

Laboratório de Docker

Conteúdo será cobrado na prova

Laboratório do docker

- Muitos recursos online, em ingles
 - <https://github.com/delner/docker-training/tree/master>
 - Em casa, pode tambem usar o <https://training.play-with-docker.com/>

Docker na vm, inicial NGIX/Interativo

- Instalação do docker conform abaixo

- Para instalar o docker

sudo apt update -y

sudo apt upgrade -y

sudo apt install docker.io

- Verificar se o serviço está no ar

sudo systemctl status docker



```
File Edit View Search Terminal Help
prompt] [-T timeout] [-u user] file ...
rivaldo@mintvm:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-06-21 17:51:50 -03; 9min ago
     TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
    Main PID: 63199 (dockerd)
       Tasks: 11
      Memory: 119.1M
        CGroup: /system.slice/docker.service
                └─63199 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Jun 21 17:51:49 mintvm dockerd[63199]: time="2023-06-21T17:51:49.868169546-03:00"
Jun 21 17:51:49 mintvm dockerd[63199]: time="2023-06-21T17:51:49.938156155-03:00"
Jun 21 17:51:49 mintvm dockerd[63199]: time="2023-06-21T17:51:49.982728735-03:00"
Jun 21 17:51:49 mintvm dockerd[63199]: time="2023-06-21T17:51:49.998193005-03:00"
Jun 21 17:51:49 mintvm dockerd[63199]: time="2023-06-21T17:51:49.998269831-03:00"
Jun 21 17:51:50 mintvm systemd[1]: Started Docker Application Container Engine.
Jun 21 17:51:50 mintvm dockerd[63199]: time="2023-06-21T17:51:50.019353064-03:00"
Jun 21 17:55:29 mintvm dockerd[63199]: time="2023-06-21T17:55:29.022987942-03:00"
Jun 21 17:55:29 mintvm dockerd[63199]: time="2023-06-21T17:55:29.152943229-03:00"
Jun 21 17:55:47 mintvm dockerd[63199]: time="2023-06-21T17:55:47.705404650-03:00"
lines 1-21/21 (END)
```

Docker na vm

- Testar
 - **`$sudo docker run hello-world`**

Este comando vai buscar uma imagem de teste e executa esta imagem.

*Vamos entender a diferença entre
container e imagem*

`$sudo docker images`

→ lista as imagens, veja que tem o hello-world

→ a imagem é o que vai rodar em um container, seria o equivalente ao “iso” do Sistema operacional quando instalamos no virtualbox, equivalente, mas não é exatamente isso, a imagem é mais que o iso pois terá tudo que eu quero rodar, e ao mesmo tempo é menos, pois tem SOMENTE o que eu preciso para rodar minha aplicação e não mais que isso

`$sudo docker ps`

→ lista os containers

→ o container é equivalente a uma VM (maquina virtual), é onde eu executo as imagens

→ o hello-world executa e termina, não fica “rodando”

```
ricar@ANIETE C:\Users\ricar
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:c2e23624975516c7e27b1b25be3682a8c6c4c0cea011b791ce98aa423b5040a0
Status: Downloaded newer image for hello-world:latest
```

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

To generate this message, Docker took the following steps:

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

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

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

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

```
ubuntu@instance-20220211-1908:~$ sudo docker images
REPOSITORY      TAG         IMAGE ID      CREATED        SIZE
hello-world     latest     9c7a54a9a43c  7 weeks ago   13.3kB
ubuntu@instance-20220211-1908:~$
```

```
ubuntu@instance-20220211-1908:~$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
ubuntu@instance-20220211-1908:~$
```

Docker na vm, inicial NGIX/Interativo

- Vamos criar uma imagem docker utilizando uma imagem pronta do ubuntu. Este exercício trabalho com o conceito de imagem pronta, modificação de imagem e commit para salvar as alterações feitas.
- Siga o roteiro abaixo a partir de **Create a Docker Instance**
- **Original em**
 - <https://www.letscloud.io/community/how-to-launch-a-docker-container-with-an-interactive-shell>
 - https://www.letscloud-io.translate.goog/community/how-to-launch-a-docker-container-with-an-interactive-shell? x_tr_sl=en& x_tr_tl=pt
- Nas paginas a seguir esta o que deve ser feito, ou olhe na referência acima.

Criar uma instância do Docker

Primeiro, você precisará extrair a imagem do Ubuntu do Docker Hub antes de iniciar qualquer coisa.

Você pode facilmente extrair a imagem do Ubuntu do registro público do Docker com o seguinte comando.

```
$ sudo docker pull ubuntu
```

Depois que a imagem do Ubuntu for baixada, você deverá ver a seguinte saída:

```
Using default tag: latest
latest: Pulling from library/ubuntu
6adf03819f3e: Pull complete
0573fgd463211: Pull complete
0du67c50as3e: Pull complete
Digest: sha256:f08sjajsd75ldj90065187e7eabdfac3c96e5ff0f6b2fs2k4ddi84hjdd
Status: Downloaded newer image for ubuntu:latest
```

Em seguida, crie uma instância do Ubuntu em um contêiner do Docker e anexe um bash shell executando o seguinte comando:

```
$ sudo docker run -i -t ubuntu bash
```

Você deve ver a seguinte saída:

```
root@13246cfc6417: /#
```

Instale o Nginx no Container em execução

Agora, depois que a imagem base do Ubuntu com a instância estiver pronta, você pode facilmente instalar o Nginx Server interativamente para ela. Para fazer isso, você precisará executar o seguinte comando em um terminal.

```
$ sudo apt-get update -y
$ sudo apt-get install nginx -y
```

Após a conclusão da instalação, você pode sair do shell atual executando o seguinte comando:

```
$ exit
```

Em seguida, você pode salvar as alterações feitas na instância do Ubuntu. Para fazer isso, primeiro você precisará do Container ID da instância do Ubuntu em execução. Para obter isso, execute:

```
$ sudo docker ps -a
```

Você deve ver a seguinte saída:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
13246cfc6417	ubuntu	"bash"	3 minutes ago	Exited (0) 8 seco
ff2deb4f97b1	ubuntu	"/bin/bash"	30 hours ago	Exited (0) 30 hou



Agora, salve as alterações como uma nova imagem com o nome ubuntu-nginx executando o seguinte comando:

```
$ sudo docker commit 13246cfc6417 ubuntu-nginx
```

A saída se parece com isto:

```
sha256:3d880d3d819eaf383f454da974602878a4354f99cd5eeb30a4c89b5cfas3485ds
```

Você pode ver que as alterações são salvas usando o ID do contêiner e o nome da imagem ubuntu-nginx. Para verificar se a nova imagem está em execução ou não, execute:

```
$ sudo docker images
```

Você pode ver a imagem ubuntu-nginx listada abaixo:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu-nginx	latest	7b280d3d819e	18 seconds ago	191MB
ubuntu	latest	7698f282e524	2 weeks ago	69.9MB

Perceba que temos duas imagens, a original “ubuntu” e a alterada com o nginx “ubuntu-nginx”

Adicione o conteúdo do site à imagem ubuntu-nginx

Agora, você tem uma nova imagem que contém um servidor Web Nginx. Em seguida, você precisará criar um Dockerfile com base nessa imagem e adicionar os arquivos necessários. Dado o caminho relativo para um tarball do conteúdo do site. O Docker descompacta ou descompacta automaticamente os arquivos em um arquivo tar ou zip de origem no diretório de destino.

Para fazer isso, você precisa criar um `index.html` arquivo no sistema host e adicioná-lo a um tarball chamado `testsite.tar` no diretório atual:

```
$ mkdir -p docker/testsite  
$ cd docker
```

```
$ nano testsite/index.html
```

adicione o seguinte conteúdo:

```
testsite/index.html  
  
<html> Este é meu primeiro servidor Web Nginx </htm>
```

Salve e feche o arquivo

Para copiar (ctrl-c)

<html> Este é meu primeiro servidor Web Nginx </html>

Agora, comprima o diretório do site usando tar:

Para copiar (ctrl-c)

```
$ tar -cvf testsite.tar testsite
```

```
tar -cvf testsite.tar testsite
```

Agora, crie o Dockerfile para adicionar o conteúdo do site à imagem ubuntu-nginx e inicie o Nginx na porta 80:

```
$ nano dockerfile
```

Adicione o seguinte conteúdo

dockerfile

```
FROM ubuntu-nginx
ADD testsite.tar /tmp/
RUN mv /tmp/testsite/* /var/www/html/
EXPOSE 80
CMD ["/usr/sbin/nginx" "-g" "daemon off;"]
```

```
FROM ubuntu-nginx
ADD testsite.tar /tmp/
RUN mv /tmp/testsite/* /var/www/html/
EXPOSE 80
CMD ["/usr/sbin/nginx" "-g" "daemon off;"]
```

Salve e feche o arquivo.

No dockerfile acima, o conteúdo do site `testsite.tar` será extraído automaticamente para `/tmp/` a pasta. Em seguida, todo o site será movido para o diretório raiz do Nginx `/var/www/html/` e a exposição `80` abrirá a porta `80` para que o site fique disponível normalmente. Em seguida, o ponto de entrada é definido `/usr/sbin/nginx` para que o servidor Nginx seja executado.

Criar e executar contêiner usando Dockerfile

Agora, você pode criar um Container usando o Dockerfile que acabou de criar para adicionar o site nele.

Para fazer isso, execute o seguinte comando:

```
$ cd docker
$ sudo docker build -t testsite .
```

Você deve ver a seguinte saída:

```
Sending build context to Docker daemon 14.34kB
Step 1/6 : FROM ubuntu-nginx
--> 7b280d3d819e
Step 2/6 : ADD testsite.tar /tmp/
--> 697007c7e514
Step 3/6 : RUN mv /tmp/testsite/* /var/www/html/
--> Running in 68097dd4b53b
Removing intermediate container 68097dd4b53b
--> 0630c6e738a8
Step 4/6 : EXPOSE 80
--> Running in 00cc1a93ae16
Removing intermediate container 00cc1a93ae16
--> 24efe8462fba
Step 5/6 : ENTRYPOINT [ "/usr/sbin/nginx" ]
--> Running in 19f560394a01
Removing intermediate container 19f560394a01
--> Running in 564d4b993f07
Removing intermediate container 564d4b993f07
--> e964ce72b333
Successfully built e964ce72b333
Successfully tagged testsite:latest
```

Depois que a imagem foi construída, agora você pode prosseguir criando um contêiner executando a instância Nginx nele.

```
$ sudo docker run -d -P testsite
```

Agora, use o `docker ps` comando para determinar a porta ativada e, em seguida, use curl para inspecionar o conteúdo da amostra.

```
$ sudo docker ps
```

Você deve ver a seguinte saída:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
216063fc335d	testsite	"/usr/sbin/nginx..."	19 seconds ago	Up 14 seconds	0.0.0.0:32768->80/tcp	pensive_cori

O que é o names ?

Agora, verifique seu servidor Web Nginx executando o seguinte comando:

```
$ curl localhost:32768
```

ou

```
$ curl "Container IP Address":80
```

Você deve ver a página da Web Nginx que você criou anteriormente:

```
This is my first Nginx Web Server
```

Conclusão

Parabéns! você hospedou com sucesso o site da Web usando o servidor da Web Nginx no contêiner Docker usando um shell interativo. Espero que agora você possa instalar facilmente qualquer coisa dentro do contêiner Docker com um shell interativo.

Docker é transiente, nada é preservado no container, somente a imagem

- Por isso, para permitir “salvar” coisas precisamos criar volumes, vamos fazer um exercício com o apache e volumes conforme explicado em

<https://github.com/delner/docker-training/tree/master/exercises/basic/5-volumes>

e traduzido

<https://github-com.translate.goog/delner/docker-training/tree/master/exercises/basic/5-volumes? x tr sl=en& x tr tl=pt>

Exercício 5: Volumes

Neste exercício, aprenderemos a trabalhar com volumes do Docker para manter dados entre contêineres.

Para fazer isso, configuraremos um servidor web Apache HTTPD e manteremos alguns arquivos HTML em um volume.

Configurando o servidor

Para executar nosso servidor Apache HTTPD, execute este comando:

```
$ docker run --rm -d --name apache -p 80:80 httpd:2.4  
d87e0a193dde5652ac762d8849983c2cadb5116b80c8a61a4180e350d678b4d2
```

Este commando vai buscar a imagem e executar

Este comando iniciará um novo contêiner a partir do HTTP 2.4, nomeá-lo `apache`, vinculará a porta `80` à máquina host (mais sobre isso posteriormente) e definirá um sinalizador para excluir o contêiner quando ele parar.

Depois de iniciado, podemos executar `curl localhost` a consulta ao servidor da web para a página padrão:

```
$ curl localhost  
<html><body><h1>It works!</h1></body></html>  
$
```


Este é o arquivo padrão `index.html` incluído em uma nova instalação do Apache 2.4. Vamos substituir este arquivo HTML por um novo conteúdo.

Para isso, usaremos o `docker cp` comando, semelhante a `scp`, que copia os arquivos entre o host e os containers. Vamos fornecer o `index.html` arquivo do diretório em que este README está:

```
$ docker cp index.html apache:/usr/local/apache2/htdocs/  
$
```

O primeiro caminho é o caminho de origem, representando nosso novo arquivo em nossa máquina host, e o segundo caminho, nosso destino. `apache` é o nome do contêiner para o qual queremos copiar e `/usr/local/apache2/htdocs/` é de onde o servidor web serve o HTML.

Executar `curl` novamente agora parece um pouco diferente:

```
$ curl localhost  
<html><body><h1>It works in Docker!</h1></body></html>  
$
```

Um possível problema de dados

Esse contêiner, durante sua vida útil, continuará a servir nosso novo arquivo HTML.

No entanto, os contêineres no Docker são, na prática, considerados efêmeros. Eles podem morrer inesperadamente e, em certas implantações, ser removidos sem aviso prévio. Se você estiver dependendo do estado do contêiner para seu aplicativo, poderá perder dados importantes quando esses contêineres morrerem. Isso é especialmente preocupante para aplicativos como bancos de dados, que devem ser considerados armazenamentos de dados permanentes.

No caso do nosso servidor HTTPD, simplesmente parar o contêiner fará com que ele seja removido automaticamente. Podemos trazer outro container de volta em seu lugar, mas ele não terá mais nossas alterações.

```
$ docker stop apache
apache
$ docker run --rm -d --name apache -p 80:80 httpd:2.4
9bd0620e3d8464456c368b1fe9b82733282d980a7c3f854b8cba7726f0a02958
$ curl localhost
<html><body><h1>It works!</h1></body></html>
$
```

[Sem título]

Para preservar nossos dados entre interrupções ou atualizações do sistema, podemos usar volumes para persistir nossos dados em gerações de contêineres.

Gerenciando volumes

Os volumes no Docker são armazenamentos de arquivos, que ficam independentemente dos contêineres do Docker. A função como os volumes EBS da Amazon Web Services e outras mídias montáveis, como pen drives USB. Eles podem ser criados, excluídos e montados em contêineres em locais específicos dentro de uma imagem, como você faria com `mount` o comando no Linux.

Para listar seus volumes, execute `docker volume ls` :

```
$ docker volume ls
DRIVER          VOLUME NAME
$
```

Para criar um novo volume, execute `docker volume create` e dê a ele um nome de volume.

```
$ docker volume create myvolume
myvolume
$ docker volume ls
DRIVER          VOLUME NAME
local          myvolume
$
```

Para remover um volume, execute `docker volume rm` e dê a ele o nome do volume.

```
$ docker volume rm myvolume
myvolume
$ docker volume ls
DRIVER          VOLUME NAME
$
```

Montando volumes em contêineres

Primeiro crie um novo volume chamado `httpd_htdocs` :

```
$ docker volume create httpd_htdocs
httpd_htdocs
$
```

Em seguida, execute novamente nosso `docker run` comando, fornecendo o `-v` sinalizador de montagem.

```
$ docker run --rm -d --name apache -p 80:80 -v httpd_htdocs:/usr/local/apache2/htdocs/ httpd:2.4
c21dd93fea83d710b4d4c954911862760030723df6a5b42650e462e388fe6049
$
```

E copie novamente em nosso arquivo HTML modificado.

```
$ docker cp index.html apache:/usr/local/apache2/htdocs/
$
```

E corra `curl` para verificar se funcionou.

```
$ curl localhost
<html><body><h1>It works in Docker!</h1></body></html>
$
```

Agora, para ver o volume em ação, vamos parar o contêiner. Ao fornecer o `--rm` sinalizador durante `run`, ele deve remover o contêiner ao parar.

```
$ docker stop apache
apache
$
```

Então, mais uma vez, inicie o httpd com o mesmo comando de execução da última vez. Desta vez, no entanto, podemos `curl` ver que as alterações de nosso arquivo ainda estão lá de antes.

```
$ docker run --rm -d --name apache -p 80:80 -v httpd_htdocs:/usr/local/apache2/htdocs/ httpd:2.4
c21dd93fea83d710b4d4c954911862760030723df6a5b42650e462e388fe6049
$ curl localhost
<html><body><h1>It works in Docker!</h1></body></html>
$
```

Podemos pegar esse volume e montá-lo em qualquer contêiner HTTPD agora, o que nos dá flexibilidade para trocar nosso contêiner por versões mais recentes sem perder nossos dados, se desejarmos.

Vá em frente e corra `docker stop apache` para parar e remover o recipiente e, em seguida, `docker volume rm httpd_htdocs` remover o volume.

Montando diretórios de host em contêineres

Como alternativa ao uso de volumes, se você tiver um diretório em sua máquina host que gostaria de usar como um volume, também poderá montá-lo. Essa técnica é útil em ambientes de desenvolvimento, nos quais você pode querer montar seu repositório local em uma imagem do Docker e modificar ativamente o conteúdo de um contêiner do Docker sem reconstruir ou copiar arquivos para ele.

A `-v` bandeira para fazer isso é quase idêntica à anterior. Basta especificar um caminho absoluto para um diretório local. No nosso caso, passaremos `.` a especificar o `5-volumes` diretório neste repositório, que convenientemente contém uma versão modificada do arquivo HTML.

```
$ pwd
/home/david/src/docker-training/exercises/basic/5-volumes/
$ docker run --rm -d --name apache -p 80:80 -v /home/david/src/docker-training/exercises/basic/5-volumes/:/usr/local/apache2/htdocs/
0d91516b20ea6113b5dcca08ada6465095dc68663b3d2201dc0490165764f842
$ curl localhost
<html><body><h1>It works in Docker!</h1></body></html>
$
```

Com a montagem do diretório do host no lugar, modifique o `index.html` arquivo neste diretório com qualquer mensagem que desejar, salve o arquivo e execute novamente `curl`.

```
$ curl localhost
<html><body><h1>It works quite well in Docker!</h1></body></html>
$
```

Você pode ver as alterações de arquivo ocorrendo imediatamente no contêiner do Docker, sem a necessidade de execução `docker cp`.

Vá em frente e corra `docker stop apache` para parar e remover o contêiner.

Lembrando, conteudo de prova

- Caso tenham ficado com dúvidas, perguntem durante a aula, ou na próxima.
- Para fazer em casa
 - <https://github.com/delner/docker-training/blob/master/exercises/basic/6-networking/README.md>
 - É importante entender a rede do docker , porém este não cobrarei na prova