APRENDENDO DOCKER

Autor: Eng.º Fabrício de Lima Ribeiro
Data: 12/01/2023

Índice

- INSTALANDO O DOCKER

O Docker só pode ser instalados em sistemas x64 com o kernell do linux maior que a versão 3.8.

- Para saber se o sistema é 32 ou 64:
$ uname -m

- Para saber a versão do kernell:
$ uname -r

-Instalando:

Entre no modo e na pasta root e digite:
 # curl -fsSL https://get.docker.com/ | sh

Este comando também é usado para atualizar o docker sistema.

obs: caso o curl não esteja instalador, intalar através do comando:

 # apt-get install curl

- Se não quiser usar o docker no modo root, basta adicionar o usuário no grupo docker:
$ sudo usermod -aG docker your-user

- Para saber a versão do docker:
 # docker --version

- Caso o docker não esteja estartado, digite:
 # /etc/init.d/docker start

ou

 # service docker start

- INSTALANDO O DOCKER EM UM RASPBERRY PI

Para instalar o docker em um raspberry pi, siga as seguintes linhas de comando:

sudo apt update
sudo apt upgrade
sudo apt install raspberrypi-kernel raspberrypi-kernel-headers
curl -sSL https://get.docker.com | sh
sudo usermod -aG docker pi
sudo reboot
docker --version

"pi" nome de usuário

- INSTALANDO O DOCKER-COMPOSE

sudo apt-get install libffi-dev libssl-dev
sudo apt install python3-dev
sudo apt-get install -y python3 python3-pip
sudo pip3 install docker-compose
sudo systemctl enable docker

- TRABALHANDO COM IMAGENS

- Para listar todos as imagens baixadas e salvas no computador:
 # docker images

```
root@desktop:/home/fabricio# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
alpine       latest    49176f190c7e   6 weeks ago   7.05MB
root@desktop:/home/fabricio#
```

Neste exemplo podemos observar que temos a imagem da distribuição Alpine baixada e salva no computador.

- Para procurar uma imagem:
 # docker search “nome da imagem”

Exemplo:
 # docker search ubuntu

```
root@desktop:/home/fabricio# docker search ubuntu
NAME                            DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu                          Ubuntu is a Debian-based Linux operating sys...   15419     [OK]
websphere-liberty               WebSphere Liberty multi-architecture images ...   291       [OK]
ubuntu-upstart                  DEPRECATED, as is Upstart (find other proces...   112       [OK]
neurodebian                     NeuroDebian provides neuroscience research s...   97        [OK]
ubuntu/nginx                    Nginx, a high-performance reverse proxy & we...   73
open-liberty                    Open Liberty multi-architecture images based...   56        [OK]
ubuntu/apache2                  Apache, a secure & extensible open-source HT...   51
ubuntu-debootstrap              DEPRECATED; use "ubuntu" instead                  50        [OK]
ubuntu/squid                    Squid is a caching proxy for the Web. Long-t...   47
ubuntu/mysql                    MySQL open source fast, stable, multi-thread...   41
ubuntu/bind9                    BIND 9 is a very flexible, full-featured DNS...   36
ubuntu/prometheus               Prometheus is a systems and service monitori...   33
ubuntu/postgres                 PostgreSQL is an open source object-relation...   22
ubuntu/kafka                    Apache Kafka, a distributed event streaming ...   19
ubuntu/redis                    Redis, an open source key-value store. Long-...   16
ubuntu/prometheus-alertmanager  Alertmanager handles client alerts from Prom...   8
ubuntu/grafana                  Grafana, a feature rich metrics dashboard & ...   6
ubuntu/memcached                Memcached, in-memory keyvalue store for smal...   5
ubuntu/zookeeper                ZooKeeper maintains configuration informatio...   5
ubuntu/dotnet-runtime           Chiselled Ubuntu runtime image for .NET apps...   5
ubuntu/dotnet-deps              Chiselled Ubuntu for self-contained .NET & A...   5
ubuntu/telegraf                 Telegraf collects, processes, aggregates & w...   4
ubuntu/cortex                   Cortex provides storage for Prometheus. Long...   3
ubuntu/dotnet-aspnet            Chiselled Ubuntu runtime image for ASP.NET a...   3
ubuntu/cassandra                Cassandra, an open source NoSQL distributed ...   2
root@desktop:/home/fabricio#
```

Podemos perceber uma quantidade grande de imagens, as marcadas com [OK] são oficiais. Podemos também procurar uma imagem com uma versão específica:

Exemplo:
 # docker search ubuntu:14.04

```
root@desktop:/home/fabricio# docker search ubuntu:14.04
NAME                                DESCRIPTION                                     STARS   OFFICIAL   AUTOMATED
edse/ubuntu-nginx-mysql-php-nodejs   Docker container built from Ubuntu:14.04 wit...   15                 [OK]
blacktobacco/ajenti                 Ajenti web hosting panel server based on Ubu...   7                  [OK]
leslau/nxlog                        nxlog ce image base on ubuntu:14.04               2                  [OK]
huangchaosuper/devops               ubuntu:14.04 logstash:1.4.2 zabbix-agent:2.2...   1
mikefaille/etcd                     etcd image for docker build on nuagebec/ubun...   1                  [OK]
jorgeluisrmx/ros-indigo             ROS Indigo on Ubuntu:14.04                        1                  [OK]
takeharu/ubuntu-mysql               MySQL on Ubuntu:14.04                             1                  [OK]
jorgeluisrmx/ubuntu-dev-base        Ubuntu:14.04 + C/C++ Python  dev libs image      1                  [OK]
bohanzhang/ubuntu_cn                国内用户的 Ubuntu Dockerfile 基于 ubuntu:14.04     1                  [OK]
pkubicki/atom-rc-base               Open SSH with SASS and RSYNC installed on Ub...   1
nuagebec/etcd                       etcd image for docker build on nuagebec/ubun...   1                  [OK]
jcirizar/devimg                     Ubuntu:14.04 with NVM and more.                   1                  [OK]
mrhub/snort                         snort on ubuntu:14.04                             0                  [OK]
vik733/haproxy-consul_template      Container on Ubuntu:14.04 with haproxy and c...   0
slyn/mongodb                        mongodb - ubuntu:14.04                            0                  [OK]
pranay/sinatra                      FROM ubuntu:14.04 RUN apt-get update && apt-...   0
savaki/dynamodb                     DynamoDB using ubuntu:14.04 and oracle-java8      0                  [OK]
letterer/swift-docker               ubuntu:14.04 with Apple swift                     0                  [OK]
zeerdonker/docker-oracle-java       ubuntu:14.04 based oracle-java 8 build            0                  [OK]
bdkdevorg/bm.docker.varnish         ubuntu:14.04, varnish, curl, supervisor           0                  [OK]
jorgeluisrmx/gazebo5                Gazebo5 on Ubuntu:14.04                           0                  [OK]
meedan/base                         meedan base image FROM ubuntu:14.04               0                  [OK]
rpaliwal/golang                     golang repo based on ubuntu:14.04 instead of...   0
hieisky/ubuntu-base                 To build a ubuntu:14.04.4 base image.             0                  [OK]
devorbitus/ubuntu-bash-jq-curl      Based on Ubuntu:14.04 and only adding curl a...   0
root@desktop:/home/fabricio#
```

- Para baixar a imagem selecionada:
 # docker pull ubuntu:14.04

```
root@desktop:/home/fabricio# docker pull ubuntu:14.04
14.04: Pulling from library/ubuntu
2e6e20c8e2e6: Pull complete
0551a797c01d: Pull complete
512123a864da: Pull complete
Digest: sha256:64483f3496c1373bfd55348e88694d1c4d0c9b660dee6bfef5e12f43b9933b30
Status: Downloaded newer image for ubuntu:14.04
docker.io/library/ubuntu:14.04
root@desktop:/home/fabricio#
```

- Para verificar se a imagem foi salva:
 # docker images

```
root@desktop:/home/fabricio# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
alpine       latest    49176f190c7e   6 weeks ago     7.05MB
ubuntu       14.04     13b66b487594   21 months ago   197MB
root@desktop:/home/fabricio#
```

- Para remover uma imagem:
 # docker rmi “nome da imagem”

Exemplo:

```
root@desktop:/home/fabricio# docker images
```

```
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
alpine       latest    49176f190c7e   6 weeks ago     7.05MB
ubuntu       14.04     13b66b487594   21 months ago   197MB
ubuntu       10.04     e21dbcc7c9de   8 years ago     183MB
root@desktop:/home/fabricio#
```

Note que temos 3 imagens salvas: ***alpine:latest, ubuntu:14.04 e ubuntu:10.04***. Vamos remover a imagem do ***ubuntu:10.04***:

 # docker rmi ubuntu:10.04

```
root@desktop:/home/fabricio# docker rmi ubuntu:10.04
Untagged: ubuntu:10.04
Untagged:
ubuntu@sha256:f6695b2d24dd2e1da0a79fa72459e33505da79939c13ce50e90675c32988ab64
Deleted: sha256:e21dbcc7c9de73a19fc19187e8189bbe43617a08bc44f5a9ab124ed442ace155
Deleted: sha256:f500c3a7dec437bf271921d67a6d240c574a1aa186b7fa211818e7564f255da1
Deleted: sha256:170b376f64fb30995c140276be3d71dfb256b308d86183ca3b22aa93a79ad548
Deleted: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
root@desktop:/home/fabricio#
```

- Para verificar:

```
root@desktop:/home/fabricio# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
alpine       latest    49176f190c7e   6 weeks ago     7.05MB
ubuntu       14.04     13b66b487594   21 months ago   197MB
root@desktop:/home/fabricio#
```

Note que a imagem do ***ubuntu:10.04*** foi removida.

- PROCURANDO AS IMAGENS EM UM DETERMINADO REPOSITÓRIO

Para saber as imagens disponíveis em um determinado repositório, basta digitar o seguinte comando:

 # docker search “nome do repositório”

Exemplo: Para saber as imagens disponíveis no repositório **fabricioitajuba**.

```
PS C:\Docker\Lamp-CRUD1> docker search fabricioitajuba
NAME                                DESCRIPTION                                     TARS     OFFICIAL
fabricioitajuba/flr-api-php1        Esta imagem gera um container que irá gerar ...  0
fabricioitajuba/apache-php-mysqli   Imagem do apache com php e com mysqli instal...  0
PS C:\Docker\Lamp-CRUD1>
```

- APAGANDO UMA IMAGEM COM CONTAINERS EM EXECUÇÃO

No exemplo anterior, a imagem foi removida mas não tem nenhum container em execução a partir daquela imagem. Se tiver um ou mais containers em execução a partir de uma imagem e caso queira apagar a imagem o atributo **"-f"** deverá ser utilizado. Deve-se levar em consideração que quando uma imagem é excluída os containers associados à ela também serão excluídos.

Para apagar uma imagem com algum ou alguns containers em execução, utilize a seguinte linha de comando:

- Para remover uma imagem:
 # docker rmi -f "nome da imagem"

- DESCOBRINDO OS PASSOS DE CRIAÇÃO DE UMA IMAGEM

Podemos saber os passos de criação de uma imagem através do atributo history. Logo a linha de comando fica:

 # docker image history "nome da imagem"

Vamos baixar a imagem do apache através da seguinte linha de comando:

 # docker pull httpd

Agora digite a seguinte linha de comando:

 # docker image history httpd

```
root@desktop:/home/fabricio# docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
3f4ca61aafcd: Pull complete
2e3d233b6299: Pull complete
6d859023da80: Pull complete
f856a04699cc: Pull complete
ec3bbe99d2b1: Pull complete
Digest: sha256:f8c7bdfa89fb4448c95856c6145359f67dd447134018247609e7a23e5c5ec03a
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
root@desktop:/home/fabricio# docker image history httpd
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
73c10eb9266e   2 weeks ago   /bin/sh -c #(nop)  CMD ["httpd-foreground"]     0B
<missing>      2 weeks ago   /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>      2 weeks ago   /bin/sh -c #(nop) COPY file:c432ff61c4993ecd...   138B
<missing>      2 weeks ago   /bin/sh -c #(nop)  STOPSIGNAL SIGWINCH          0B
<missing>      2 weeks ago   /bin/sh -c set -eux;  savedAptMark="$(apt-m...   59.9MB
<missing>      2 weeks ago   /bin/sh -c #(nop)  ENV HTTPD_PATCHES=           0B
<missing>      2 weeks ago   /bin/sh -c #(nop)  ENV HTTPD_SHA256=eb397fee...   0B
<missing>      2 weeks ago   /bin/sh -c #(nop)  ENV HTTPD_VERSION=2.4.54     0B
<missing>      2 weeks ago   /bin/sh -c set -eux;  apt-get update;  apt-g...   4.76MB
<missing>      2 weeks ago   /bin/sh -c #(nop) WORKDIR /usr/local/apache2    0B
```

```
<missing>      2 weeks ago   /bin/sh -c mkdir -p "$HTTPD_PREFIX"  && chow...   0B
<missing>      2 weeks ago   /bin/sh -c #(nop)  ENV PATH=/usr/local/apach...   0B
<missing>      2 weeks ago   /bin/sh -c #(nop)  ENV HTTPD_PREFIX=/usr/loc...   0B
<missing>      2 weeks ago   /bin/sh -c #(nop)  CMD ["bash"]                   0B
<missing>      2 weeks ago   /bin/sh -c #(nop) ADD file:73e68ae6852c9afbb...   80.5MB
root@desktop:/home/fabricio#
```

Observe os passos de criação da imagem, cada passo é chamado de camada e cada camada possui um tamanho.

- SALVANDO E RECUPERANDO UMA IMAGEM

Podemos salvar uma imagem em nossa máquina e guarda-la em um local seguro. Quando salvamos uma imagem podemos escolher o local e toda a imagem será compactada através de uma arquivo **“.tar”**.

Para salvarmos uma imagem utilize a seguinte linha de comando:

 # docker save -o “local/nome do arquivo” “nome da imagem”

Para recuperar a imagem salva:

 # docker load -i “local/nome do arquivo”

- TRABALHANDO COM CONTAINERS

- Para listar todos os containers em execução:
 # docker ps

```
root@desktop:/home/fabricio# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@desktop:/home/fabricio#
```

Podemos observar que não existe nenhum container em execussão.

- Para listar todos os containers em execução e inativos:
 # docker ps -a

```
root@desktop:/home/fabricio# docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS                      PORTS     NAMES
9514ff0c4428   ubuntu    "bash"      55 seconds ago   Exited (0) 12 seconds ago             compassionate_hugle
66722c01bd75   alpine    "/bin/sh"   2 days ago       Exited (255) 2 days ago               friendly_snyder
root@desktop:/home/fabricio#
```

Note que nesse exemplo existem 2 containers que foram executados e agora estão inativos. O primeiro container foi utilizado uma imagem do **ubuntu** e a outra do **alpine**. Os números em hexadecimal 9514ff0c4428 e 66722c01bd75 são os id’s dos containers. Cada container recebeu um nome aleatório compassionate_hugle e friendly_snyder.

No momento da criação do container, se não especificarmos um nome o docker escolhe um nome aleatório.

Para manipular os containers, podemos utilizar o seu ID ou seu nome.

- EXECUTANDO O PRIMEIRO CONTAINER – HELLO-WORLD

No docker hub existe uma imagem de um container bastante simples. Ela normalmente é utilizada para testar se o docker foi instalado corretamente. É a imagem **hello-world**. Para baixá-la e executá-la, basta utilizar o seguinte comando:

 # docker run hello-world

Após a execução do comando, aparecerá as seguintes mensagens na tela:

```
root@desktop:/home/fabricio# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:94ebc7edf3401f299cd3376a1669bc0a49aef92d6d2669005f9bc5ef028dc333
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/

root@desktop:/home/fabricio#
```

As primeiras linhas são informações da imagem. Em seguida é apresentada a mensagem **“Hello from Docker!”**. No final o prompt é devolvido para o usuário. É um container com tempo de vida muito pequeno.

- CRIANDO UM CONTAINER A PARTIR DE UM SISTEMA OPERACIONAL

Vamos criar um container através de uma imagem do alpine.

 # docker run -it alpine

Após a execução do comando, teremos a seguinte tela:

```
root@desktop:/home/fabricio# docker run -it alpine
/ #
```

Note que o prompt de comando mudou para **"/#",** isso significa que o container foi criado e estamos dentro dele. Caso esse container for criado pela primeira vez, será mostrado a imagem sendo baixada.

Vamos digitar um simples comando para visualizar o que tem dentro do container:

/ # ls

Aparecerá a seguinte tela com as pastas dentro do container:

```
root@desktop:/home/fabricio# docker run -it alpine
/ # ls
bin   etc   lib   mnt   proc  run   srv   tmp   var
dev   home  media opt   root  sbin  sys   usr
/ #
```

Para sair do container, temos duas alternativas:

- Se pressionarmos as teclas **p e q** com a tecla **Ctrl** pressionada, sairemos do container sem "mata-lo". Isto é, ele continuará em execução em segundo plano;

- Se digitarmos o comando **exit**, sairemos do container e o mesmo será "morto".

Pressione as teclas **p e q** com a tecla **Ctrl** pressionada:

```
root@desktop:/home/fabricio# docker run -it alpine
/ # ls
bin   etc   lib   mnt   proc  run   srv   tmp   var
dev   home  media opt   root  sbin  sys   usr
/ # root@desktop:/home/fabricio#
```

Note que retornamos ao prompt de comando do nosso computador.

Para retornarmos ao container, primeiro teremos que descobrir qual é o id ou o nome do container. Digite o seguinte comando para sabermos essas informações:

 # docker ps

```
root@desktop:/home/fabricio# docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
51ec70602915   alpine    "/bin/sh"   13 minutes ago   Up 12 minutes             suspicious_aryabhata
root@desktop:/home/fabricio#
```

O id do container nesse exemplo é **51ec70602915** e o nome **suspicious_aryabhata**.

Vamos retornar ao container através do seguinte comando, podemos utilizar o id ou o nome do container, vamos utilizar o id:

 # docker attach 51ec70602915

```
root@desktop:/home/fabricio# docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
51ec70602915   alpine    "/bin/sh"   13 minutes ago   Up 12 minutes             suspicious_aryabhata
root@desktop:/home/fabricio# docker attach 51ec70602915
/ #
```

Note novamente que o cursor foi alterado.

Vamos sair do container digitando o comando **exit**:

/ # exit

```
root@desktop:/home/fabricio# docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
51ec70602915   alpine    "/bin/sh"   13 minutes ago   Up 12 minutes             suspicious_aryabhata
root@desktop:/home/fabricio# docker attach 51ec70602915
/ # exit
root@desktop:/home/fabricio#
```

Note novamente que o prompt altera para o prompt do usuário.

Se executarmos o comando **docker ps**, podemos observar que o container não está mais sendo executado:

 # docker ps

```
root@desktop:/home/fabricio# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@desktop:/home/fabricio#
```

Se executarmos o comando **docker ps -a**, podemos observar que o container foi executado e está "morto" ou desabilitado:

 # docker ps -a

```
root@desktop:/home/fabricio# docker ps -a
CONTAINER ID   IMAGE         COMMAND     CREATED          STATUS                      PORTS   NAMES
51ec70602915   alpine        "/bin/sh"   7 minutes ago    Exited (0) 6 minutes ago            suspicious_aryabhata
d2277af9dc61   hello-world   "/hello"    2 days ago       Exited (0) 2 days ago               stoic_gagarin
root@desktop:/home/fabricio#
```

- REMOVENDO UM CONTAINER

Se quisermos remover um container, basta digitarmos o comando abaixo com o “id” ou o “nome” do container, vamos remover utilizando seu id:

 # docker rm 51ec70602915

Se o container estiver em execução, teremos que utilizar o atributo **“-f”**:

 # docker rm -f 51ec70602915

Ao digitarmos o comando **docker ps -a**, podemos observar que o container não existe mais:

 # docker ps -a

```
root@desktop:/home/fabricio# docker ps -a
CONTAINER ID   IMAGE         COMMAND     CREATED          STATUS                      PORTS   NAMES
d2277af9dc61   hello-world   "/hello"    2 days ago       Exited (0) 2 days ago               stoic_gagarin
root@desktop:/home/fabricio#
```

Observe que o container foi apagado.

- Baixando e executando o container do ubuntu versão 14.04, dando um nome específico ao container e entrando no modo terminal:

$ docker run -ti --name meu_ubuntu ubuntu:14.04 /bin/bash

- Para verificar os “ids” dos containers em execução:

$ docker ps -q

- Para verificar os “ids” de todos os containers em execução:

$ docker ps -aq

- Para parar todos os containers de uma vez só:

$ docker stop $(docker ps -q)

- Para executar todos os containers de uma vez só:

$ docker start $(docker ps -aq)

- CRIANDO UM CONTAINER MAIS ELABORADO

Vamos agora, criar um container bem mais elaborado que o anterior. O container anterior, fizemos toda a sua manipulação através do seu "**id**" e para sairmos tínhamos sempre usar as teclas **Ctrl, P+Q**. No container a seguir, adicionaremos um nome e executaremos o mesmo em segundo plano. Através dessa maneira, podemos entrar no container e sair simplesmente digitando o comando **exit**.

Para darmos um nome ao container, utilizaremos a diretiva **-- name** e o nome do container. E para executa-lo em "background" ou seja em segundo plano, utilizaremos a opção **-d**.

Para criarmos o container utilizando a imagem do alpine, com os recursos apresentandos, digite a seguinte linha de comando (como exemplo vamos colocar o nome "**teste**"):

 # docker run -it -d --name teste alpine

```
root@desktop:/home/fabricio# docker run -it -d --name teste alpine
588ed814b3a19b3aac9123d7ce647717fa50370da67d12adc3d52351455790e5
root@desktop:/home/fabricio#
```

Note que o container foi criado mas não entramos dentro do container. O prompt é o mesmo do host.

Digitando o comando docker ps, podemos observar o container sendo executado em segundo plano e com o nome **teste**:

 # docker ps

```
root@desktop:/home/fabricio# docker run -it -d --name teste alpine
588ed814b3a19b3aac9123d7ce647717fa50370da67d12adc3d52351455790e5
root@desktop:/home/fabricio# docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS              PORTS     NAMES
588ed814b3a1   alpine    "/bin/sh"   2 minutes ago   Up About a minute             teste
root@desktop:/home/fabricio#
```

- EXECUTANDO UM COMANDO DENTRO DO CONTAINER SEM ACESSÁ-LO

Podemos executar um comando dentro do container sem estarmos dentro dele através da diretiva **exec**.

Vamos listar o conteúdo do container através do comando "**ls**" sendo executado através da diretiva **exec**.

 # docker exec teste ls

```
root@desktop:/home/fabricio# docker exec teste ls
bin
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
root@desktop:/home/fabricio#
```

Se quisermos a apresentação de uma forma mais interativa, basta associarmos a diretiva **-it**:

 # docker exec -it teste ls

```
root@desktop:/home/fabricio# docker exec -it teste ls
bin   etc   lib   mnt   proc  run   srv   tmp   var
dev   home  media opt   root  sbin  sys   usr
root@desktop:/home/fabricio#
```

Note que o comando é executado e retorna-se ao prompt do host.

Se quisermos acessar o container, no caso do alpine, pasta substituir o comando **ls** pelo comando **sh**:

 # docker exec -it teste sh

```
root@desktop:/home/fabricio# docker exec -it teste sh
/ #
```

Perceba que o prompt foi alterado indicando que estamos dentro do container.

Para sairmos do container, basta digitar o comando **exit**:

```
root@desktop:/home/fabricio# docker exec -it teste sh
/ # exit
root@desktop:/home/fabricio#
```

Perceba que o prompt retornou ao prompt do host.

Vamos verificar se o container ainda está em execução através do comando **docker ps**:

```
root@desktop:/home/fabricio# docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS     NAMES
588ed814b3a1   alpine    "/bin/sh"   20 minutes ago   Up 6 minutes              teste
root@desktop:/home/fabricio#
```

Note que o container ainda está em execução.

- PARANDO A EXECUÇÃO DE UM CONTAINER

Para pararmos a execução de um container, basta utilizar a opção **stop**, logo:

 # docker stop teste

Execute o comando **docker ps** e veja que o container não está mais em execução.

- ESTARTANDO UM CONTAINER

Para estartamos um container que foi parado pelo comando **stop**, basta utilizar a opção **start**, logo:

 # docker start teste

Execute o comando **docker ps** e veja que o container está em execução.

- PAUSANDO A EXECUÇÃO DE UM CONTAINER

Para pausarmos a execução de um container, basta utilizar a opção **pause**, logo:

 # docker pause teste

Execute o comando **docker ps** e veja que o container está pausado.

- ESTARTANDO UM CONTAINER QUE ESTÁ PAUSADO

Para estartar novamente um container que está pausado, basta utilizar a opção **pause**, logo:

 # docker unpause teste

Execute o comando **docker ps** e veja que o container voltou a funcionar.

- SABENDO A QUANTIDADE DE RECURSOS QUE O CONTAINER ESTÁ UTILIZANDO DA MÁQUINA HOST

Podemos saber quando de CPU e Memória que o container está utilizando da máquina host através do comando:

 # docker stats teste

Ao executarmos o comando, uma tela aparecerá com a quantidade de recursos que o container está utilizando:

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O   PIDS
588ed814b3a1   teste     0.00%     1.461MiB / 3.74GiB    0.04%     9.43kB / 0B   1.19MB / 0B   1
```

Para **sair**, pressione junto as teclas **Ctrl+c**.

Se quiser saber os recursos utilizados por todos os containers em execução, utilize a seguinte linha de comando:

 # docker stats

- SABENDO A QUANTIDADE DE PROCESSOS QUE ESTÃO SENDO EXECUTADOS NO CONTAINER

Para sabermos quais processos estão sendo executados no container, execute a seguinte linha de comando:

 # docker top teste

Na tela a seguir, podemos ver quais processos estão sendo executados:

```
root@desktop:/home/fabricio# docker top teste
UID          PID          PPID          C          STIME          TTY          TIME          CMD
root         9963         9943          0          13:22          ?            00:00:00      /bin/sh
root@desktop:/home/fabricio#
```

- VISUALIZANDO LOGS DO CONTAINER

Podemos ter acesso aos logs do container através da seguinte linha de comando:

 # docker logs teste

Uma tela com os logs do container será mostrada:

```
root@desktop:/home/fabricio# docker logs teste
/ # ls
bin   etc   lib   mnt   proc  run   srv   tmp   var
dev   home  media opt   root  sbin  sys   usr
/ # exit
/ # root@desktop:/home/fabricio#
```

- EXECUTANDO UM CONTAINER PARA ESTARTAR JUNTO COM A MÁQUINA HOST

Nos exemplos anteriores, criamos um container que funcionará enquanto a máquina host estiver funcionando. Caso precisa-se desligar ou reiniciar a máquina host, o container deixará de funcionar e teremos que estarta-lá manualmente. Para estartamos o container junto com a máquina host, podemos utilizar a diretiva **-- restart always**.

Vamos remover o container **teste** em execução através da seguinte linha de comando:

 # docker rm -f teste

Vamos criar um container utilizando a diretiva **--restart always**:

 # docker run -it -d --name teste --restart always alpine

Note que o container foi criado e toda vez que a máquina host for ligada, o container reiniciará junto.

- CRIANDO UMA IMAGEM A PARTIR DE UM CONTAINER

Suponhamos que criamos um container e foram feitas modificações dentro desse container. Podemos salvar esse container em uma imagem para usa-lá quando for necessário.

Para criar uma imagem a partir de um container, basta utilizar a seguinte linha:

 # docker commit “nome ou id do container” “nome da nova imagem”

Podemos também, criar uma imagem customizada através de um arquivo chamado Dockerfile como será mostrado posteriormente.

- CRIANDO IMAGENS CUSTOMIZADAS ATRAVÉS DO ARQUIVO DOCKERFILE

Quando na criação da imagem, precisar de alguns comandos específicos, o **Dockerfile** é um arquivo que ajudará na criação da mesma.

O arquivo de dockerfile precisa ser escrito com o seguinte nome:

Dockerfile

Para executar o **Dockerfile** para criar a imagem, basta entrar com o seguinte comando:

 # docker build -t “nome:versao” .

O “**.**” significa que o arquivo **Dockerfile** está no diretório corrente.

Opções que podem ser usadas dentro do arquivo:

- FROM -> determina qual imagem de sistema operacional será usado para criar a imagem da aplicação.

ex:
FROM ubuntu

- MAINTAINER -> Descreve em é o mantenedor do container

ex:
MAINTAINER Fabricio Ribeiro

- RUN -> Executa comandos no início da criação do container, sempre usar "&&" quando for usar mais de um comando. evitar o uso de vários RUNs no Dockerfile.

ex:
RUN apt-get update && apt-get install apache2 && apt-get clean

- ADD -> adiciona um arquivo ou uma pasta do host em uma pasta do container (envia arquivos .tar)

ex:
ADD arquivo.txt /pasta/

- CMD -> comando ou parâmetro do entrypoint (principal processo dentro de um container)

ex:
CMD ["sh", "-c", "echo", "$HOME"]

- LABEL -> coloca metadata (descreve a versão por exemplo do container)

ex:
LABEL Description="Isso é um teste"

- COPY -> copia arquivo e diretórios para do host para o container menos arquivo .tar (empacotados)

ex:
COPY arquivo.txt /pasta/

- ENTRYPOINT -> Faz com que um processo seja o principal do container, se ele "cair", o container também "cai".

ex:
ENTRYPOINT ["/usr/bin/apache2ctl", "-D", "FOREGROUND"]

Caso o **ENTRYPOINT** esteja configurado, o **CMD** será um parâmetro dele. Caso o **ENTRYPOINT** não esteja configurado, o **CMD** funciona como comando.

- ENV -> configura variáveis de ambiente para o container.

ex:
ENV meunome="Fabricio Ribeiro"

- EXPOSE -> expõe a porta do container

ex:
EXPOSE 80

- USER -> Define quem será o usuário default da imagem, caso não configurado o usuário será o root.

ex:
USER fabricio

- WORKDIR -> seleciona o diretório em que o container será executado.

ex:
WORKDIR /catota

Quando executada a imagem, o container estará todo na pasta /catota

VOLUME -> determina o volume

ex:
VOLUME /diretório

Para criar a imagem, utilize a seguinte linha de comando:

 # docker build -t “nome da imagem” .

O parâmetro **“.”** significa que o arquivo está dentro do diretório corrente;
O parâmetro **“-t”** permite colocar o nome da imagem.

- CRIANDO UM SIMPLES DOCKERFILE

Como exemplo, vamos criar um simples **Dockerfile** e posteriormente um mais elaborado. O exemplo será a imagem de um container que mostrará uma mensagem de **Hello World!** na tela quando o container for executado.

Inicialmente teremos a imagem escolhida do alpine, em seguida um comentário dizendo o nome do programa. Em seguida o nome do mantenedor e por final rodaremos o comando **“echo Hello Wolrd!”**

- Crie um diretório com o nome **hello**:

 # mkdir hello

- Entre no diretório criado:

 # cd hello

Através de um editor de texto, crie um arquivo com o nome **Dockerfile** e digite as seguintes linhas de comando dentro do arquivo:

```
FROM alpine
LABEL Programa Hello World
MAINTAINER Fabricio Ribeiro
CMD [“echo”, “Hello World!”]
```

- Salve e saia do arquivo, digite a seguinte linha de comando:

 # docker build -t hello_image .

Note o processo de criação da imagem.

```
root@desktop:/home/fabricio/hello# docker build -t hello_image .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM alpine
 ---> 49176f190c7e
Step 2/4 : LABEL Programa Hello World
 ---> Running in 33a61e225627
Removing intermediate container 33a61e225627
 ---> 97106c49e009
Step 3/4 : MAINTAINER Fabricio Ribeiro
 ---> Running in df7e27f489e0
Removing intermediate container df7e27f489e0
 ---> 9ce7bbe77506
Step 4/4 : CMD ["echo", "Hello World!"]
 ---> Running in 8d827ab99837
```

```
Removing intermediate container 8d827ab99837
 ---> dc7f7cb40609
Successfully built dc7f7cb40609
Successfully tagged hello_image:latest
root@desktop:/home/fabricio/hello#
```

- Execute o seguinte comando para visualizar as imagens salvas:

 # docker images

```
root@desktop:/home/fabricio/hello# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
hello_image   latest    dc7f7cb40609   31 seconds ago   7.05MB
htop_image    latest    3bbcc98ff320   4 hours ago      167MB
httpd         latest    73c10eb9266e   2 weeks ago      145MB
ubuntu        latest    6b7dfa7e8fdb   4 weeks ago      77.8MB
alpine        latest    49176f190c7e   6 weeks ago      7.05MB
hello-world   latest    feb5d9fea6a5   15 months ago    13.3kB
ubuntu        16.04     b6f507652425   16 months ago    135MB
ubuntu        14.04     13b66b487594   21 months ago    197MB
root@desktop:/home/fabricio/hello#
```

Perceba a imagem **“hello_image”** criada.

- Vamos executar um container a partir da imagem criada.

 # docker run -it --rm --name hello_container hello_image

```
root@desktop:/home/fabricio/hello# docker run -it --rm --name hello_container hello_image
Hello World!
root@desktop:/home/fabricio/hello#
```

Observe que após a execução o container é “morto”. A diretiva **“--rm”** remove o container logo após a sua execução.

- Vamos verificar o processo de criação da imagem através da seguinte linha de comando:

 # docker image history hello_image

```
root@desktop:/home/fabricio/hello# docker image history hello_image
IMAGE          CREATED         CREATED BY                                      SIZE      COMMENT
dc7f7cb40609   2 minutes ago   /bin/sh -c #(nop)  CMD ["echo" "Hello World!...   0B
9ce7bbe77506   2 minutes ago   /bin/sh -c #(nop)  MAINTAINER Fabricio Ribei...   0B
97106c49e009   2 minutes ago   /bin/sh -c #(nop)  LABEL Programa=Hello World   0B
49176f190c7e   6 weeks ago     /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      6 weeks ago     /bin/sh -c #(nop) ADD file:587cae71969871d3c...   7.05MB
root@desktop:/home/fabricio/hello#
```

- CRIANDO UM DOCKERFILE UM POUCO MAIS ELABORADO

Neste exemplo criaremos a imagem de um container um pouco mais elaborada utilizando uma imagem já criada do python. O programa mostrará novamente um “Hello World!” na tela em python.

- Crie um diretório com o nome **hello-py**:

 # mkdir hello-py

- Entre no diretório criado:

 # cd hello-py

- Crie um diretório com o nome **app**:

 # mkdir app

- Entre no diretório criado:

 # cd app

Através de um editor de texto, crie um arquivo com o nome **index.py** e digite a seguinte linha de comando dentro do arquivo:

```
print(“Hello World!”)
```

- Salve e saia do arquivo;

- Retorne um diretório através do comando:

 # cd ..

Novamente, através de um editor de texto, crie um arquivo com o nome **Dockerfile** e digite as seguintes linhas de comando dentro do arquivo:

```
FROM python:3
WORKDIR /app
COPY . .
CMD [“python”, “app/index.py”]
```

- Salve e saia do arquivo;

Novamente, através de um editor de texto, crie um arquivo com o nome **.Dockerignore** e digite as seguintes linhas dentro do arquivo:

```
Dockerfile
```

- Salve e saia do arquivo;

O arquivo **.Dockerignore** fará que durante o processo de cópia dos arquivos para a pasta **/app** dentro do container, todo o conteúdo será copiado para a pasta **/app** menos o arquivo **Dockerfile**.

Para “buildar a imagem”, digite a seguinte linha de comando:

 # docker build -t hello-py_image .

Note o processo de criação da imagem.

```
root@desktop:/home/fabricio/docker/hello-py# ls
root@desktop:/home/fabricio/docker/hello-py# mkdir app
root@desktop:/home/fabricio/docker/hello-py# cd app
root@desktop:/home/fabricio/docker/hello-py/app# nano index.py
root@desktop:/home/fabricio/docker/hello-py/app# cd ..
root@desktop:/home/fabricio/docker/hello-py# nano Dockerfile
root@desktop:/home/fabricio/docker/hello-py# nano .Dockerignore
root@desktop:/home/fabricio/docker/hello-py# docker build -t hello-py_image .
Sending build context to Docker daemon  4.608kB
Step 1/4 : FROM python:3
3: Pulling from library/python
32de3c850997: Pull complete
fa1d4c8d85a4: Pull complete
c796299bbbdd: Pull complete
81283a9569ad: Pull complete
60b38700e7fb: Pull complete
0f67f32c26d3: Pull complete
1922a20932d4: Pull complete
47dd72d73dba: Pull complete
9b2b0e41cfb6: Pull complete
Digest: sha256:a46b962871434568d186ef17ae7038055e17c670833ca5320fc107435fa146d7
Status: Downloaded newer image for python:3
 ---> 9cbe331577ed
Step 2/4 : WORKDIR /app
 ---> Running in c55fb5e06d44
Removing intermediate container c55fb5e06d44
 ---> 43cac060d632
Step 3/4 : COPY . .
 ---> bd71c3828c72
Step 4/4 : CMD ["python", "app/index.py"]
 ---> Running in c3bcb39f1a8f
Removing intermediate container c3bcb39f1a8f
 ---> 9812e3fb41a5
Successfully built 9812e3fb41a5
Successfully tagged hello-py_image:latest
root@desktop:/home/fabricio/docker/hello-py#
```

- Execute o seguinte comando para visualizar as imagens salvas:

 # docker images

```
root@desktop:/home/fabricio/docker/hello-py# docker images
REPOSITORY      TAG       IMAGE ID       CREATED              SIZE
hello-py_image  latest    9812e3fb41a5   About a minute ago   932MB
hello_image     latest    dc7f7cb40609   29 hours ago         7.05MB
htop_image      latest    3bbcc98ff320   34 hours ago         167MB
python          3         9cbe331577ed   4 days ago           932MB
```

```
httpd          latest    73c10eb9266e   2 weeks ago         145MB
ubuntu          latest    6b7dfa7e8fdb   4 weeks ago         77.8MB
alpine         latest    49176f190c7e   7 weeks ago        7.05MB
hello-world      latest    feb5d9fea6a5   15 months ago       13.3kB
ubuntu          16.04     b6f507652425   16 months ago       135MB
ubuntu          14.04     13b66b487594   21 months ago       197MB
root@desktop:/home/fabricio/docker/hello-py#
```

Perceba a imagem **“hello-py_image”** criada.

- Vamos executar um container a partir da imagem criada.

 # docker run -it --rm --name hello-py_container hello-py_image

```
root@desktop:/home/fabricio/docker/hello-py# docker run -it --rm --name hello-py_container hello-py_image
Hello World!
root@desktop:/home/fabricio/docker/hello-py#
```

- TRABALHANDO COM REDES

- Para verificar quais opções posso trabalhar com o docker:

 # docker network --help

```
root@desktop:/home/fabricio# docker network --help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
root@desktop:/home/fabricio#
```

- Podemos também, saber mais informações sobre o comando específico (exemplo “create”):

 # docker network create --help

```
root@desktop:/home/fabricio# docker network create --help

Usage:  docker network create [OPTIONS] NETWORK
```

```
Create a network

Options:
      --attachable          Enable manual container attachment
      --aux-address map     Auxiliary IPv4 or IPv6 addresses used by
                            Network driver (default map[])
      --config-from string  The network from which to copy the configuration
      --config-only         Create a configuration only network
  -d, --driver string       Driver to manage the Network (default "bridge")
      --gateway strings     IPv4 or IPv6 Gateway for the master subnet
      --ingress             Create swarm routing-mesh network
      --internal            Restrict external access to the network
      --ip-range strings    Allocate container ip from a sub-range
      --ipam-driver string  IP Address Management Driver (default "default")
      --ipam-opt map        Set IPAM driver specific options (default map[])
      --ipv6                Enable IPv6 networking
      --label list          Set metadata on a network
  -o, --opt map             Set driver specific options (default map[])
      --scope string        Control the network's scope
      --subnet strings      Subnet in CIDR format that represents a
                            network segment
root@desktop:/home/fabricio#
```

- Para listar todas as redes do docker:

 # docker network ls

```
root@desktop:/home/fabricio# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
e0770daca82e   bridge    bridge    local
91177046584d   host      host      local
222e7edd4860   none      null      local
root@desktop:/home/fabricio#
```

O docker possui três redes padrão: **bridge, host, none**.

Quando um container é criado e não especificamos uma rede para ele, o docker inclui o mesmo na rede bridge padrão. Na rede bridge os IP’s dos containers sempre começam com o endereço 172.17.x.x/16. Lembrando que é feito NAT entre os containers e o computador onde está sendo executado o docker.

Na rede **bridge** padrão, a comunicação entre os containers será feita através de seus IP’s, não tendo a possibilidade de comunicação via DNS onde o próprio nome do container é vinculado ao seu IP.

Para comunicarmos com um outro container utlizando DNS devemos criar outra rede com o driver bridge.

A rede **none**, isola o container das demais redes. O mesmo só possuirá o localhost.

A rede **host** permite que o container use os recursos de rede do computador onde está sendo executado o docker.

- Criando um container e associando à rede host:

 # docker run -itd --name teste --net host ubuntu

- Para saber as informações de uma rede (exemplo a rede “bridge”):

 # docker network inspect bridge

```
fabricio@notebook-lg:~/GitHub/Docker-Help$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "e2aeae6ddeb3117d3da59ba14afa748934df88ac60b7ea4eb71d2590562f223a",
        "Created": "2021-10-14T21:01:59.460041387-03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
fabricio@notebook-lg:~/GitHub/Docker-Help$
```

- Para criar uma nova rede (exemplo “nova_rede”):

 # docker network create nova_rede

```
fabricio@notebook-lg:~/GitHub/Docker-Help$ docker network create nova_rede
af738829bc6a88832611b5b4a48518a88e00f72a55420ea62bfaed14ccd47e6e
fabricio@notebook-lg:~/GitHub/Docker-Help$
```

- Para criar uma rede com um “range” diferente (exemplo “minha_rede2”):

 # docker network create minha_rede2 --subnet 192.168.134.0/24 --gateway 192.168.134.1

```
fabricio@notebook-lg:~/GitHub/Docker-Help$ docker network create minha_rede2 --subnet 192.168.134.0/24 --gateway 192.168.134.1
6ce1d7c42a6124898b6930987e12ab120229cc9de0a35057c8a13b0f0c5c2672
fabricio@notebook-lg:~/GitHub/Docker-Help$
```

- Associando um container criado e executando a nova rede criada:

 # docker network connect minha_rede2 meu_ubuntu

```
fabricio@notebook-lg:~/GitHub/Docker-Help$ docker network connect minha_rede2 meu_ubuntu
fabricio@notebook-lg:~/GitHub/Docker-Help$
```

- Para saber os containers (em execução) associados a rede:

 # docker network inspect minha_rede2

```
fabricio@notebook-lg:~/GitHub/Docker-Help$ docker network inspect minha_rede2
[
    {
        "Name": "minha_rede2",
        "Id": "c4a068d87bbf2759b7218882f9b6cb4b067aa65cec3a1a30d2aabcbf924cc350",
        "Created": "2021-10-14T22:02:18.468161564-03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.134.0/24",
                    "Gateway": "192.168.134.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "581647ff3cffdabaea19af49fc7b801e77f04afe1dc3015cc76eea9a7c16f8f1": {
                "Name": "meu_ubuntu",
                "EndpointID": "07fab6120081095dcdcf2fd64fc82e0487fea138049f0716d146b8cd23f59fb5",
                "MacAddress": "02:42:c0:a8:86:02",
                "IPv4Address": "192.168.134.2/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
fabricio@notebook-lg:~/GitHub/Docker-Help$
```

- Para visualizar melhor os ips associados a rede:

 # docker network inspect “nome ou id da rede”

```
C:\docker\mysql_wordpress_phpmyadmin>docker network ls
NETWORK ID          NAME                                              DRIVER    SCOPE
ff2d5c662261        bridge                                            bridge    local
340059fd224e        host                                              host      local
04ed6bdb0df5        mysql_wordpress_phpmyadmin_wordpress_net          bridge    local
2408b701388e        none                                              null      local

C:\docker\mysql_wordpress_phpmyadmin>docker network inspect 04 | grep IPv4Address
                "IPv4Address": "172.18.0.3/16",
                "IPv4Address": "172.18.0.4/16",
                "IPv4Address": "172.18.0.2/16",

C:\docker\mysql_wordpress_phpmyadmin>
```

Observe que não precisamos digitar todo o id, somente o início.

- Para remover uma rede criada (exemplo “nova_rede”):

 # docker network rm nova_rede

```
fabricio@notebook-lg:~/GitHub/Docker-Help$ docker network rm nova_rede
nova_rede
fabricio@notebook-lg:~/GitHub/Docker-Help$
```

- Para remover todas as redes que não estão sendo utilizadas:

 # docker network prune

```
fabricio@notebook-lg:~/GitHub/Docker-Help$ docker network prune
WARNING! This will remove all custom networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
redelocal
minha_rede2

fabricio@notebook-lg:~/GitHub/Docker-Help$
```

Alguns parâmetros são importantes na criação de containers utilizando redes:

- Configuração de dns

Ex:
docker run -ti **--dns 8.8.8.8** debian

- Configuração de hostname

Ex:
docker run -ti **--hostname catota** debian

- COMUNICAÇÃO ENTRE CONTAINERS UTILIZANDO DNS

- Crie o Container1:

 # docker run -ti -d --name container1 ubuntu

- Entre no container1:

 # docker exec -it container1 sh

- Instale o comando ping:

 # apt-get update && apt-get install -y iputils-ping && apt-get clean

saia do container digitando: **exit**

- Crie o Container2:

 # docker run -ti -d --name container2 ubuntu

- Entre no container2:

 # docker exec -it container2 sh

- Instale o comando ping:

 # apt-get update && apt-get install -y iputils-ping && apt-get clean

saia do container digitando: **exit**

-Verifique se os containers foram criados e estão sendo executados:

 # docker ps

- Crie a rede:

 # docker network create rede

-Verifique se a rede foi criada:

 # docker network ls

- Caso queira apagar as redes que não estão sendo utilizadas:

 # docker network pune

- Conecte o container1 na rede:

 # docker network connect rede container1

- Conecte o container2 na rede:

 # docker network connect rede container2

- Verifique se os containers foram conectados a rede:

 # docker network inspect rede

ou

 # docker network inspect rede | grep Name

- Entre dentro do container1:

 # docker exec -it container1 /bin/bash

- Execute um ping para o container2

 # ping container2

Para parar o comando ping: **Ctrl+c**

Para sair do container1, digite **exit**

- Entre dentro do container2:

 # docker exec -it container2 /bin/bash

- Execute um ping para o container1

 # ping container1

Para parar o comando ping: **Ctrl+c**

Para sair do container1, digite exit

- Caso queira desativar os containers 1 e 2:

 # docker container stop container1

 # docker container stop container2

- Caso queira ativar os containers 1 e 2:

 # docker container start container1
 # docker container start container2

- Para remover os containers em execução e apaga-los:

 # docker rm container1 container2

- Remova os volumes não utilizados:

 # docker volume prune

- Remova as redes não utilizadas:

 # docker network prune

O mesmo exemplo pode ser executado através do **Dockerfile** juntamente com o **Docker-compose**:

Antes de criar os arquivos **Dockerfile** e **docker-compose.yml**, crie uma pasta com o nome **"two-ubuntu"**, dentro da pasta, crie outra pasta com o nome **"pasta"**.

A pasta "pasta"é um volume lincado com a pasta **/home** dos dois containers ubuntu criados.

Crie o arquivo **Dockerfile** para criar a imagem **teste-ubuntu:14.04** apartir da imagem do **ubuntu 14.04**:

Dockerfile

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y iputils-ping && apt-get clean
```

Crie a imagem a partir do **Dockerfile** através da seguinte linha de comando:

 # docker build -t teste-ubuntu:14.04 .

Crie o arquivo **docker-compose.yml** para criar os dois containers do exemplo:

docker-compose.yml

```
version: "3.7"

services:

  container1:
     container_name: container1
     image: teste-ubuntu:14.04
```

```
      networks:
        ubuntu_rede:
          aliases:
            - rede
      volumes:
            - ./pasta:/home/
      entrypoint: /bin/sh
      stdin_open: true
      tty: true

  container2:
      container_name: container2
      image: teste-ubuntu:14.04
      networks:
        ubuntu_rede:
          aliases:
            - rede
      volumes:
            - ./pasta:/home/
      entrypoint: /bin/sh
      stdin_open: true
      tty: true

networks:
  ubuntu_rede:
     name: rede
     driver: bridge
     ipam:
       driver: default
```

Para executar:

- Processar o arquivo de composição e iniciar a aplicação:

 # docker-compose up -d

- Para remover os containers, redes e volumes descritos no arquivo de composição:

 # docker-compose down -v

- Para mostrar os containers criados:

 # docker-compose ps

- EXEMPLO COMPLETO UTILIZANDO O DOCKER COMPOSE

Neste exemplo iremos criar 3 containers com o wordpress, mysql e phpmyadmin utilizando variáveis de ambiente em um arquivo .env.

- Crie uma pasta com o nome **mysql_wordpress_phpmyadmin**;

- Dentro da pasta, crie um arquivo com o nome **compose.yaml**;

- Edite o arquivo compose.yaml digitando o seguinte código:

```
services:
  mysql:
    image: mysql:8.3.0
    container_name: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
      MYSQL_DATABASE: ${DB_DATABASE}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASSWORD}
    ports:
      - 3306:3306
    volumes:
      #- mysql_vol:/var/lib/mysql
      - ./mysql_vol:/var/lib/mysql
    networks:
      - wordpress_net

  wordpress:
    depends_on:
      - mysql
    image: wordpress:6.4.3
    container_name: wordpress
    restart: always
    environment:
      WORDPRESS_DB_HOST: mysql:3306
      WORDPRESS_DB_USER: ${DB_USER}
      WORDPRESS_DB_PASSWORD: ${DB_PASSWORD}
      WORDPRESS_DB_NAME: ${DB_DATABASE}
    volumes:
      #- wordpress_vol:/var/www/html
       - ./wordpress_vol:/var/www/html
    ports:
      - 8080:80
    networks:
      - wordpress_net

  phpmyadmin:
    depends_on:
      - mysql
```

```
    image: phpmyadmin:latest
    container_name: phpmyadmin
    restart: always
    ports:
      - 8081:80
    environment:
      - PMA_ARBITRARY=1
    networks:
      - wordpress_net

volumes:
  mysql_vol:
  wordpress_vol:

networks:
  wordpress_net:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: "192.168.1.0/24"
          gateway: "192.168.1.1"
```

- Salve e feche o arquivo;

- Crie outro arquivo com o nome **.env**;

- Digite as seguintes linhas no arquivo:

```
DB_ROOT_PASSWORD = pass
DB_DATABASE = wordpress
DB_USER = admin
DB_PASSWORD = pass
```

Neste arquivo será armazenada as senhas e o nome do banco de dados.

- Salve e feche o arquivo;

- Para subir os containers:

 # docker compose --env-file .env up -d

- Para parar os containers sem deletar os volumes e redes:

 # docker compose down

- Para parar os containers, deletar os volumes e redes:

 # docker compose down -v

Para acessar o wordpress: localhost:8080

Para acessar o phpmyadmin: localhost:8081

Servidor: mysql

Usuário: root

Senha: pass

Quando executamos o comando **docker compose up**, os containers são criados de forma aleatória e também alguns demoram mais que os outros para serem criados. Isso é um problema quando um container depende do outro para funcionar. Para resolver esse problema, utilizamos a tag **“depends_on”**. Isso significa que o container que tiver essa tag, será criado depois que o container que não possui a tag ser criado.

Os volumes estão configurados para serem salvos na pasta do projeto, caso queira salvar em uma pasta determinada para o docker, comente a linha correspondente ao container e descomente a linha anterior.

- CRIANDO UMA API EM PHP E PUBLICANDO NO DOCKERHUB

Neste exemplo iremos criar uma simples api que gera um número aleatório entre 0 e 100 utilizando uma imagem com apache e o php. Esta api é muito útil para testes.

- Crie uma pasta com o nome **api-php1**;

- Dentro da pasta, crie um arquivo com o nome **Dockerfile**;

- Abra o arquivo **Dockerfile** inserindo as seguintes linhas:

Dockerfile

```
FROM php:7.2-apache
COPY index.php /var/www/html/
```

- Salve e Feche o arquivo;

- Dentro da pasta, crie um arquivo com o nome **index.php**;

- Abra o arquivo **index.php** inserindo as seguintes linhas:

index.php

```
<?php
  session_start();
  $number = rand(0, 100);
  $dados = array('number' => $number,);
  echo json_encode($dados);
?>
```

- Salve e Feche o arquivo;

- CRIANDO A IMAGEM:

Para criar a imagem, digite a seguinte linda de comando:

 # docker build -t flr-api-php1:latest .

- Crie o container através da seguinte linda de comando:

 # docker run -d -p 7376:80 --name api-php1 flr-api-php1

- Abra o navegador e digite o seguinte endereço:

localhost:7376

- Atualize a página várias vezes e observe que cada vez que a página é atualizada, um novo número é gerado.

- PUBLICANDO NO DOCKERHUB

Para publicar sua imagem no dockerhub, Você terá que ter uma conta, caso não tenha, abra uma.

- Entre no site, clicke em **“Repositories”**, **“Create a repository”**;

- Dê um nome ao repositório, exemplo: **“flr-api-php1”**;

- Escreva uma descrição à respeito do repositório: exemplo: **“Esta imagem gera um container que irá gerar um número aleatório entre 0 e 100”**;

- Clique em **“Create”**;

- No terminal, digite o seguinte comando para criar uma **“tag”** para imagem (troque fabricioitajuba pelo nome da sua conta no docker hub):

 # docker tag flr-api-php1 fabricioitajuba/flr-api-php1:latest

- Faça o login no terminal:

 # docker login

- Faça o push da imagem (troque fabricioitajuba pelo nome da sua conta no docker hub):

 # docker push fabricioitajuba/flr-api-php1:latest

Verifique se a imagem foi para o repositório do docker hub, no site, clique em **“Repositores”**.

- FAZENDO O PULL DA IMAGEM:

- Apague as imagens criadas no docker do computador;

- Digite a seguinte linha de comando:

 # docker pull fabricioitajuba/flr-api-php1

- Crie novamente o container para testar a image.

 # docker run -d -p 7376:80 --name api-php1 fabricioitajuba/flr-api-php1

- CRIANDO UM CONTAINER COMPLETO DO MYSQL:

 # docker run -e MYSQL_ROOT_PASSWORD=root --name mysql --restart always -d -p 3306:3306 -v mysql-db:/home/fabricio/Temp mysql:5.7

- no raspberrypi
 # docker run -e MYSQL_ROOT_PASSWORD=root --name mysql --restart always -d -p 3306:3306 -v mysql-db:/home/pi/docker/mysql hypriot/rpi-mysql

- phpmyadmin
 # docker run --name phpmyadmin --restart always -d --link mysql:db -p 8080:80 phpmyadmin:5.1.0