32 HTTP/1.1" 200 -

```

Desafio, execute os passos de 2 a 4

2. Logs de todos os serviços em execução são enviados para STDOUT. Vamos mandar isso para o fundo; mata o aplicativo com `CTRL + C`, enviando um `SIGTERM` para todos os processos em execução; alguma saída imediatamente, enquanto outros aguardam um tempo limite de 10s antes de serem mortos por um `SIGKILL` subsequente. Inicie o aplicativo novamente em segundo plano:

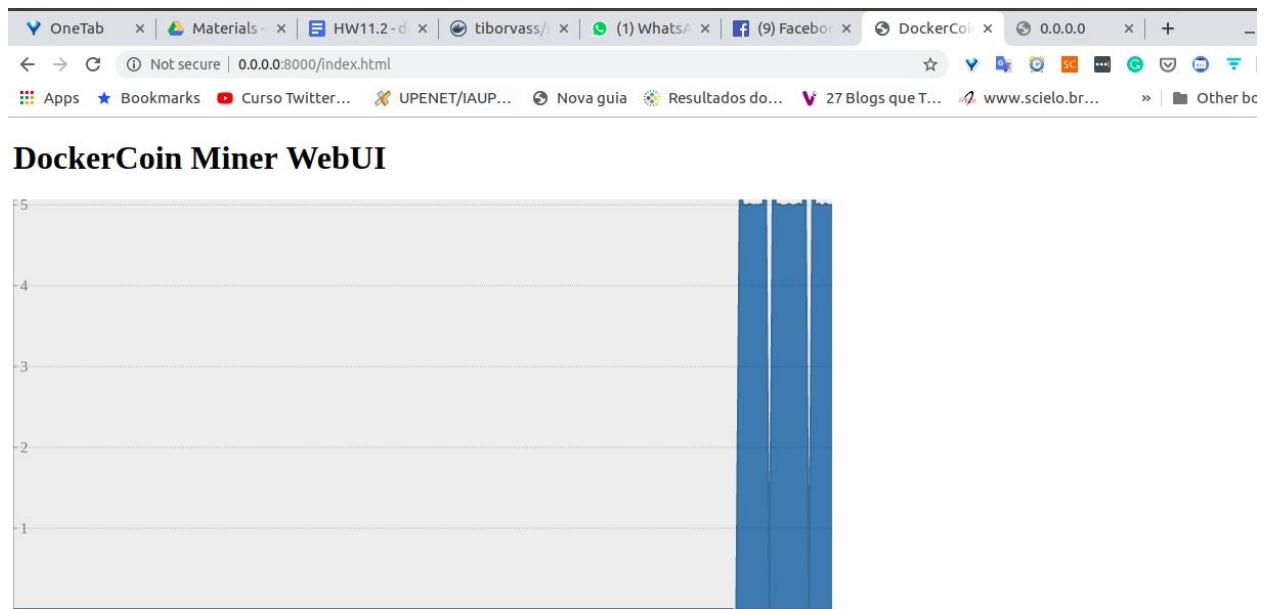
```
$ docker-compose up -d
```

3. Confira quais contêineres estão funcionando graças ao Compose:

```
$ docker-compose ps
```

4. Compare isso com o habitual `docker container ls`; Neste ponto, deve ser o mesmo. Inicie qualquer outro contêiner com `docker container run` e tente os dois comandos `ls` novamente. Você percebe alguma diferença?

5. Com todos os cinco contêineres em funcionamento, visite a interface da Web da Dockercoins em <IP>:8000. Você deve ver um gráfico de velocidade de mineração, em torno de 4 hashes por segundo.



### 17.3 Visualizando Logs

1. Veja os registros de um aplicativo Compose-managed por meio de:

```
$ docker-compose logs
```

Tente o passo 2 e observe a saída.

2. A API de criação de log no Compose segue de perto a API principal de criação de log do Docker. Por exemplo, tente seguir a parte final dos logs da mesma forma que faria com os logs de contêiner regulares:

```
$ docker-compose logs --tail 10 --follow
```

Observe que, ao seguir um log, as opções **CTRL+S** e **CTRL+O** são pausadas e retomam a transmissão ao vivo.

## 17.4 Conclusão

Neste exercício, você viu como iniciar um aplicativo Compose pré-definido e como inspecionar seus registros.

## 18 Dimensionando um aplicativo do Compose

## 18.1 Escalando um serviço

Qualquer serviço definido em nosso `docker-compose.yml` pode ser ampliado a partir da API de composição; nesse contexto, "dimensionamento" significa lançar vários contêineres para o mesmo serviço, que o Docker Compose pode encaminhar solicitações de e para.

1. Escalar o worker app do serviço Dockercoins ter dois workers gerando candidatos a moedas, enquanto verificamos a lista de contêineres em execução antes e depois:

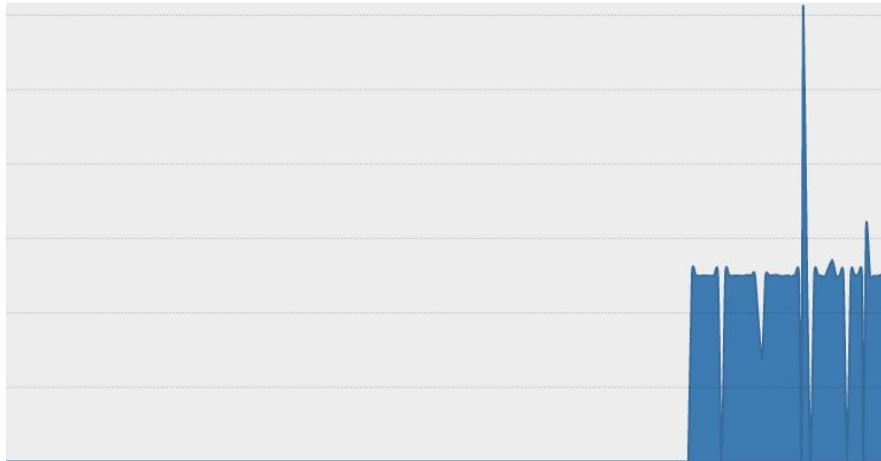
```
$ docker-compose ps
$ docker-compose scale worker=2
$ docker-compose ps

root@Nx:/home/mucio/orchestration-workshop/dockercoins
File Edit View Search Terminal Help
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker-compose ps
      Name          Command     State    Ports
-----+-----+-----+-----+
dockercoins_hasher_1   node webui.js      Exit 137
dockercoins_redis_1    docker-entrypoint.sh redis ...  Exit 0
dockercoins_rng_1      python rng.py      Exit 137
dockercoins_webui_1    node webui.js      Exit 137
dockercoins_worker_1   python worker.py  Exit 137
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker-compose scale worker=2
WARNING: The scale command is deprecated. Use the up command with the --scale flag instead.
Starting dockercoins_worker_1 ... done
Creating dockercoins_worker_2 ... done
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker-compose ps
      Name          Command     State    Ports
-----+-----+-----+-----+
dockercoins_hasher_1   node webui.js      Exit 137
dockercoins_redis_1    docker-entrypoint.sh redis ...  Exit 0
dockercoins_rng_1      python rng.py      Exit 137
dockercoins_webui_1    node webui.js      Exit 137
dockercoins_worker_1   python worker.py  Up
dockercoins_worker_2   python worker.py  Up
root@Nx:/home/mucio/orchestration-workshop/dockercoins# █
```

2. Veja o gráfico de desempenho fornecido pela interface web; a taxa de mineração de moedas deveria ter dobrado. Verifique também os logs usando a API de registro que aprendemos no último exercício; você deve ver um segundo relatório de instância do worker.

→ C ⓘ Not secure | 0.0.0.0:8000/index.html  
Apps ⚡ Bookmarks 🎵 Curso Twitter... 🔧 UPENET/IAUP... 🌐 Nova guia 🌐 Resultados do... 💡 27 Blog

## DockerCoin Miner WebUI



urrent mining speed: ~5.0 hashes/second ([Tweet this!](#))

## 18.2 Não-linearidade de escala

1. Tente executar `top` para inspecionar o uso de recursos do sistema; ainda deve ser razoavelmente n-elegível. Então, continue ampliando seus workers:

```
$ docker-compose scale worker=10  
$ docker-compose ps
```

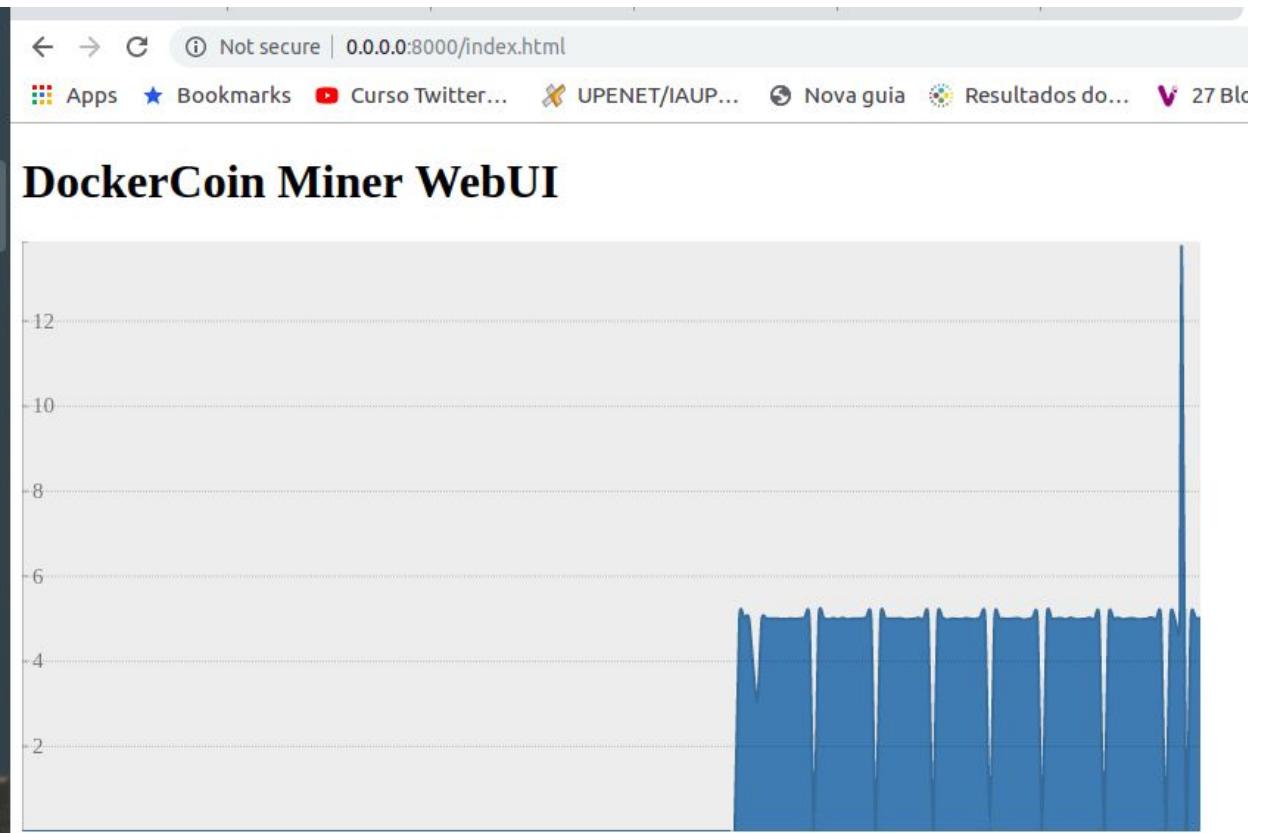
```

root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker-compose scale worker=10
WARNING: The scale command is deprecated. Use the up command with the --scale flag instead.
Starting dockercoins_worker_1 ... done
Creating dockercoins_worker_2 ... done
Creating dockercoins_worker_3 ... done
Creating dockercoins_worker_4 ... done
Creating dockercoins_worker_5 ... done
Creating dockercoins_worker_6 ... done
Creating dockercoins_worker_7 ... done
Creating dockercoins_worker_8 ... done
Creating dockercoins_worker_9 ... done
Creating dockercoins_worker_10 ... done
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker-compose ps
          Name            Command       State    Ports
-----  

dockercoins_hasher_1   node webui.js      Exit 137
dockercoins_redis_1    docker-entrypoint.sh redis ...
dockercoins_rng_1      python rng.py      Exit 137
dockercoins_webui_1    node webui.js      Exit 137
dockercoins_worker_1   python worker.py   Up
dockercoins_worker_10  python worker.py   Up
dockercoins_worker_2   python worker.py   Up
dockercoins_worker_3   python worker.py   Up
dockercoins_worker_4   python worker.py   Up
dockercoins_worker_5   python worker.py   Up
dockercoins_worker_6   python worker.py   Up
dockercoins_worker_7   python worker.py   Up
dockercoins_worker_8   python worker.py   Up
dockercoins_worker_9   python worker.py   Up
root@Nx:/home/mucio/orchestration-workshop/dockercoins#

```

2. Verifique sua interface web novamente; vai de 2 a 10 workers desde um aumento de desempenho de 5x? Parece que alguma coisa está afundando nosso aplicativo; qualquer aplicativo distribuído, como o Dockercoins, precisa de ferramentas para entender onde estão os gargalos, de modo que o aplicativo possa ser dimensionado de maneira inteligente.



3. Olhar dentro docker-compose.yml no rng e nos serviços hashes; eles estão expostos nas portas do host 8001 e 8002, para que possamos usar o httping para investigar sua latência:

Obs: Talvez seja necessário instalar httping [ use \$apt-get install httping]

```
$ httping -c 10 localhost:8001
```

```
root@Nx:/home/mucio/orchestration-workshop/dockercoins# httping -c 10 localhost:8001
PING localhost:8001 (/):
connected to 127.0.0.1:8001 (158 bytes), seq=0 time= 2,87 ms
connected to 127.0.0.1:8001 (158 bytes), seq=1 time= 2,88 ms
connected to 127.0.0.1:8001 (158 bytes), seq=2 time= 5,41 ms
connected to 127.0.0.1:8001 (158 bytes), seq=3 time= 3,49 ms
connected to 127.0.0.1:8001 (158 bytes), seq=4 time= 6,47 ms
connected to 127.0.0.1:8001 (158 bytes), seq=5 time= 2,47 ms
```

Tente agora e observe:

```
$ httping -c 10 localhost:8002
```

rng na porta 8001 tem a latência muito maior, sugerindo que pode ser o nosso gargalo. Um gerador de números aleatórios com base na entropia não melhorará, iniciando mais instâncias na mesma máquina. precisaremos de uma maneira de trazer mais nós para nosso aplicativo, além disso, vamos explorá-lo na próxima unidade do

Docker Swarm.

4. Por enquanto, desligue seu aplicativo:

```
$ docker-compose down
```

### 18.3 Conclusão

Neste exercício, vimos como ampliar um serviço definido em nosso aplicativo Compose usando o objeto da API de scale. Além disso, vimos como é crucial ter monitoramento e ferramentas detalhados em um aplicativo orientado a microsserviços, a fim de identificar corretamente os gargalos e aproveitar a simplicidade do dimensionamento com o Docker.

```
root@Nx:/home/mucio/orchestration-workshop/dockercoins# docker-compose down
Stopping dockercoins_webui_1 ... done
Stopping dockercoins_hasher_1 ... done
Stopping dockercoins_rng_1 ... done
Stopping dockercoins_worker_1 ... done
Stopping dockercoins_redis_1 ... done
Removing dockercoins_webui_1 ... done
Removing dockercoins_hasher_1 ... done
Removing dockercoins_rng_1 ... done
Removing dockercoins_worker_1 ... done
Removing dockercoins_redis_1 ... done
Removing network dockercoins_dockercoins
root@Nx:/home/mucio/orchestration-workshop/dockercoins# 
```