Docker

Manual simples

Agradecimentos

Esse simples manual foi gerado como um compilado de várias fontes com o intuido de ser um guia e uma referência ao tema, não tomando o escrito como minha própria autoria e sim algo como uma junção para a comunidade.

Introdução

Este manual visa:

- Demonstrar o que é o Docker;
- Diferença do docker e máquinas virtuais;
- Porque da utilização do do Docker;
- Demonstrar de forma introdutório, completa e menos formal possível sobre a plataforma e seu uso;

Atenção

Para todos que desejam um contêudo mais rico doque o apresentado neste manual, recomendo o **DOCS** do próprio Docker, Link: https://docs.docker.com/

Um pouco de história

Virtualização surgiu como um método de compartilhar o tempo de processamento de forma a se obter mais resultados com o custo de mesmo tempo, exemplo:

Sem virtualização:

- Se eram necessários 2 horas para duas tarefas;

Com virtualização:

- Necessário 1 hora para as duas tarefas

O motivo desse menor tempo é claro a virtualização, duas máquinas "distintas" executaram tarefas indepentendentes e entregaram seus resultados.

A virtualização começou a ser implementanda realmente no anos 2000, mesmo que seu conceito já existise em 1960, o hardware não era capacitado o suficiente, em 2000, o hardware teve grandes avanços de várias formas, melhores tecnologias, preço menor e diversos outros , além de que os propríos softwares tiveram grande evolução após o fim da "crise do software", possibilitando novos empreendimentos na técnologia.

Qual a graça da virtualização

A graça é a criação de um novo sistema operacional lógico e separado do sistema base do **HOST**, onde essa máquina será servida por um **POOL** de hardware instanciado e controlado por uma tecnologia virtualização hipervisor.

Então e o Docker?

Tecnologias de containers não são novas, a LXC é uma desses moelos, ela trabalha em containers de Hipervisior 1, onde o LXC controla os containers que serão os sistemas que se utilizaram do hardware.

O Docker diferente do LXC é um hypervisior 2, onde ele opera por cima de um sistema operacional base, seu nome atualmente é famoso por N motivos, más os principais são:

- Densevolvido em GO (Linguagem da Google);
- Facilidade de crescimento, instancias e demais;

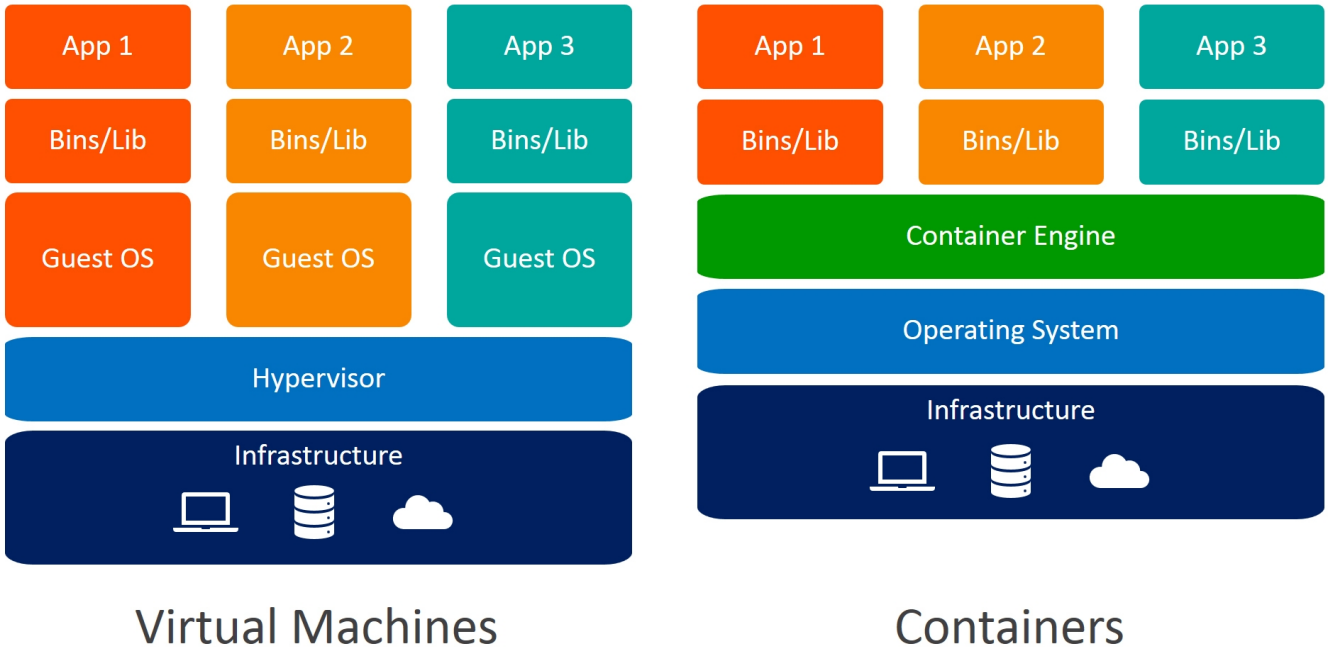
Falou, falou, falou, más não falou nada

Containers e V.M são diferentes em constituição, porém podem ser implementados pelos mesmo propósito, más o resultados podem ser váriados.

Diferença de uma V.M para um Container

Uma V.M (Virtual machine ou máquina virtual) simula um ambiente completo sobre o atual sistema operacional, já um container Docker é parte de um S.O, ele se utilizada da base para seu funcionamento e importa bibliotecas necessários para funcionar, más isso faz alguma diferença? A respota é: **Uma V.M é um S.O inteiro novo, coisa que um container não é!**

Título: Demonstração da diferença em níveis de uma VM e Contianers



Fonte: Weave Works, Um guia prático para escolher entre contêineres Docker e VMs

Um exemplo que é sempre visto pela comunidade é "porque vou instalar tudo se só quero um software N", essa é uma das sacadas do container em relação a virtulização, ambientes mais enxutos e menores a qual são criados para propósitos unicos, que possam ser reconstruidos a vontagem e com agilidade.

Motivos da virtualização e container

O uso de ambos é por N motivos, testes, montagem de ambientes de produção, escalabilidade, Black-friday e demais, ambientes que os utilizam podem ser melhores gerenciados por motivo de toda automação que ambos trazem.

Alguns motivos para seus usos:

- Aplicações tentando sair pela mesma por ao mesmo tempo (deadlock);
- Se uma aplicação travar o HOST virtual ou container não afeta os demais ou ao HOST hospedero;
- Criar ambientes com multiplas versões de softwares necessários;
- Melhor uso do custo por tempo de processamento;

Claro que ambos não são soluções finais para todos os problemas do ambiente, porém são apoiadores de ambientes flexiveis, com seu uso a criação de cluster de máquinas, subir ambientes e demais se tornam mais simples e rápidos;

E o motivo de se utilizar o Docker?

Aqui entra o motivo de se utilizar Docker em vez da virtualização, um exemplo, uma V.M virtualiza um S.O interno dentro do sistema atual, isso não é desejado por causar cunsumos de recursos, digamos que precisamos somente de um interpretador BASH dentro da V.M e foi instalado o Ubuntu inteiro com o GNOME, sabe aquele container que funcionaria com 256MB de RAM, essa V.M precisaria de uns 2GB de RAM além de mais espaço em disco;

Hardware	Ubuntu VM	Ubuntu Container
Núcleos de CPU	4	1
Memória	4GB	512MB
Armazenamento	20GB	300MB

A tabela demonstra um Ubuntu GNOME com todos os seus requisitos sendo atendidos durante a criação da V.M, já o container somente irá trazer o minimo para funcionamento, lembrando que o docker gerencia dinamicamente recursos da máquina para o container, porém é possível travar valores fixos máximos, como o exemplo demonstra acima.

Além de demais vantagens do Docker sobre a virtualização são:

- Criação de ambientes;
- Baixo consumo de recursos de forma inicial;
- Resositório online de imagens;
- Facilidade de migração de sistemas de uma OS para ourto;

Sobre o Docker

Falou, falou e falou, más não falou, o que é Docker?

Docker é inicialmente um projeto da empresa criado para automação na criação de ambientes na empresa dotCloud em 2012-2013, após o "BUM" da tecnologia com o Docker, a empresa se renomeou para Docker Inc e disponibilizou o código-fonte da aplicadação, além da mesma ainda ser uma mantenedora da mesma, o inicialmente foi um conjunto das tecnologias LXC e Linguagem GO, após várias atualizações a Docker Inc largou a base do LXC e migrou para um modelo de prórpia autoria, a libcontainer

Más como ele surgiu?

Inicialente dotcloud era uma empresa para facilitar migrações de clientes a nuvem, assim eles eles usavam o AWS da Amazon para contratar uma máquinas e para instanciar recursos e forma autonoma, eles criaram o Docker, que tomava como tarefa o gerenciamento e criação de conteiners sobre demanda.

Atualmente

A empresa se renomeou depois do BUM e atualmente é a mantenedora principal do Docker e do Docker HUB, a qual possui serviços e recebe de investimentos e doações para continuar as atividade

Tecnologia

Ferramentas modernas para deployar e rodar aplicações;

- **Docker engine:** É o cara que faz o intermedio do sistema e dos containers;
- **Docker compose:** Facilitar a manipulações de multiplos containers de uma vez(orquestra);
- **Docker swarm:** Multiplos docker para criar um cluster;
- **Docker hub:** Repositorio com mais de 250 mil imagens de conteiners;

- **Docker machine:** Instalar e gerenciar hosts virtuais;

Requisitos:

- Ativar a virtualização na BIOS ou no S.O caso necessário;
- Instalar o programa do Docker na máquina;

Instalação:

Recomendo seguir esse link para a instalação:

- https://docs.docker.com/get-docker/

Ambiente:

Versão do S.O utilizado (Debian 10):

```
Linux version 4.19.0-10-amd64 (debian-kernel@lists.debian.org) (gcc version 8.3.0
(Debian 8.3.0-6)) #1 SMP Debian 4.19.132-1 (2020-07-24)
```

Versão do Docker utilizado:

```
Docker version 18.09.1, build 4c52b90
```

Ajuda

Com certeza essa é o comando mais amigo que você vai achar:

```
docker --help
docker-compose --help
```

Há, se você quer saber como tal comando funciona, você só precisa como um `--help` na frente, exemplo:

```
docker run --help
```

O resultado vai ser a tela de instruções e parametros do comando.

---

root@debian: docker

Abaixo está a saída de paramentros para ações no Docker, digitar `Docker` ou `docker --help`, gera a mesma saída de comando, Ex:

```
root@debian:~# docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default
"/root/.docker")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level
("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
"/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default
"/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  engine      Manage the docker engine
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running
container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
```

```
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by
default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.
```

Não pretendo falar de todos os comandos e sim dos que mais utilizo em prática, esse manual será atualizado com o tempo para a adição de alterações e reformas, nesses tempos, se necessário, será criada explicações e exemplos de comandos ainda não listados.

Informação do Docker

Listar a versão do docker:

```
root@debian:~# docker version
Client:
 Version:           18.09.1
 API version:       1.39
 Go version:        go1.11.6
 Git commit:        4c52b90
```

```
 Built:             Sun, 14 Jun 2020 22:12:29 +0200
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          18.09.1
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.11.6
  Git commit:       4c52b90
  Built:            Sun Jun 14 20:12:29 2020
  OS/Arch:          linux/amd64
  Experimental:     false
```

Ou de forma resumida:

```
root@debian:~# docker --version
Docker version 18.09.1, build 4c52b90
```

O comando `docker info` é um comando que lista as caracteristicas do docker instalado no **HOST**:

```
root@debian:~# docker info
Containers: 1
 Running: 0
 Paused: 0
 Stopped: 1
Images: 3
Server Version: 18.09.1
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: 9754871865f7fe2f4e74d43e2fc7ccd237edcbce
runc version: 1.0.0~rc6+dfsg1-3
init version: v0.18.0 (expected: fec3683b971d9c3ef73f284f176672c44b448662)
Security Options:
 apparmor
 seccomp
```

```
  Profile: default
Kernel Version: 4.19.0-10-amd64
Operating System: Debian GNU/Linux 10 (buster)
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 987.4MiB
Name: debian
ID: SS4R:AJ7J:TSTN:6AJL:UD2I:6P6Y:WDWA:V6T3:7UXA:G3UU:L7P4:7P4P
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
WARNING: No swap limit support
```

Iniciando no Docker

Iniciando a demonstração da parte prática do Docker.

Olha o hello world

Executando um hello world:

```
docker run hello-world
```

Explicando:

docker	run	hello-world
Chamando o programa	Parametro para ação, no caso um executar	Imagem que quero executar

OBS: Quando você executar o docker vai procurar o container se ele existe na máquina, caso não achar ele baixa o container automaticamente do docker hub.

Outra coisa, podemos decidir que versão baixar de determinado container, como exemplo:

Ultima versão do Ubuntu:

```
docker run ubuntu
```

Ou podemos apontar a versão dele como a ultima assim:

```
docker run ubuntu:latest
```

Ou podemos usar uma versão antiga do Ubuntu:

```
docker run ubuntu:xenial
```

A decisão da versão a se trabalhar como base é a cargo da pessoa que decide montar o ambiente, o DockerHub possui algumas versões de determinados S.O prontas que podem ser ou não de distros oficiais, no caso do Ubuntu, é uma distro mantida pela própria Canonical.

A execução de um container, no caso o latest ubuntu:

```
root@teste-teste:/home/guilherme# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest
```

Explicando:

- **Comando** -> root@teste-teste:/home/guilherme# docker run hello-world
- **Procura interna** -> Unable to find image 'hello-world:latest' locally
- **Achando no docker hub** -> latest: Pulling from library/hello-world
- **Download no hello-word** ->
    - 1b930d010525: Pull complete
    - Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
- **Status** -> Status: Downloaded newer image for hello-world:latest

A saída da criação do container docker hello-world:

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

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

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

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

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

Agora é a hora dos **MÁS!**

Status

Depois de executar um **RUN** e ter sua saída, o que acontece depois? Para isso execute o comando:

```
docker ps
```

Após você executar o mesmo, terá a bela surpresa dessa tela:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Está vazia e você provavelmente não entende o que é essa tela, quando se utilizado o comando `docker ps`, nós é listados todos os containers em ativo no momento atual, isso é, sabe o `hello-world`, ele já não está mais ativo, para vermos ele podemos executar o:

```
docker ps -a
```

A saída desse comando deve gerar algo parecido com isso:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
06ea4331b9d9        hello-world         "/hello"            14 minutes ago
Exited (0) 14 minutes ago                       optimistic_tharp
```

Explicando esse menu:

- **CONTAINER ID** -> Numero do container, nome dele, sua identificação;
- **IMAGE** -> Container é baseado a qual imagem;
- **COMMAND** -> Comando que inicia o container;
- **CREATED** -> A que horas foi criado o container;

- **STATUS** -> O estado do container;
- **PORTS** -> Portas que o container está utilizando para saída;
- **NAMES** -> Nome RANDOM que o Docker dá para o container criado;

Agora explicando o docker ps:

Docker ps é um comando para listar os container's, o mesmo possui parametros para se realizar filtros e demais, Ex:

- **docker ps** -> Somente lista os containers em funcionamento;
- **docker ps -a** -> Lista todos os containers existentes nos host, não importando seus status;

Agora temos mais algums opções que podemos ver com o `docker ps --help`:

```
root@debian:~# docker ps --help

Usage:  docker ps [OPTIONS]

List containers

Options:
  -a, --all             Show all containers (default shows just running)
Mostrar todos os contêineres (o padrão mostra apenas em execução)
  -f, --filter filter   Filter output based on conditions provided
Filtre a saída com base nas condições fornecidas
      --format string   Pretty-print containers using a Go template
Recipientes com impressão bonita usando um modelo Go
  -n, --last int        Show n last created containers (includes all states)
(default -1)     Mostrar n últimos contêineres criados (inclui todos os estados)
(padrão -1)
  -l, --latest          Show the latest created container
includes all states
      --no-trunc        Don't truncate output
Não truncar a saída
  -q, --quiet           Only display numeric IDs
Exibir apenas IDs numéricos
  -s, --size            Display total file sizes
Exibir o tamanho total dos arquivos
```

Tradução: Google Tradutor, Windows tradutor ou meu próprio inglês;

Agora, executando um novo container

redirect_to: /foo/

Para se executar um novo container, pode-se utilizar o mesmo `docker run hello-world`, ele vai criar um novo container sobre a imagem já utilizada, subir, executar sua função e fechar, ou se pode subir um novo container diferente, ex:

```
docker run ubuntu echo "hello world"
```

Más se você executar esse cara, você vai ter o mesmo resultado do `hello-world`, ele executa e some, exemplo:

```
root@debian:~# docker run ubuntu echo "hello world"
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
54ee1f796a1e: Pull complete
f7bfea53ad12: Pull complete
46d371e02073: Pull complete
b66c17bbf772: Pull complete
Digest: sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Status: Downloaded newer image for ubuntu:latest
hello world
```

Sobre o que você está vendo eu vou explicar em parte, primeiramente, você precisa ter em mente o conceito de camadas que o Docker utiliza para execução de suas imagens... Opa, acho que também nem expliquei o que é imagens né... Tá, você já sentiu o gostinho do Docker, tá na hora de você sair mais sobre ele.

O conceito de camadas

O docker funciona sobre um sistema de camadas de imagens, alimentadas por um sistema de PULL, quanto mais completa for a imagem mais pull's ela vai precisar ter, um exemplo disso é a imagem **ubuntu** que forá executado acima, ela precisou de 4 pull's para seu funcionamento, já o **hello-world** necessitou de somente 1 pull.

Cada pull é uma camada, essa uma camada é parte do funcionamento de uma imagem container e a mesma pode ser vária, por exemplo, posso baixar uma camada de web server e demais outros, agora um exemplo legal de como o Docker trata cada container como isolado, más não as camadas que foram as imagens:

1° Container:

- Precisou baixar o container que possue 5 camadas, isso é 5 pulls;

2º Container:

- Agora por certos motivos preciso subir outro container para a demanda XYZ, eu dou o comando e é outras 5 camadas, más só foram realizados 2 pulls, porquê?

Esse é o conceito de imagens e pulls do Docker, todas as imagens compartilham suas camadas umas com as outros dentro do HOST, isso faz que não seja necessário fazer o download novamente de uma camada, já que a mesma já existe no HOST.

Leitor: Nossa, sério, só isso, esse textão todo só prá isso?

Eu: Calma, ainda tem mais uma coisinha interessante que é necessário para se trabalhar com o Docker e seus containers.

Somente leitura

Como dito o Docker trabalha sobre um sistema de download de camadas para montar uma imagem, quando o mesmo existe ele evita o download para evitar "gastos" denecessários para ambos os lados, porém a forma de manter a integridade dessas camadas é as travando, isso é, elas são só para leitura e não para a escrita, quando o docker precisar realizar alterações, ele cria uma camada acima das atuais e inicia as alterações nela.

Más o que isso me afeta?

Bem, vamos supor que você subiu um container, fez todas as alterações e depois você saiu do container ou ele parou de funcionar... VOCÊ PERDEU TUDO O QUE FEZ!!!! Sério, como dito, ele cria uma camada para escrita sobre as de leitura, porém, ele não salva essa camada, assim é necessário executar um `build` sobre a imagem atual para criar a imagem com essas alterações **personalizadas**.

O nome dessa camada é **Read/write layer**, um container é formado de pulls que são camadas **READ** e quando executados como um container **RUN**, se cria uma camada **WRITE** para o usuário se trabalhar.

Descanso

Se você leu até aqui, entenda o que você acabou de ler e descance um pouco, esses conceitos são simples, porém são fundamentais para se usar o Docker.

Voltando

Agora com o conceito de imagens e pull's firmado, vamos a mais alguns comandinhos.

Dentro de um container

Para executar um container já vimos que é simples:

```
docker run <imagem>
```

Más e se queremos interagir com ela? Podemos fazer tambem de uma forma simples, Exemplo:

```
docker run <imagem> <comando>
```

OU

```
docker run ubuntu echo "teste"
```

Com a saída:

```
root@debian:~# docker run ubuntu echo "teste"
teste
```

Incrivél!!!! Não, ata, sei, quer mais interação, então vá para dentro do container, só que antes de tudo **NÃO SE DESESPERE NÃO DE UM EXIT OU SHUTDOWN PARA SAIR DO CONTAINER** só continue a ler.

Como interagir com o container:

```
docker run -it <container>
```

Exemplo:

```
root@debian:~# docker run -it ubuntu
root@5d4a35d4e367:/#
```

Já estou dentro do container, acredita não, pera:

Debian HOST:

```
root@debian:~# ls /
bin  boot  dev  etc  home  initrd.img  initrd.img.old  lib  lib32  lib64  libx32
lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
vmlinuz  vmlinuz.old
```

Ubuntu container:

```
root@2b839d07bfab:/# ls /
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root
run  sbin  srv  sys  tmp  usr  var
```

Vê alguma diferença? Com isso podemos notar que ambos os ambientes são separados, isso traz segurança a você pois sabe que um não afeta o outro, pelo menos não nessa fase inicial.

Não consegue sair né?

Como todas as pessoas que usaram **VI** ou **VIM** pela primeira vez na vida, você não sabe como sair desse container, **BEM NÃO FAÇA ISSO AQUI NÃO**:

```
root@2b839d07bfab:/# exit
exit
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Não entendeu o que acabou de acontecer, bem, como explicado em imagens, você acabou de matar seu container, por esse motivo ele não mostra no `docker ps`, exemplo, se você tem um container funcionando, a saída desse `docker ps` será assim:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
5d4a35d4e367        ubuntu              "/bin/bash"         18 minutes ago      Up
39 seconds                          competent_robinson
```

Pera, voltando ao problema

Para sair do container atual, aperte:

```
ctrl + p + q
```

Dessa forma, você saiu do container, más ele ainda está executando, se você quiser retornar para dentro dele, faça:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
5d4a35d4e367        ubuntu              "/bin/bash"         21 minutes ago      Up
3 minutes                           competent_robinson
root@debian:~# docker attach 5d4a35d4e367
root@5d4a35d4e367:/#
```

Más o que foi isso tudo:

- CTRL+D+Q deixa você sair do container atual e voltar para seu HOST;
- `docker attach` é se atrelar a determinado container em execução;

Com essas duas opções você consegue entrar e sair de um container sem afetar o funcionamento do mesmo.

Pulo do gato no RUN

Existe formas de manter um container executando, sem precisar se atrelar ao mesmo, esse é o `-d` ou modo detached, o container ainda continua funcionado, porém você se mantém no seu terminal;

Agora para dar um pequeno susto, olhe o `--help` de um `docker run`:

Original:

```
root@debian:~# docker run --help

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:
      --add-host list                  Add a custom host-to-IP mapping (host:ip)
  -a, --attach list                    Attach to STDIN, STDOUT or STDERR
      --blkio-weight uint16            Block IO (relative weight), between 10 and
1000, or 0 to disable (default 0)
      --blkio-weight-device list       Block IO weight (relative device weight)
(default [])
      --cap-add list                   Add Linux capabilities
      --cap-drop list                  Drop Linux capabilities
      --cgroup-parent string           Optional parent cgroup for the container
      --cidfile string                 Write the container ID to the file
      --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler)
period
      --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler)
quota
      --cpu-rt-period int              Limit CPU real-time period in microseconds
      --cpu-rt-runtime int             Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int                 CPU shares (relative weight)
      --cpus decimal                   Number of CPUs
      --cpuset-cpus string             CPUs in which to allow execution (0-3, 0,1)
      --cpuset-mems string             MEMs in which to allow execution (0-3, 0,1)
  -d, --detach                         Run container in background and print
container ID
      --detach-keys string             Override the key sequence for detaching a
container
      --device list                    Add a host device to the container
      --device-cgroup-rule list        Add a rule to the cgroup allowed devices
list
      --device-read-bps list           Limit read rate (bytes per second) from a
device (default [])
      --device-read-iops list          Limit read rate (IO per second) from a
device (default [])
      --device-write-bps list          Limit write rate (bytes per second) to a
device (default [])
      --device-write-iops list         Limit write rate (IO per second) to a
device (default [])
      --disable-content-trust          Skip image verification (default true)
      --dns list                       Set custom DNS servers
      --dns-option list                Set DNS options
      --dns-search list                Set custom DNS search domains
```

```
      --entrypoint string              Overwrite the default ENTRYPOINT of the
image
  -e, --env list                       Set environment variables
      --env-file list                  Read in a file of environment variables
      --expose list                    Expose a port or a range of ports
      --group-add list                 Add additional groups to join
      --health-cmd string              Command to run to check health
      --health-interval duration       Time between running the check (ms|s|m|h)
(default 0s)
      --health-retries int             Consecutive failures needed to report
unhealthy
      --health-start-period duration   Start period for the container to
initialize before starting health-retries countdown (ms|s|m|h) (default 0s)
      --health-timeout duration        Maximum time to allow one check to run
(ms|s|m|h) (default 0s)
      --help                           Print usage
  -h, --hostname string                Container host name
      --init                           Run an init inside the container that
forwards signals and reaps processes
  -i, --interactive                    Keep STDIN open even if not attached
      --ip string                      IPv4 address (e.g., 172.30.100.104)
      --ip6 string                     IPv6 address (e.g., 2001:db8::33)
      --ipc string                     IPC mode to use
      --isolation string               Container isolation technology
      --kernel-memory bytes            Kernel memory limit
  -l, --label list                     Set meta data on a container
      --label-file list                Read in a line delimited file of labels
      --link list                      Add link to another container
      --link-local-ip list             Container IPv4/IPv6 link-local addresses
      --log-driver string              Logging driver for the container
      --log-opt list                   Log driver options
      --mac-address string             Container MAC address (e.g.,
92:d0:c6:0a:29:33)
  -m, --memory bytes                   Memory limit
      --memory-reservation bytes       Memory soft limit
      --memory-swap bytes              Swap limit equal to memory plus swap: '-1'
to enable unlimited swap
      --memory-swappiness int          Tune container memory swappiness (0 to 100)
(default -1)
      --mount mount                    Attach a filesystem mount to the container
      --name string                    Assign a name to the container
      --network string                 Connect a container to a network (default
"default")
      --network-alias list             Add network-scoped alias for the container
      --no-healthcheck                 Disable any container-specified HEALTHCHECK
      --oom-kill-disable               Disable OOM Killer
      --oom-score-adj int              Tune host's OOM preferences (-1000 to 1000)
      --pid string                     PID namespace to use
      --pids-limit int                 Tune container pids limit (set -1 for
unlimited)
      --privileged                     Give extended privileges to this container
  -p, --publish list                   Publish a container's port(s) to the host
  -P, --publish-all                    Publish all exposed ports to random ports
      --read-only                      Mount the container's root filesystem as
```

```
read only
      --restart string                Restart policy to apply when a container
exits (default "no")
      --rm                            Automatically remove the container when it
exits
      --runtime string                Runtime to use for this container
      --security-opt list             Security Options
      --shm-size bytes                Size of /dev/shm
      --sig-proxy                     Proxy received signals to the process
(default true)
      --stop-signal string            Signal to stop a container (default
"SIGTERM")
      --stop-timeout int              Timeout (in seconds) to stop a container
      --storage-opt list              Storage driver options for the container
      --sysctl map                    Sysctl options (default map[])
      --tmpfs list                    Mount a tmpfs directory
  -t, --tty                           Allocate a pseudo-TTY
      --ulimit ulimit                 Ulimit options (default [])
  -u, --user string                   Username or UID (format: <name|uid>[:
<group|gid>])
      --userns string                 User namespace to use
      --uts string                    UTS namespace to use
  -v, --volume list                   Bind mount a volume
      --volume-driver string          Optional volume driver for the container
      --volumes-from list             Mount volumes from the specified
container(s)
  -w, --workdir string                Working directory inside the container
```

Traduzido:

```
root@debian:~# docker run --help

Uso: docker executar [OPÇÕES] IMAGEM [COMANDO] [ARG...]

Execute um comando em um novo contêiner

Opções:
      -lista de host adicionais Adicione um mapeamento personalizado de host-to-IP
(host:ip)
  -a, --anexar lista Anexar a STDIN, STDOUT ou STDERR
      --blkio-peso uint16 Bloco IO (peso relativo), entre 10 e 1000, ou 0 para
desativar (padrão 0)
      --blkio-peso-dispositivo lista Peso de IO de bloco (peso relativo do
dispositivo) (padrão [])
      --lista de adicionar tampas Adicionar recursos do Linux
      --lista de drop-drop recursos do Drop Linux
      --cgroup-parent cgroup Cgroup Optional parent cgroup for the container
      --string de cidfile Escreva o ID do contêiner para o arquivo
      Período de CPU limite do período -cpu int (Agendar completamente justo)
período
      --cpu-cota int limitar a cota CFS (Completely Fair Scheduler)
```

```
      --cpu-rt-period int Período de tempo real da CPU em microsegundos
      --cpu-rt-runtime int Limitar o tempo de execução da CPU em tempo real em
microsegundos
  -c, --cpu-ações int CPU ações (peso relativo)
      --cpus número decimal de CPUs
      --cpuset-cpus cpus cpus em que permitir a execução (0-3, 0,1)
      --cpuset-mems string MEMs em que permitir a execução (0-3, 0,1)
  --desprender-desprender Recipiente de execução em fundo e imprimir iD do
recipiente
      --desprender-teclas sequência Anular a sequência de teclas para desapegar um
recipiente
      -lista de dispositivos Adicione um dispositivo host ao contêiner
      -lista de regras de cgroup-dispositivo Adicionar uma regra à lista de
dispositivos permitidos pelo cgroup
      --lista de leitura de dispositivos-bps Taxa de leitura limite (bytes por
segundo) de um dispositivo (padrão [])
      --lista de leitura de dispositivos-iops Taxa de leitura limite (IO por
segundo) de um dispositivo (padrão [])
      -lista de gravação-gravação-bps -Taxa de gravação limite (bytes por segundo)
para um dispositivo (padrão [])
      --lista de gravação-iops do dispositivo -taxa de gravação limite (IO por
segundo) para um dispositivo (padrão [])
      -desativar-desativar-confiança de conteúdo Pular verificação de imagem
(padrão verdadeiro)
      Lista -dns Definir servidores DNS personalizados
      Lista de opções -dns Definir opções DNS
      --dns-lista de pesquisa Definir domínios de pesquisa DNS personalizados
      --entrada string Substitua o ENTRYPOINT padrão da imagem
  -e, --env lista Definir variáveis de ambiente
      --lista de arquivos env Leia em um arquivo de variáveis de ambiente
      -expor lista Expor uma porta ou uma gama de portas
      -lista de adoção de grupo Adicione grupos adicionais para participar
      --comando de cordas de cmd de saúde para executar para verificar a saúde
      --duração do intervalo de saúde Tempo entre executar a verificação
(ms|s|m|h) (padrão 0s)
      --problemas de saúde int Falhas consecutivas necessárias para relatar
insalubridade
      --duração do início da saúde-período de início para que o recipiente seja
inicializado antes de iniciar a contagem regressiva de reexame de saúde (ms|s|m|h)
(padrão 0s)
      --duração do tempo limite de saúde Tempo máximo para permitir que uma
verificação seja executada (ms|s|m|h) (padrão 0s)
      --ajudar o uso da impressão
  -h, --hostname string Container nome do host
      --init Executar um init dentro do recipiente que encaminha sinais e colhe
processos
  -i, --interativo Manter STDIN aberto mesmo que não anexado
      Endereço IPv4 de string ip (por exemplo, 172.30.100.104)
      --ip6 string IPv6 (por exemplo, 2001:db8::33)
      --modo IPC string IPC para usar
      --isolar a tecnologia de isolamento de contêineres
      --kernel-memória bytes Limite de memória kernel
  -l, --lista de rótulos Definir meta dados em um contêiner
      -lista de arquivos de rótulos Leia em uma linha de arquivo delimitado de
```

```
rótulos
      Lista de links -Adicionar link a outro contêiner
      -link-local-ip lista Contêiner IPv4/IPv6 endereços locais de link
      --driver de sequência de driver de registro driver para o contêiner
      --log-opt-opt list Registre opções de driver
      Endereço MAC-end string Container MAC (por exemplo, 92:d0:c6:0a:29:33)
  -m, --memória bytes limite de memória
      --memória-reserva bytes Limite suave de memória
      --troca de memória Limite de troca igual à memória mais swap: '-1' para
ativar swap ilimitado
      --troca de memória int Sintonize a troca de memória do recipiente (0 a 100)
(padrão -1)
      -Montaria Anexar um suporte de sistema de arquivos ao recipiente
      --string nome Atribuir um nome ao recipiente
      --cadeia de rede Conecte um contêiner a uma rede (padrão "padrão")
      --lista de alias de rede Adicionar alias com escopo de rede para o contêiner
      --verificação sem saúde Desativar qualquer HEALTHCHECK especificado por
contêiner
      -oom-kill-desativar Disable OOM Killer
      -oom-score-adj int Tune preferências OOM do host (-1000 a 1000)
      --pid string PID namespace para usar
      --pids-limit int Tune container pids limite (definir -1 para ilimitado)
      --privilegiado Dar privilégios estendidos a este contêiner
  -p, --publicar lista Publicar porta(s) de um contêiner para o host
  -P, --publicar todas publica todas as portas expostas em portas aleatórias
      -leia-somente montar o sistema de arquivos raiz do recipiente como lido
apenas
      -Reiniciar a política de reinicialização da sequência de sequências para
aplicar quando um contêiner sair (padrão "não")
      -rm Remova automaticamente o recipiente quando ele sair
      --tempo de execução Tempo de execução Tempo de execução Tempo para usar para
este recipiente
      -Opções de segurança optam por segurança
      --shm-size bytes Tamanho de /dev/shm
      --sig-proxy Proxy recebeu sinais para o processo (padrão verdadeiro)
      Sinal de sequência de sinal de parada para parar um contêiner (padrão
"SIGTERM")
      --stop-timeout int Timeout (em segundos) para parar um contêiner
      Lista de armazenamento opção Opções de driver de armazenamento para o
contêiner
      --opções sysctl map Sysctl (mapa padrão[])
      --lista tmpfs Montar um diretório tmpfs
  -t, --tty Alocar um pseudo-TTY
      --opções ulimit ulimit Ulimit Ulimit (padrão [])
  -u, --string de usuário Nome de usuário ou UID (formato: <nome|uid>[:
<group|gid>])
      --usuários string Namespace de usuário para usar
      -uts string UTS namespace para usar
  -v, --lista de volumes Vincular montar um volume
      --cadeia de driver de volume Driver opcional para o contêiner
      -volumes -da lista Volumes de montagem do contêiner especificado(s)
  -w, --workdir string Diretório de trabalho dentro do contêiner
```

Notemos que existem vários parametros que permitem configurações personalizadas para os mesmo.

Attach

Attach ou anexe é uma opção que permite você a se atrelar dentro de um container.

Original:

```
Usage:  docker attach [OPTIONS] CONTAINER

Attach local standard input, output, and error streams to a running container

Options:
      --detach-keys string   Override the key sequence for detaching a container
      --no-stdin             Do not attach STDIN
      --sig-proxy            Proxy all received signals to the process (default
true)


Uso: docker attach [OPTIONS] CONTAINER
```

Traduzido:

```
Uso: docker attach [OPTIONS] CONTAINER

Anexe entrada, saída e fluxos de erro padrão locais a um contêiner em execução

Opções:
      --detach-keys string Substitui a sequência de teclas para desanexar um
contêiner
      --no-stdin Não anexar STDIN
      --sig-proxy Proxy todos os sinais recebidos para o processo (padrão
verdadeiro)

Uso: docker attach [OPTIONS] CONTAINER
```

Lembra do ps -a?

Então, se você der o comando `docker ps -a`, ele vai listar todos os containers que você já criou, más com qual finalidade isso? Bem, você pode subir esses antigos denovo, exemplo:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
c7acf3635420        4e2eef94cd6b        "/bin/bash"         3 minutes ago
Exited (0) 7 seconds ago                        dreamy_archimedes
```

```
1f2c62f99928        4e2eef94cd6b        "echo teste"        4 minutes ago
Exited (0) 3 minutes ago                        nostalgic_ardinghelli
21e8388f42a6        4e2eef94cd6b        "echo teste"        4 minutes ago
Exited (0) 4 minutes ago                        nervous_boyd
b52f131632dc        4e2eef94cd6b        "/bin/bash"         5 minutes ago
Exited (127) 5 minutes ago                      adoring_saha
778125f73d75        4e2eef94cd6b        "/bin/bash"         5 minutes ago
Exited (0) 5 minutes ago                        hungry_blackwell
```

Para fazer um desses container's voltar a funcionar, faça:

```
docker start <CONTAINER ID>
```

O **START** vai fazer o container voltar a funcionar, más depende muito do tipo do container também, exemplo:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
c7acf3635420        4e2eef94cd6b        "/bin/bash"         7 minutes ago
Exited (0) 3 minutes ago                        dreamy_archimedes
1f2c62f99928        4e2eef94cd6b        "echo teste"        7 minutes ago
Exited (0) 6 minutes ago                        nostalgic_ardinghelli
21e8388f42a6        4e2eef94cd6b        "echo teste"        8 minutes ago
Exited (0) 8 minutes ago                        nervous_boyd
b52f131632dc        4e2eef94cd6b        "/bin/bash"         8 minutes ago
Exited (127) 8 minutes ago                      adoring_saha
778125f73d75        4e2eef94cd6b        "/bin/bash"         8 minutes ago
Exited (0) 8 minutes ago                        hungry_blackwell
root@debian:~# docker start 1f2c62f99928
1f2c62f99928
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
c7acf3635420        4e2eef94cd6b        "/bin/bash"         7 minutes ago
Exited (0) 3 minutes ago                        dreamy_archimedes
1f2c62f99928        4e2eef94cd6b        "echo teste"        7 minutes ago
Exited (0) 8 seconds ago                        nostalgic_ardinghelli
21e8388f42a6        4e2eef94cd6b        "echo teste"        8 minutes ago
Exited (0) 8 minutes ago                        nervous_boyd
b52f131632dc        4e2eef94cd6b        "/bin/bash"         8 minutes ago
Exited (127) 8 minutes ago                      adoring_saha
778125f73d75        4e2eef94cd6b        "/bin/bash"         9 minutes ago
```

```
Exited (0) 9 minutes ago                         hungry_blackwell
```

O que aconteceu? Eu mandei executar um container que que estava parado, o **1f2c62f99928**, o resultado foi que ele realmente foi executado, podemos ver pelo **STATUS** dele que a ultima vez que ele iniciou foi a 8 segundos atrás, más ele não mostrou nada né, tem um motivo, não pedimos para ele fazer isso, quer ver?

Original:

```
root@debian:~# docker start --help

Usage:  docker start [OPTIONS] CONTAINER [CONTAINER...]

Start one or more stopped containers

Options:
  -a, --attach               Attach STDOUT/STDERR and forward signals
      --detach-keys string   Override the key sequence for detaching a container
  -i, --interactive          Attach container's STDIN
```

Traduzido:

```
root @ debian: ~ # docker start --help

Uso: docker start [OPÇÕES] CONTAINER [CONTAINER ...]

Inicie um ou mais contêineres parados

Opções:
  -a, --attach Anexar STDOUT / STDERR e sinais de encaminhamento
      --detach-keys string Substitui a sequência de teclas para desanexar um
contêiner
  -i, --interactive Anexar STDIN do contêiner
```

Se usarmos a opção `-a` durante o start, podemos ter a saída em terminal do comando echo ordenado ao container, veja:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
c7acf3635420        4e2eef94cd6b        "/bin/bash"         19 minutes ago
Exited (0) 15 minutes ago                               dreamy_archimedes
1f2c62f99928        4e2eef94cd6b        "echo teste"        19 minutes ago
Exited (0) 12 minutes ago                               nostalgic_ardinghelli
21e8388f42a6        4e2eef94cd6b        "echo teste"        20 minutes ago
Exited (0) 20 minutes ago                               nervous_boyd
```

```
b52f131632dc        4e2eef94cd6b        "/bin/bash"         21 minutes ago
Exited (127) 20 minutes ago                         adoring_saha
778125f73d75        4e2eef94cd6b        "/bin/bash"         21 minutes ago
Exited (0) 21 minutes ago                           hungry_blackwell
root@debian:~# docker start -a 1f2c62f99928
teste
```

Ou podemos usar a opção `-i` para entrar de modo interativo num container como o **c7acf3635420**, veja:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
c7acf3635420        4e2eef94cd6b        "/bin/bash"         23 minutes ago
Exited (130) 3 seconds ago                          dreamy_archimedes
1f2c62f99928        4e2eef94cd6b        "echo teste"        23 minutes ago
Exited (0) 3 minutes ago                            nostalgic_ardinghelli
21e8388f42a6        4e2eef94cd6b        "echo teste"        23 minutes ago
Exited (0) 23 minutes ago                           nervous_boyd
b52f131632dc        4e2eef94cd6b        "/bin/bash"         24 minutes ago
Exited (127) 24 minutes ago                         adoring_saha
778125f73d75        4e2eef94cd6b        "/bin/bash"         24 minutes ago
Exited (0) 24 minutes ago                           hungry_blackwell
root@debian:~# docker start -i c7acf3635420
root@c7acf3635420:/#
```

Más agora que foi falado no **START**, acho que devo comentar sobre comandos o **PAUSE** e **STOP**, notemos pelo exemplo a seguir a diferença de um **STOP** para um **PAUSE**:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         4 seconds ago       Up
3 seconds                               objective_easley
root@debian:~# docker pause 19490514cfd1
19490514cfd1
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         2 minutes ago       Up
About a minute (Paused)                         objective_easley
root@debian:~# docker unpause 19490514cfd1
19490514cfd1
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         2 minutes ago       Up
2 minutes                               objective_easley
root@debian:~# docker stop 19490514cfd1
19490514cfd1
```

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         2 minutes ago
Exited (0) 4 seconds ago                        objective_easley
e3299d35d1c6        4e2eef94cd6b        "/bin/bash"         3 minutes ago
Exited (0) 3 minutes ago                        condescending_hodgkin
6e1a9c33eed1        ubuntu:xenial       "/bin/bash"         13 minutes ago
Exited (0) 13 minutes ago                       awesome_panini
362828e0cead        ubuntu:latest       "/bin/bash"         17 minutes ago
Exited (0) 17 minutes ago                       admiring_nightingale
```

Quando executamos os **PAUSE** ele para o container e permite sua reativação pelo `UNPAUSE`, já o **STOP** para o mesmo, mandando-o para os containers em inativo e neste caso, tem que dar um **START** para ele voltar.

Note a diferença:

STOP:

```
root@debian:~# docker stop
"docker stop" requires at least 1 argument.
See 'docker stop --help'.

Usage:  docker stop [OPTIONS] CONTAINER [CONTAINER...]

Stop one or more running containers
```

- Pare a execução de um ou mais containers;

O **HELP** no **STOP** resulta em:

```
root@debian:~# docker stop --help

Usage:  docker stop [OPTIONS] CONTAINER [CONTAINER...]

Stop one or more running containers

Options:
-t, --time int   Seconds to wait for stop before killing it (default 10)
```

- Basicamente lhe permite definir um tempo em segundo antes de matar o container, exemplo disso é:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
```

```
STATUS              PORTS               NAMES
7c26f32bcb5d        ubuntu              "/bin/bash"         44 seconds ago      Up
43 seconds                              zealous_bardeen
root@debian:~# docker stop -t 10 7c26f32bcb5d
7c26f32bcb5d
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
7c26f32bcb5d        ubuntu              "/bin/bash"         About a minute ago
Exited (0) 17 seconds ago                               zealous_bardeen
```

PAUSE:

```
root@debian:~# docker pause
"docker pause" requires at least 1 argument.
See 'docker pause --help'.

Usage:  docker pause CONTAINER [CONTAINER...]

Pause all processes within one or more containers
```

- Pause todos os processos de um ou mais containers

O **HELP** em **PAUSE** não resulta em nada diferente, porém é valido completar que quando um container está em pause, as formas de retormar o mesmo são feitas por:

UNPAUSE:

```
docker unpause <ID container>
```

Há, só pra avisar, não funciona dar um **START** e/ou **ATTACH** em um container pausado, ele manda dar um **UNPAUSE** para assim o container voltar a executar, veja:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
7c26f32bcb5d        ubuntu              "/bin/bash"         5 minutes ago       Up
3 minutes                               zealous_bardeen
root@debian:~# docker pause 7c26f32bcb5d
7c26f32bcb5d
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                  PORTS               NAMES
7c26f32bcb5d        ubuntu              "/bin/bash"         6 minutes ago       Up
3 minutes (Paused)                       zealous_bardeen
root@debian:~# docker start 7c26f32bcb5d
Error response from daemon: cannot start a paused container, try unpause instead
```

```
Error: failed to start containers: 7c26f32bcb5d
root@debian:~# docker attach 7c26f32bcb5d
You cannot attach to a paused container, unpause it first
root@debian:~# docker unpause 7c26f32bcb5d
7c26f32bcb5d
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
7c26f32bcb5d        ubuntu              "/bin/bash"         6 minutes ago       Up
3 minutes                           zealous_bardeen
```

Ordenar sem interagir diretamente

O comando `exec` permite executar funções dentro de um container sem necessitar interagir diretamente com ele, como mostrado antes, podemos entrar dentro do container pelo modo interativo para executar comandos, o `exec` evita esse trabalho:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         23 minutes ago      Up
13 minutes                          objective_easley
root@debian:~# docker exec 19490514cfd1 echo "teste"
teste
```

O que o **HELP** desse cara nos diz:

Original:

```
root@debian:~# docker exec --help

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

Options:
  -d, --detach               Detached mode: run command in the background
      --detach-keys string   Override the key sequence for detaching a container
  -e, --env list             Set environment variables
  -i, --interactive          Keep STDIN open even if not attached
      --privileged           Give extended privileges to the command
  -t, --tty                  Allocate a pseudo-TTY
  -u, --user string          Username or UID (format: <name|uid>[:<group|gid>])
  -w, --workdir string       Working directory inside the container
```

Traduzido:

```
root @ debian: ~ # docker exec --help

Uso: docker exec [OPÇÕES] COMANDO DE CONTAINER [ARG ...]

Execute um comando em um contêiner em execução

Opções:
  -d, --detach Modo separado: executa o comando em segundo plano
      --detach-keys string Substitui a sequência de teclas para desanexar um
contêiner
  -e, --env list Definir variáveis de ambiente
  -i, --interactive Mantém STDIN aberto mesmo se não estiver conectado
      --privileged Concede privilégios estendidos ao comando
  -t, --tty Aloca um pseudo-TTY
  -u, --user string Nome de usuário ou UID (formato: <nome | uid> [: <grupo |
gid>])
  -w, --workdir string Diretório de trabalho dentro do contêiner
```

Como deletar as coisas

Para ser mais preciso, o que pode ser deletado? O que é possível deletar? Melhor ainda, e você der um `docker ps -a` você deva estar vendo muitos containers parados e que possívemente você não vai mais usar, além de mais coisas, vamos detalhar esse ponto:

Para deletar um container que não está mais em uso, podemos executar:

```
docker rm <CONTAINER ID>
```

Exemplo:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         About an hour ago   Up
About an hour                                   objective_easley
e3299d35d1c6        4e2eef94cd6b        "/bin/bash"         About an hour ago
Exited (0) About an hour ago                    condescending_hodgkin
6e1a9c33eed1        ubuntu:xenial       "/bin/bash"         About an hour ago
Exited (0) About an hour ago                    awesome_panini
362828e0cead        ubuntu:latest       "/bin/bash"         About an hour ago
Exited (0) About an hour ago                    admiring_nightingale
root@debian:~# docker rm 362828e0cead
362828e0cead
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         About an hour ago   Up
```

```
About an hour                                    objective_easley
e3299d35d1c6        4e2eef94cd6b        "/bin/bash"         About an hour ago
Exited (0) About an hour ago                        condescending_hodgkin
6e1a9c33eed1        ubuntu:xenial       "/bin/bash"         About an hour ago
Exited (0) About an hour ago                        awesome_panini
```

Tambem se pode usar o delete para um ID container resumido, exemplo:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
bb72e2b64cd5        ubuntu              "/bin/bash"         11 minutes ago      Up
11 minutes                            tender_hofstadter
root@debian:~# docker rm -f bb72
bb72
```

De qualquer forma, notemos que o container alvo foi eliminado, más e se tentar-mos com um em execução:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
bb72e2b64cd5        ubuntu              "/bin/bash"         9 minutes ago       Up
9 minutes                             tender_hofstadter
root@debian:~# docker rm bb72e2b64cd5
Error response from daemon: You cannot remove a running container
bb72e2b64cd56340204de475d8c5b1d6b1cc2f75e0f28f7ce7a088b164d3d99f. Stop the
container before attempting removal or force remove
```

Ele vai responder de forma negativa, a ação não pode ser realizada dessa forma, para se ter maior idéia, vamos ver o que **HELP** desse cara nos diz?

Original:

```
root@debian:~# docker rm --help

Usage:  docker rm [OPTIONS] CONTAINER [CONTAINER...]

Remove one or more containers

Options:
  -f, --force     Force the removal of a running container (uses SIGKILL)
  -l, --link      Remove the specified link
  -v, --volumes   Remove the volumes associated with the container
```

Traduzido:

```
root @ debian: ~ # docker rm --help

Uso: docker rm [OPÇÕES] CONTAINER [CONTAINER ...]

Remova um ou mais recipientes

Opções:
  -f, --force Força a remoção de um contêiner em execução (usa SIGKILL)
  -l, --link Remove o link especificado
  -v, --volumes Remove os volumes associados ao contêiner
```

Bem, pelo que podemos notar, podemos manipular certas propriedades com o **RM**, forçar o delete de containers mesmo em execução, demonstrar volumes sendo utilizados por esses containers ou mesmo remover o link entre containers.

Vamos ver um `-f` em ação para se ter uma idéia da utilidade da ferramenta:

```
root@debian:~# docker run -ti ubuntu
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
ecd255b49ca8        ubuntu              "/bin/bash"         5 seconds ago       Up
3 seconds                           focused_jones
bb72e2b64cd5        ubuntu              "/bin/bash"         7 minutes ago       Up
7 minutes                           tender_hofstadter
root@debian:~# docker rm -f ecd255b49ca8
ecd255b49ca8
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
bb72e2b64cd5        ubuntu              "/bin/bash"         7 minutes ago       Up
7 minutes                           tender_hofstadter
```

Mesmo tendo o container em execução o mesmo foi deletado.

Agora, tambem existem outros métodos de delePorém existem outros comandos para dar suporte a essa parte, como exemplo o `prune`:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
bb72e2b64cd5        ubuntu              "/bin/bash"         About a minute ago
Up About a minute                               tender_hofstadter
88ea6ecb1446        ubuntu              "/bin/bash"         2 minutes ago
Exited (0) 2 minutes ago                        romantic_mirzakhani
532dd4809c28        ubuntu              "/bin/bash"         2 minutes ago
Exited (0) 2 minutes ago                        relaxed_roentgen
```

```
acaea6050b5e        ubuntu              "/bin/bash"         2 minutes ago
Exited (0) 2 minutes ago                       heuristic_diffie
root@debian:~#
root@debian:~# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
88ea6ecb1446c5a493d9ab834a73834ffa93071ccfa2c5a70b77eae9941e73da
532dd4809c280114b70940458974e6912253ce152741b2cad13d8d24e385d46f
acaea6050b5edc7ce66aca35dc270a2479081b47e47eed0ef9247a69694cd2ed

Total reclaimed space: 0B
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
bb72e2b64cd5        ubuntu              "/bin/bash"         2 minutes ago       Up
2 minutes                           tender_hofstadter
```

O **PRUNE** é um comando derivado do `docker container`, um outro conjunto de ferramentas para manusear os containers, este será dito mais a frente.

Pequena dica para a vida

Digamos que você tem este cenário no eu `ps -a`:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         About an hour ago   Up
About an hour                                       objective_easley
e3299d35d1c6        4e2eef94cd6b        "/bin/bash"         About an hour ago
Exited (0) About an hour ago                        condescending_hodgkin
6e1a9c33eed1        ubuntu:xenial       "/bin/bash"         About an hour ago
Exited (0) About an hour ago                        awesome_panini
```

Más você quer limpar tudo isso e deixar sem nenhum container criado, parado ou executando, para isso podemos usar uma combinação de comandos docker e SHELL para resolver, como esse:

```
docker rm -f $(docker ps -aq)
```

Agora se for executado o `ps -a` novamente, qual será o resultado?

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         About an hour ago   Up
About an hour                       objective_easley
```

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
19490514cfd1        4e2eef94cd6b        "/bin/bash"         About an hour ago   Up
About an hour                                   objective_easley
e3299d35d1c6        4e2eef94cd6b        "/bin/bash"         About an hour ago
Exited (0) About an hour ago                        condescending_hodgkin
6e1a9c33eed1        ubuntu:xenial       "/bin/bash"         About an hour ago
Exited (0) About an hour ago                        awesome_panini
root@debian:~# docker rm -f $(docker ps -aq)
19490514cfd1
e3299d35d1c6
6e1a9c33eed1
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Ola só o resultado devastador desse comando, ele simplemente passou por cima de tudo e deletou, até mesmo o container em execução, más há um porém, só foram deletados os containers montados, nenhum das imagens forá afetada, vamos ver um pouco sobre elas agora, há e mais para frente, tem um comando que faz o mesmo sobre as imagens.

Imagens dos containers

Muito importante, más até agora não comentada a fundo, bem, aqui estamos, uma imagem como dito é um conjunto de **PULL'S**, más o que era mesmo isso? veja:

```
root@debian:~# docker run ubuntu:xenial
Unable to find image 'ubuntu:xenial' locally
xenial: Pulling from library/ubuntu
8e097b52bfb8: Pull complete
a613a9b4553c: Pull complete
acc000f01536: Pull complete
73eef93b7466: Pull complete
Digest: sha256:3dd44f7ca10f07f86add9d0dc611998a1641f501833692a2651c96defe8db940
Status: Downloaded newer image for ubuntu:xenial
```

PULL's são as camadas para a formação de uma imagem que pode ser usada para subir containers, más onde estão essas imagens? Bem, se estiver no Linux ou no Mac, provavelmente será neste caminho padrão:

```
/var/lib/docker/
```

Agora no Windows, o mesmo fica dentro do diretório de programas, nessa possível pasta padrão:

```
C:\program data\docker
```

OBS: A letra da unidade muda conforme o a instalação realizada.

Agora sobre as imagens

Execute o comando abaixo:

```
docker images
```

Ele provavelmente irá mostrar uma tela assim:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              xenial              4b22027ede29        2 weeks ago
127MB
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
```

Más o que é cada um desses campos? Veja:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
Repositório é a imagem sendo utilizada	Versão da imagem	ID da imagem dentro do HOST, pode ser usada para dar um **RUN**	Quando a imagem foi criada, no caso quando autor de **BUILD** na imagem baixada	Espaço em disco da imagem

Se der um **HELP** aqui, olha o que sai:

Original:

```
root@debian:~# docker images --help

Usage:  docker images [OPTIONS] [REPOSITORY[:TAG]]

List images

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
```

```
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
```

Traduzido:

```
root @ debian: ~ # docker images --help

Uso: imagens do docker [OPÇÕES] [REPOSITÓRIO [: TAG]]

Listar imagens

Opções:
  -a, --all Mostra todas as imagens (o padrão oculta as imagens intermediárias)
      --digests Mostra resumos
  -f, --filter filter Filtra a saída com base nas condições fornecidas
      --format string Imprima imagens bonitas usando um modelo Go
      --no-trunc Não truncar a saída
  -q, --quiet Mostra apenas IDs numéricos
```

Posso dizer que só o `docker images` é necessário para uso minimo, porém é interessante conhecer as demais opções, como o exemplo abaixo:

```
root@debian:~# docker images -aq
4b22027ede29
4e2eef94cd6b
bf756fb1ae65
```

Esse comando imprimiu todos os **IMAGES ID** que você possui no **HOST**, com eles é possível deletar todas imagens por exemplo.

Más para deletar uma imagem no **HOST** você deve ficar ciente que, como o processo funciona em PULL's, imagens compartilham os mesmo, assim se você deletar uma imagem que compartilha suas camadas com outra, a imagem que bebe de outra ficará quebrada.

Um exemplo de como remover uma imagem:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              xenial              4b22027ede29        2 weeks ago
127MB
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
root@debian:~# docker rmi 4e2eef94cd6b
Untagged: ubuntu:latest
```

```
Untagged:
ubuntu@sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Deleted: sha256:4e2eef94cd6b93dd4d794c18b45c763f72edc22858e0da5b6e63a4566a54c03c
Deleted: sha256:160004bdd9a2800d0085be0315b769a9ce04c07ca175ecae89593eeee9aeb944
Deleted: sha256:9ed638911072c3379e75d2eaf7c2502220d6757446325c8d96236410b0729268
Deleted: sha256:ce7da152e578608030e9a05f9f5259b329fe5dcc5bf48b9f544e48bd69a5f630
Deleted: sha256:2ce3c188c38d7ad46d2df5e6af7e7aed846bc3321bdd89706d5262fefd6a3390
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              xenial              4b22027ede29        2 weeks ago
127MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
```

Veja que ele deleta os PULL's e como podemos ver, nenhuma imagem fazia referência a aquela que foi apagada.

Agora vamos testar deletar uma imagem que faz referência a outras imagens:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
teste               latest              b1d74098cb41        About an hour ago
944MB
node                latest              40ce906a3734        2 days ago
944MB
root@debian:~# docker rmi 40ce906a3734
Error response from daemon: conflict: unable to delete 40ce906a3734 (cannot be
forced) - image has dependent child images
root@debian:~# docker rmi -f 40ce906a3734
Error response from daemon: conflict: unable to delete 40ce906a3734 (cannot be
forced) - image has dependent child images
```

Notemos que tivemos resistência mesmo usando o **-F** para forçar a operação, más ainda sim falho, o motivo é simples, uma imagem não faz referência, más sim foi gerada sobre a outra, assim se deletar essa imagem comprometeria sua **filha**, más e se deletar-mos a **filha** antes, veja:

```
root@debian:~# docker rmi -f b1d74098cb41
Untagged: teste:latest
Deleted: sha256:b1d74098cb419e8f34416342c09bfdb4eb85a0e8bdf15e32e9f3f6f1ee5d50d8
Deleted: sha256:185d8fc716087f55068f809bf861ebf4d86cdcc04a4670839443205e93075f2d
Deleted: sha256:c3f362a5e4fdaeba28d5ddab57e2bd2624d2cfe2d24551101b4396ce4b23514b
Deleted: sha256:14618cf7f0d7df0420cdfaf698e4d4b160b48ea74c0821e03b95c7a2594a1625
Deleted: sha256:82ff017d11e7ae49bf9554f6b430a6f64788cbe36044ac4fdf864187908f01af
Deleted: sha256:d35d966a4e067d40998342c8a1a98954b7ebb555fa74de93fb3b55308ad1dae1
Deleted: sha256:4bbdc20f37da07b92362a4e36144cf1e065f947840f001c3091ec760a1a3168f
Deleted: sha256:dcd702923b40e2771d602324ec24a419406c13684256faf5eb040820ea5694d9
root@debian:~# docker images
```

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
node                latest              40ce906a3734        2 days ago
944MB
root@debian:~# docker rmi 40ce906a3734
Untagged: node:latest
Untagged:
node@sha256:ce506ed8986a0c8a364757771679706ebd129fa466165fcc6e2c7dc449a0baac
Deleted: sha256:40ce906a37347c6f7af9c2a031bc8f3846707084cfe34d48cf1a671ff6e35bc7
Deleted: sha256:eb98c6b0b7f568e55559505c79ad70dfa1e69d7fd40bd8da64fab66e7e40d400
Deleted: sha256:cbb6354e205e7fb063e08e9ef882c57588bfd04f79b6169e5d2c617db81396f9
Deleted: sha256:78c5a8e2c9e3a97197b49c8da877e4f4212fb720e401ba6fa2a989106ed03d16
Deleted: sha256:636859e63d6275982c1e4f16d930e3df48f7cc6fd85acede66131c144813a835
Deleted: sha256:49250fa5f097bc3019d60510dc4f4e11503ad00145e68a40f76b8e42b11c48e5
Deleted: sha256:12b655ddd5543bba6d0bb263edad5c4440aa05e7bf207a17031984eb55d48fda
Deleted: sha256:ec2cbfca075a717928b072100ca1d70b7a95fe5b8ecdf9418824a44a839e255e
Deleted: sha256:438031af5d1e5a47e5edef5a986f6a6549f9f05dc067b9e39f79b05c7c10370e
Deleted: sha256:4e38024e7e09292105545a625272c47b49dbd1db721040f2e85662e0a41ad587
```

Bem... deu para entender o conceito básico de se utilizar um mesmo **PULL** para várias imagens ao mesmo tempo, más e se tentarmos deletar uma imagem que está sendo usada em um container? Olha o resultado:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3f877c157a15        ubuntu:xenial       "/bin/bash"         5 seconds ago       Up
4 seconds                               youthful_lamarr
7a8a4d60e74b        ubuntu:xenial       "/bin/bash"         23 seconds ago      Up
22 seconds                              laughing_cray
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              xenial              4b22027ede29        2 weeks ago
127MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
root@debian:~# docker rmi 4b22027ede29
Error response from daemon: conflict: unable to delete 4b22027ede29 (cannot be
forced) - image is being used by running container 7a8a4d60e74b
```

Ele acusa um erro que a imagem está sendo usada e até aponta quem a está usando e até mesmo aponta que não adianta dar **FORCE** ou `-f` para deletar a imagem, neste caso primeiramente, se deve matar o container para então fazer a operação, veja:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3f877c157a15        ubuntu:xenial       "/bin/bash"         4 minutes ago       Up
4 minutes                               youthful_lamarr
```

```
7a8a4d60e74b        ubuntu:xenial       "/bin/bash"         4 minutes ago       Up
4 minutes                               laughing_cray
root@debian:~# docker stop 3f877c157a15
3f877c157a15
root@debian:~# docker stop 7a8a4d60e74b
7a8a4d60e74b
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
3f877c157a15        ubuntu:xenial       "/bin/bash"         4 minutes ago
Exited (0) 8 seconds ago                        youthful_lamarr
7a8a4d60e74b        ubuntu:xenial       "/bin/bash"         4 minutes ago
Exited (0) 2 seconds ago                        laughing_cray
dfd386e35947        ubuntu:xenial       "/bin/bash"         11 hours ago
Exited (0) 11 hours ago                         brave_mahavira
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              xenial              4b22027ede29        2 weeks ago
127MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
root@debian:~# docker rmi -f 4b22027ede29
Untagged: ubuntu:xenial
Untagged:
ubuntu@sha256:3dd44f7ca10f07f86add9d0dc611998a1641f501833692a2651c96defe8db940
Deleted: sha256:4b22027ede299ea02d9d6236db8767e87b67392cf81535c18f7c202294a4a208
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                       PORTS               NAMES
3f877c157a15        4b22027ede29        "/bin/bash"         4 minutes ago
Exited (0) 34 seconds ago                       youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         5 minutes ago
Exited (0) 29 seconds ago                       laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         11 hours ago
Exited (0) 11 hours ago                         brave_mahavira
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
```

Olhe, foram deixados imagens orfãs no local onde estava o nome da imagem referencia dentro do **HOST**, porém esses containers ainda estão funcionando normalmente por já estarem criados, quer ver:

```
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
```

```
STATUS                      PORTS               NAMES
3f877c157a15        4b22027ede29        "/bin/bash"         6 minutes ago
Exited (0) 2 minutes ago                        youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         7 minutes ago
Exited (0) 2 minutes ago                        laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         11 hours ago
Exited (0) 11 hours ago                         brave_mahavira
root@debian:~# docker start 3f877c157a15
3f877c157a15
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
3f877c157a15        4b22027ede29        "/bin/bash"         7 minutes ago       Up
3 seconds                                   youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         7 minutes ago
Exited (0) 2 minutes ago                        laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         11 hours ago
Exited (0) 11 hours ago                         brave_mahavira
root@debian:~# docker start -i 3f877c157a15
root@3f877c157a15:/# exit
```

Más e agora? E se eu quiser subir um novo container que nem os que eu já tenho, vou ter que fazer download da imagem denovo? Nops, veja isso:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
39edf2f15b71        ubuntu              "/bin/bash"         2 minutes ago
Exited (0) 2 minutes ago                        unruffled_spence
8c96e7e8a7ce        ubuntu              "/bin/bash"         9 minutes ago
Exited (0) 3 minutes ago                        lucid_elgamal
3f877c157a15        4b22027ede29        "/bin/bash"         28 minutes ago
Exited (0) 20 minutes ago                       youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         28 minutes ago
Exited (0) 24 minutes ago                       laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         11 hours ago
Exited (0) 11 hours ago                         brave_mahavira
root@debian:~# docker rmi -f 4e2eef94cd6b
Untagged: ubuntu:latest
Untagged:
ubuntu@sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Deleted: sha256:4e2eef94cd6b93dd4d794c18b45c763f72edc22858e0da5b6e63a4566a54c03c
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
```

```
STATUS                      PORTS               NAMES
39edf2f15b71        4e2eef94cd6b        "/bin/bash"         2 minutes ago
Exited (0) 2 minutes ago                        unruffled_spence
8c96e7e8a7ce        4e2eef94cd6b        "/bin/bash"         10 minutes ago
Exited (0) 3 minutes ago                        lucid_elgamal
3f877c157a15        4b22027ede29        "/bin/bash"         28 minutes ago
Exited (0) 21 minutes ago                       youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         29 minutes ago
Exited (0) 24 minutes ago                       laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         11 hours ago
Exited (0) 11 hours ago                         brave_mahavira
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
root@debian:~# docker run ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
54ee1f796a1e: Already exists
f7bfea53ad12: Already exists
46d371e02073: Already exists
b66c17bbf772: Already exists
Digest: sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Status: Downloaded newer image for ubuntu:latest
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
1918dcca512f        ubuntu              "/bin/bash"         11 seconds ago
Exited (0) 10 seconds ago                       fervent_jackson
39edf2f15b71        ubuntu              "/bin/bash"         3 minutes ago
Exited (0) 3 minutes ago                        unruffled_spence
8c96e7e8a7ce        ubuntu              "/bin/bash"         10 minutes ago
Exited (0) 4 minutes ago                        lucid_elgamal
3f877c157a15        4b22027ede29        "/bin/bash"         29 minutes ago
Exited (0) 21 minutes ago                       youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         29 minutes ago
Exited (0) 24 minutes ago                       laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         11 hours ago
Exited (0) 11 hours ago                         brave_mahavira
```

O que aconteceu aqui? Vou explicar em pontos:

- Eu tinha uma imagem ubuntu sendo utilizada em um container;
- Após deletar essa imagem, o container perdeu a refêrencia da mesma;

- Quando foi ordenada a criação de um novo container **ubuntu**, o docker falou que não encontrou a imagem, porém afirmou que os PULL's já existiam, no caso, esses PULL's vieram dos containers parados que são camadas de leitura;
- Após afirmar o download da imagem, as referências perdidas foram colocadas novamente;

Limpar todas as imagens

Lembra aqueles comandos que limpam todos os containers? Tem um maroto também para as imagens:

```
docker rmi -f $(docker images -q)
```

Olha o resultado de quando ordenado:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
hello-world         latest              bf756fb1ae65        8 months ago
13.3kB
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
1918dcca512f        ubuntu              "/bin/bash"         13 minutes ago
Exited (0) 13 minutes ago                       fervent_jackson
39edf2f15b71        ubuntu              "/bin/bash"         16 minutes ago
Exited (0) 16 minutes ago                       unruffled_spence
8c96e7e8a7ce        ubuntu              "/bin/bash"         23 minutes ago
Exited (0) 17 minutes ago                       lucid_elgamal
3f877c157a15        4b22027ede29        "/bin/bash"         42 minutes ago
Exited (0) 34 minutes ago                       youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         42 minutes ago
Exited (0) 38 minutes ago                       laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         12 hours ago
Exited (0) 12 hours ago                         brave_mahavira
root@debian:~# docker rmi -f $(docker images -q)
Untagged: ubuntu:latest
Untagged:
ubuntu@sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Deleted: sha256:4e2eef94cd6b93dd4d794c18b45c763f72edc22858e0da5b6e63a4566a54c03c
Untagged: hello-world:latest
Untagged: hello-
world@sha256:7f0a9f93b4aa3022c3a4c147a449bf11e0941a1fd0bf4a8e6c9408b2600777c5
Deleted: sha256:bf756fb1ae65adf866bd8c456593cd24beb6a0a061dedf42b26a993176745f6b
Deleted: sha256:9c27e219663c25e0f28493790cc0b88bc973ba3b1686355f221c38a36978ac63
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
```

```
STATUS                      PORTS               NAMES
1918dcca512f        4e2eef94cd6b        "/bin/bash"         13 minutes ago
Exited (0) 13 minutes ago                       fervent_jackson
39edf2f15b71        4e2eef94cd6b        "/bin/bash"         16 minutes ago
Exited (0) 16 minutes ago                       unruffled_spence
8c96e7e8a7ce        4e2eef94cd6b        "/bin/bash"         24 minutes ago
Exited (0) 17 minutes ago                       lucid_elgamal
3f877c157a15        4b22027ede29        "/bin/bash"         42 minutes ago
Exited (0) 35 minutes ago                       youthful_lamarr
7a8a4d60e74b        4b22027ede29        "/bin/bash"         43 minutes ago
Exited (0) 38 minutes ago                       laughing_cray
dfd386e35947        4b22027ede29        "/bin/bash"         12 hours ago
Exited (0) 12 hours ago                         brave_mahavira
```

Um pequeno a mais

Dá para empilhar mais de uma imagem para se detelar de uma vez, exemplo:

```
root@debian:~# docker rmi 84f3208e6716 ac4ac064b064
Deleted: sha256:84f3208e6716b3ddf58b1d7d77deb8b8fb3072544f7d888aff07cf15cce3f0f6
Deleted: sha256:ac4ac064b064c78b7d65168e93b3ce3b07b2fd8d85c9a7e11d2eb96b9f92bca0
```

Por fim do delete

- **RM** é para deletar um container;
- **RMI** é para deletar uma imagem;

Indo para um pouco mais avançado

Tudo acima foi o introdutório do Docker, tudo para ajudar a ter um controle sobre o mesmo para a realização das tarefas, a partir desse ponto, se inicia a manipulação mais avançada.

Rede docker

O Docker cria suas proprias redes para trabalhar, sendo as padrões:

```
root@debian:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8e4a0db20c13        bridge              bridge              local
9a7d4ca72fbe        host                host                local
0c60fb4b7af7        none                null                local
```

Por padrão todo o container que não possui uma rede especificada durante sua criação, se usa do modelo **BRIDGE** para seu funcionamento, más vamos detalhar mais um pouco:

- **bridge:** Rede padrão caso não especifique uma, possui acesso externo normal e precisa que os containers redirecione as portas de saída caso necessário, são mais recomendas em aplicações Docker Host, isso é, todo o Docker em unica máquina.
- **host:** Modelo usado para quando quiser retirar as restrições da **BRIDGE**, é recomendado para quando quiser que o determinado container tome conta das portas necessárias do **HOST**, o tornando exposto a rede;
- **overlay:** Cria uma rede para comunicação entre diferentes Docker Host's, aqui tem muito o conceito de cluster, sua maior vantagem é não precisar alterar as regras de redirecionamento de rede do Docker HOST que abriga as máquinas
- **macvlan:** Faz com o Docker saia para rede com um endereço MAC valido, simulando que ele é um equipamento físico em rede, este normalmente é aplicado em casos mais especificos;
- **none:** Somente desabilida a saída de rede do container;

Redes de containers

Conectando multiplos containers

Exemplo*:

- Server web;
- aplicacao;
- Banco;
- Server de cache;

Todos interligados para criar o funcionamento, isso é cada container tem uma responsabilidade;

O docker já tem em mente o uso de redes, ele uma default network para o funcionamento dos containers e todos o containers são iniciados com ela, do 172.168.0.1 ao 254 uma rede / 24 interna do docker

Exemplo da rede de um container:

```
"NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"2a1285ce33d47a100353d07b0f4c4121f5ed969b09f237aec51ca277b3d8be2c",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "443/tcp": null,
                "80/tcp": null
            },
            "SandboxKey": "/var/run/docker/netns/2a1285ce33d4",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"dedef995f2c691296c2901c6292be1c03b7a1e9180ad39203439c05aee197e0d",
```

```
            "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":
"cddc52925969a071445ffd4739bf7ee7a2d88b6d1b0a16b59efca3e7067352c7",
                    "EndpointID":
"dedef995f2c691296c2901c6292be1c03b7a1e9180ad39203439c05aee197e0d",
                    "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
                }
            }
        }
```

Lembrando por que alguns comandos não funcionam dentro do container? O container so tem o essencial para funcionar, comandos desnecessarios nem sao colocados no container, no caso você precisa baixar os containers e lembrando que o container é temporario, caso entre em um container e faça alterações como instalar um pacote, isso será perdido no momento que o container ser morto

Provando a rede

Suba dois containers com ubuntu e instale os pacotes necessarios para o ping e ifconfig, o comando: "**apt update && apt install iputils-ping -y && apt install net-tools**", após subir os dois e instalar os comandos acima em um e pingue o outro

Exemplo:

```
root@c84a471a89dc:/# 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=0.224 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.131 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.111 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.116 ms
64 bytes from 172.17.0.3: icmp_seq=5 ttl=64 time=0.113 ms
64 bytes from 172.17.0.3: icmp_seq=6 ttl=64 time=0.114 ms
```

Para configurar a rede, isso é feito no docker hosts, o containers não precisa se conectar na rede default do sistema, eles se comunicam via docker hosts como uma camada de abstração

Criando rede no docker

O drive é como criar uma nuvem particular entre os containers, o bridge é o mais comum e resolve maiorias dos problemas, rede-containers é o nome da rede

Exemplo:

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

Mostrar as maquinas em rede do docker:

```
docker network ls
```

Associar uma rede ao container:

```
docker run ubuntu --name nome-container --network redes-containers
```

- **--network** -> fala para o container ingressar em determinada rede;

Quando criamos a rede e colocamos um nome no container, podemos pingar ele pelo nome por que define este nome como um host da rede

Criada a rede

Criada duas maquina e colocadas hostnames diferentes(obvio) e colocados na mesma rede, uma delas teve os pacotes de rede instalados e foi testado os comandos.

Teste do host teste para o teste1:

```
root@811961491abf:/# ping teste1
PING teste1 (172.18.0.3) 56(84) bytes of data.
64 bytes from teste1.rede-containers (172.18.0.3): icmp_seq=1 ttl=64 time=0.222 ms
64 bytes from teste1.rede-containers (172.18.0.3): icmp_seq=2 ttl=64 time=0.127 ms
64 bytes from teste1.rede-containers (172.18.0.3): icmp_seq=3 ttl=64 time=0.125 ms
```

Resumindo, você não mexe na rede interna do docker e sim no docker host onde você cria uma rede e atribui onde as máquina seram iniciadas com determinada rede, por padrão os containers sobem com uma default mas da para setar a rede que você quer subir usando --network, lembrando que tem que atribuir um nome para os containers para facilitar o gerenciamento

Aplicação

Executando uma aplicação que utiliza 2 containers:

```
docker pull douglasq/alura-books:cap05
docker pull mongo
```

Subir os dockers:

```
docker network create --driver bridge minha-rede
docker run -d --network minha-rede --name meu-mongo mongo
docker run -d -p 8080:3000 --network minha-rede --name meu-node node
```

Quando você abrir o localhost:8080 ele vai mostrar nada, tem que entrar numa pagina para fazer os livros subirem na aplicação

```
localhost:8080/seed
```

Após isso volte para o localhost:8080 que o livros seram carregados

Porém isso prova que ambos os containers estão conversando entre si.

Analise da rede, Mostra o que tem dentro duma rede além de outros atributo

```
docker network inspect <nome da rede>
```

Voltando ao RUN

Bem como já vimos o **RUN** é o comando para criar/iniciar uma imagem em um container, já também vimos o **HELP** desse cara anteriormente, então vamos usar alguns pararmetros que com certeza serão interessantes.

- **-P**

Esse comando é de **PORT** ou porta, como mencionado o Docker cria um container em rede **BRIDGE**, para esse container funcionar fora dessa rede ele precisa ter portas redirecionadas, Exemplo:

```
docker run -P <porta do host>:<porta do container> <imagem>
```

Agora vamos ver na prática:

```
root@debian:~# docker run -p 80:80 -ti ubuntu
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                NAMES
70539daf01fb        ubuntu              "/bin/bash"         7 seconds ago       Up
5 seconds        0.0.0.0:80->80/tcp   serene_newton
```

Podemos mostrar as portas que um container está usando com o comando `port`:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                NAMES
70539daf01fb        ubuntu              "/bin/bash"         3 minutes ago       Up
3 minutes        0.0.0.0:80->80/tcp   serene_newton
root@debian:~# docker port 70539daf01fb
80/tcp -> 0.0.0.0:80
```

Agora note também que o campo **PORT** está preenchido quando executado um `docker ps` ou um `ps -a`.

- **-V**

Montar volumes é a opção que monta um diretório HOST dentro do container, exemplo:

```
root@debian:~# docker run -v /root/teste:/mnt/ -ti ubuntu /bin/bash
root@260f104f5a69:/# ls /mnt/
file.txt
```

Um exemplo do quão utíl é esse comando é o fato que um container sempre retorna a seu estágio inicial quando é morto, assim se você montar um diretório entre os dois, os arquivos gerados no Container podem ser carregados para o Host.

- **-W**

Permite que você inicie um container em determinado diretório, exemplo:

- Sem o **-W**:

```
root@debian:~# docker run -ti ubuntu
root@94fd8a1950ba:/# pwd
/
```

- Com o **-W**:

```
root@debian:~# docker run -w /mnt -ti ubuntu
root@7e371ca782a1:/mnt# pwd
/mnt
```

- **--name**

Como dito bem no ínicio desse manual, o Docker gera nomes aleatórios para seus container, exemplo:

```
root@debian:~# docker run ubuntu
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
b1547232cdf0        ubuntu              "/bin/bash"         4 seconds ago
Exited (0) 2 seconds ago                        loving_kepler
```

Porém podemos manipular esses nomes para falacilitar nosso entendimento sobre qual container está funcionando:

```
root@debian:~# docker run --name teste ubuntu
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
76471d84ebeb        ubuntu              "/bin/bash"         4 seconds ago
Exited (0) 2 seconds ago                            teste
b1547232cdf0        ubuntu              "/bin/bash"         About a minute ago
Exited (0) About a minute ago                       loving_kepler
```

- **rename**

Podemos renomear container's já criados dessa forma:

```
docker rename <nome atual do container> <novo nome do container>
```

Ou como nesse exemplo prático:

```
root@debian:~# docker ps  -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
76471d84ebeb        ubuntu              "/bin/bash"         2 minutes ago
Exited (0) 2 minutes ago                        teste
b1547232cdf0        ubuntu              "/bin/bash"         4 minutes ago
Exited (0) 4 minutes ago                        loving_kepler
root@debian:~# docker rename loving_kepler pastel
root@debian:~# docker ps
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
root@debian:~# docker ps  -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
76471d84ebeb        ubuntu              "/bin/bash"         3 minutes ago
Exited (0) 3 minutes ago                        teste
b1547232cdf0        ubuntu              "/bin/bash"         4 minutes ago
Exited (0) 4 minutes ago                        pastel
```

Inspecionar

Inspecionar serve para verificar as informação de todo o ativo que se utiliza de **ID** dentro do Docker:

```
docker inspect <id>
```

De forma prática, olhe a informação retirada de um container de ID **9fd1352d1ad1**:

```
root@debian:~# docker inspect 9fd1352d1ad1
[
    {
        "Id": "9fd1352d1ad12848c5f71261811378fd67a35decabd5a8a6c1ea9f16342324b3",
        "Created": "2020-09-03T17:47:09.13238247Z",
        "Path": "/bin/bash",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1236,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2020-09-03T17:47:10.005715222Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:4609ccaa260d49f6f55339883c64c342a5948be70e042b1ebf4d52a19d6b77c7",
        "ResolvConfPath":
"/var/lib/docker/containers/9fd1352d1ad12848c5f71261811378fd67a35decabd5a8a6c1ea9f
16342324b3/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/9fd1352d1ad12848c5f71261811378fd67a35decabd5a8a6c1ea9f
16342324b3/hostname",
        "HostsPath":
"/var/lib/docker/containers/9fd1352d1ad12848c5f71261811378fd67a35decabd5a8a6c1ea9f
```

```
16342324b3/hosts",
        "LogPath":
"/var/lib/docker/containers/9fd1352d1ad12848c5f71261811378fd67a35decabd5a8a6c1ea9f
16342324b3/9fd1352d1ad12848c5f71261811378fd67a35decabd5a8a6c1ea9f16342324b3-
json.log",
        "Name": "/serene_neumann",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "shareable",
            "Cgroup": "",
            "Links": null,
            "OomScoreAdj": 0,
            "PidMode": "",
            "Privileged": false,
            "PublishAllPorts": false,
            "ReadonlyRootfs": false,
            "SecurityOpt": null,
            "UTSMode": "",
            "UsernsMode": "",
            "ShmSize": 67108864,
            "Runtime": "runc",
            "ConsoleSize": [
                0,
                0
            ],
            "Isolation": "",
```

```
            "CpuShares": 0,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": [],
            "BlkioDeviceReadBps": null,
            "BlkioDeviceWriteBps": null,
            "BlkioDeviceReadIOps": null,
            "BlkioDeviceWriteIOps": null,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DeviceCgroupRules": null,
            "DiskQuota": 0,
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": false,
            "PidsLimit": 0,
            "Ulimits": null,
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0,
            "MaskedPaths": [
                "/proc/asound",
                "/proc/acpi",
                "/proc/kcore",
                "/proc/keys",
                "/proc/latency_stats",
                "/proc/timer_list",
                "/proc/timer_stats",
                "/proc/sched_debug",
                "/proc/scsi",
                "/sys/firmware"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/7b541c2b0c359d2d7ae4d20b1136a542ae669739e95c9f1b19395877
```

```
6f53cd4d-
init/diff:/var/lib/docker/overlay2/965a2333b8b9d7b31a429f54499151e27a384ee581a5bb4
b5295991807acc141/diff:/var/lib/docker/overlay2/85be6a094133c6d592bfb366131297135e
88879c5d4a4e60bf6f89c964ae340d/diff:/var/lib/docker/overlay2/af77aff143a6f43106d57
6d10a1c033f75c717c5c860d328f33253ce7dabada3/diff:/var/lib/docker/overlay2/b41b6325
44d37f09a5f0e38d26e3c430628b7bd3f3efa377edd30277dcb324f1/diff",
                "MergedDir":
"/var/lib/docker/overlay2/7b541c2b0c359d2d7ae4d20b1136a542ae669739e95c9f1b19395877
6f53cd4d/merged",
                "UpperDir":
"/var/lib/docker/overlay2/7b541c2b0c359d2d7ae4d20b1136a542ae669739e95c9f1b19395877
6f53cd4d/diff",
                "WorkDir":
"/var/lib/docker/overlay2/7b541c2b0c359d2d7ae4d20b1136a542ae669739e95c9f1b19395877
6f53cd4d/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "9fd1352d1ad1",
            "Domainname": "",
            "User": "",
            "AttachStdin": true,
            "AttachStdout": true,
            "AttachStderr": true,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": true,
            "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/bash"
            ],
            "ArgsEscaped": true,
            "Image": "teste_commit:0.0.1",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"ebec81bb8d16e4a5ef5cce40962ce764cfa934f4bc49a9bf10963b276e660d62",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {},
            "SandboxKey": "/var/run/docker/netns/ebec81bb8d16",
            "SecondaryIPAddresses": null,
```

```
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"92edafd4a4356335208147d6ad616c46ddb067c37aa42ce8a5e8329593dd0983",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.3",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:03",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"cb7167af2a895daea1df622f0601bd0e9acb0d23df45e6bc8813d25ece784c28",
                    "EndpointID":
"92edafd4a4356335208147d6ad616c46ddb067c37aa42ce8a5e8329593dd0983",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.3",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:03",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

Podemos fazer isso com um container, olhe:

```
root@debian:~# docker inspect 4e2eef94cd6b
[
    {
        "Id":
"sha256:4e2eef94cd6b93dd4d794c18b45c763f72edc22858e0da5b6e63a4566a54c03c",
        "RepoTags": [
            "ubuntu:latest"
        ],
        "RepoDigests": [

"ubuntu@sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-08-19T21:14:39.216060179Z",
        "Container":
"83b09651dad2f320e2334e67eed9a69bda9de334539586d7b6ccdd124608ed26",
```

```
        "ContainerConfig": {
            "Hostname": "83b09651dad2",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"/bin/bash\"]"
            ],
            "ArgsEscaped": true,
            "Image":
"sha256:d6008edcd217c7746a0804212db824c2a4431f86fe7c019b6d6119a2bfae0bba",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "DockerVersion": "18.09.7",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/bash"
            ],
            "ArgsEscaped": true,
            "Image":
"sha256:d6008edcd217c7746a0804212db824c2a4431f86fe7c019b6d6119a2bfae0bba",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
```

```
            "OnBuild": null,
            "Labels": null
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 73861198,
        "VirtualSize": 73861198,
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/85be6a094133c6d592bfb366131297135e88879c5d4a4e60bf6f89c9
64ae340d/diff:/var/lib/docker/overlay2/af77aff143a6f43106d576d10a1c033f75c717c5c86
0d328f33253ce7dabada3/diff:/var/lib/docker/overlay2/b41b632544d37f09a5f0e38d26e3c4
30628b7bd3f3efa377edd30277dcb324f1/diff",
                "MergedDir":
"/var/lib/docker/overlay2/965a2333b8b9d7b31a429f54499151e27a384ee581a5bb4b52959918
07acc141/merged",
                "UpperDir":
"/var/lib/docker/overlay2/965a2333b8b9d7b31a429f54499151e27a384ee581a5bb4b52959918
07acc141/diff",
                "WorkDir":
"/var/lib/docker/overlay2/965a2333b8b9d7b31a429f54499151e27a384ee581a5bb4b52959918
07acc141/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [

"sha256:2ce3c188c38d7ad46d2df5e6af7e7aed846bc3321bdd89706d5262fefd6a3390",

"sha256:ad44aa179b334bbf4aeb61ecef978c3c77a3bb27cb28bcb727f5566d7f085b31",

"sha256:35a91a75d24be7ff9c68ce618dcc933f89fef502a59becac8510dbc3bf7a4a05",

"sha256:a4399aeb9a0e1ddf9da712ef222fd66f707a8c7205ed2607c9c8aac0dbabe882"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        }
    }
]
```


De todos os listados, é possível se alterar alguns valores, pois os listados são paramentros para o funcionamento da imagem ou container.

Configurando esses parâmetros

Bem, o inspect já nos mostrou que um container possível vários pontos que podem ser configurados, vamos ver um poucos deles, sendo mais especifico, o processador e memória.

Quando se usa um container Docker, o mesmo utilizada **TODA** a máquina como seu **POOL** de recursos,... Bem, imagino que você notou o perigo disso, imagine aquele container processando o mundo e fundo e do nada sua máquina host morre! Quem tem Windows já deve ter sofrido com um processo chamado de **VMMEM** com o Docker, então, é o mesmo, recursos instanciados do **HOST** para o Docker... No começo desses manual foi falado que a vantagem da virtualização e dos containers são ambientes isolados que não afetam o **HOST**, isso só é verdade quando se é configurado corretamente.

Bem antes de tudo é valido lembrar que:

- Funções de controle sobre containers são somente válidos as versões mais novas do Docker;
- Quantidade valida de processadores para um container é a quantidade de threads;
- **EX:**

```
1º Pessoa -> Eu tenho um I7 com 4 cores!
2º Pessoa -> Quantos threads ele tem?
1º Pessoa -> 8, más porquê?
2º Pessoa -> Então você tem 8 nucleos totais!
```

Entenderam a sacada, deve se ter em mente a quantidade total de threads disponíveis para se dedicar aos containers.

Vamos nessa então, veja:

```
root@debian:~# docker inspect c2a0bb6a7857 | grep -i mem
            "Memory": 0,
            "CpusetMems": "",
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
root@debian:~# docker inspect c2a0bb6a7857 | grep -i cpu
            "CpuShares": 0,
            "NanoCpus": 0,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "CpuCount": 0,
            "CpuPercent": 0,
```

Esse container esta sem configuração de limite de recursos, vamos criar um container mais contralado:

```
root@debian:~# docker run -it --cpus=1 -m 256MB ubuntu
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
25bb0a49ecf0        ubuntu              "/bin/bash"         26 seconds ago      Up
25 seconds                              gifted_hugle
root@debian:~# docker inspect 25bb0a49ecf0 | grep -i mem
            "Memory": 268435456,
            "CpusetMems": "",
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": -1,
            "MemorySwappiness": null,
root@debian:~# docker inspect 25bb0a49ecf0 | grep -i cpu
            "CpuShares": 0,
            "NanoCpus": 1000000000,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "CpuCount": 0,
            "CpuPercent": 0,
```

Veja no campo **NANOCPUS** e no campo **MEMORY** foram setados os valores ordenados pelo comando, assim esse é o limite de hardware que o determinado container possível, porém é possível alterar esse valores se utilizando do `docker update`, veja:

```
root@debian:~# docker container update -m 128M 25bb0a49ecf0
25bb0a49ecf0
Your kernel does not support swap limit capabilities or the cgroup is not mounted.
Memory limited without swap.
root@debian:~# docker inspect 25bb0a49ecf0 | grep -i mem
            "Memory": 134217728,
            "CpusetMems": "",
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": -1,
            "MemorySwappiness": null,
```

Bem, com isso já se é possível criar testes mais complexos sem o risco do seu **HOST** parar de funcionar.

COMMIT

Já mexeu em tanta coisa, más ainda não está sentido falta de nada não, por exemplo, como faz para aquele container super pronto virar uma imagem? Heheheh, parace que falta algo bem importante depois dessa

frase né, saca só:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
root@debian:~# docker run -ti ubuntu
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
57a06df5524c        ubuntu              "/bin/bash"         5 seconds ago       Up
4 seconds                               musing_chatelet
root@debian:~# docker commit 57a06df5524c teste_commit:0.0.1
sha256:4609ccaa260d49f6f55339883c64c342a5948be70e042b1ebf4d52a19d6b77c7
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
teste_commit        0.0.1               4609ccaa260d        4 seconds ago
73.9MB
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
57a06df5524c        ubuntu              "/bin/bash"         34 seconds ago      Up
33 seconds                              musing_chatelet
root@debian:~# docker run -ti teste_commit:0.0.1
root@9fd1352d1ad1:/# root@debian:~# docker ps
CONTAINER ID        IMAGE                COMMAND             CREATED
STATUS              PORTS               NAMES
9fd1352d1ad1        teste_commit:0.0.1   "/bin/bash"         7 seconds ago
Up 6 seconds                            serene_neumann
57a06df5524c        ubuntu               "/bin/bash"         About a minute ago
Up About a minute                       musing_chatelet
```

O commit gera uma imagem do container atual, como mostrado, o `teste_commit:0.0.1`, foi gerado sobre o `ubuntu:latest`, agora, mesmo sem aplicar um nome de versão sobre a nova imagem a mesma vai funcionar, porém sua criação ficará dessa forma:

```
root@debian:~# docker ps
CONTAINER ID        IMAGE                COMMAND             CREATED
STATUS              PORTS               NAMES
9fd1352d1ad1        teste_commit:0.0.1   "/bin/bash"         3 minutes ago
Up 3 minutes                            serene_neumann
57a06df5524c        ubuntu               "/bin/bash"         4 minutes ago
Up 4 minutes                            musing_chatelet
```

```
root@debian:~# docker commit 9fd1352d1ad1
sha256:643f468d73af0875485d2bd37b3008dd96defd887432e8ba9ae5c987ad40d06a
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
<none>              <none>              643f468d73af        3 seconds ago
73.9MB
teste_commit        0.0.1               4609ccaa260d        4 minutes ago
73.9MB
ubuntu              latest              4e2eef94cd6b        2 weeks ago
73.9MB
```

Um pouco sobre o estado dos containers.

A saúde de um container não tem tanta importãncia, já que o mesmo pode ser destruido e recontruido quantas vezes for necessário, más isso não é totalmente verdade, ficar destruindo e reconstruindo tem um custo, seja ele qual for, dessa forma, existe comandos para gerênciar a saúdes destes, como exemplos os: **TOP**, **STATS** e **LOG** da vida.

Os três mencionados são algo que qualquer um que já mexeu com Linux sabe o que são, porém o docker implementa os mesmos internamente, veja abaixo:

- **TOP**

O `docker top` demonstra os processos que o container está trabalhando no momento

Primeiro aquele **HELP** maroto:

- **Original**

```
root@debian:~# docker top --help

Usage:  docker top CONTAINER [ps OPTIONS]

Display the running processes of a container
```

- **Tradução**

```
Exibir os processos em execução de um contêiner
```

O TOP quando executado no linux, demonstra os processos em funcionamento em tempo real, exemplo (forma resumida):

```
top - 15:40:51 up  1:16,  1 user,  load average: 0,00, 0,00, 0,00
Tasks:  73 total,   2 running,  71 sleeping,   0 stopped,   0 zombie
%Cpu(s):   0,0/1,8     2[||
```

```
]
MiB Mem :    987,5 total,    568,4 free,    161,5 used,    257,6 buff/cache
MiB Swap:   1022,0 total,   1022,0 free,      0,0 used.    689,2 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  384 root      20   0  747228  34332  22652 S   0,8   3,4   0:10.54 docker-
containe
    1 root      20   0   21892   9884   7748 S   0,0   1,0   0:03.62 systemd
    2 root      20   0       0      0      0 S   0,0   0,0   0:00.01 kthreadd
    3 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/0:0H-
kblockd
    8 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S   0,0   0,0   0:00.40 ksoftirqd/0
...
```

Agora se executar no Docker, gera (P.S: Meu container está fazendo nada no momento):

```
root@debian:~# docker top 9fd1352d1ad1
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                1236                1219                0
14:47               pts/0               00:00:00            /bin/bash
```

Más olha agora um container usando o NGINX para gerar processos:

```
root@debian:~# docker top c2a0bb6a7857
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                2429                2411                0
15:53               ?                   00:00:00            nginx: master process
nginx -g daemon off;
systemd+            2487                2429                0
15:53               ?                   00:00:00            nginx: worker process
```

- **STATS**

Primeiro um **HELP**:

- **Original**

```
root@debian:~# docker stats --help

Usage:  docker stats [OPTIONS] [CONTAINER...]

Display a live stream of container(s) resource usage statistics
```

```
Options:
  -a, --all            Show all containers (default shows just running)
      --format string  Pretty-print images using a Go template
      --no-stream      Disable streaming stats and only pull the first result
      --no-trunc       Do not truncate output
```

- **Tradução**

```
Uso: estatísticas do docker [OPÇÕES] [CONTAINER...]

Exibir uma transmissão ao vivo das estatísticas de uso de recursos de contêineres

Opções:
  -a, --all Mostrar todos os recipientes (o padrão mostra apenas a execução)
      --sequência de formato Imagens de impressão bonita usando um modelo Go
      --sem fluxo Desativar estatísticas de streaming e apenas puxar o primeiro
resultado
      --no-trunc Não truncar saída
```

De forma resumida, você acompanha em tempo real o consumo do container alvo, Exemplo:

```
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
c2a0bb6a7857        wonderful_bohr      0.00%               2.133MiB / 987.5MiB
0.22%               1.01kB / 0B         0B / 8.19kB         2
```

- **LOG**

HELP antes de tudo:

- **ORIGINAL**

```
root@debian:~# docker logs --help

Usage:  docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

Options:
      --details        Show extra details provided to logs
  -f, --follow         Follow log output
      --since string   Show logs since timestamp (e.g. 2013-01-02T13:23:37) or
relative (e.g. 42m for 42 minutes)
      --tail string    Number of lines to show from the end of the logs (default
"all")
  -t, --timestamps     Show timestamps
```

```
      --until string   Show logs before a timestamp (e.g. 2013-01-02T13:23:37) or
relative (e.g. 42m for 42 minutes)
```

- **TRADUZIDO**

```
root@debian:~# registros de docker --ajuda

Uso: docker logs [OPÇÕES] CONTAINER

Buscar os troncos de um contêiner

Opções:
      -detalhes Mostre detalhes extras fornecidos aos registros
  -f, --seguir Siga a saída de log do Follow
      --desde os registros do string Show desde o timestamp (por exemplo, 2013-01-
02T13:23:37) ou relativo (por exemplo, 42m por 42 minutos)
      -sequência de cauda Número de linhas para mostrar a partir do final dos logs
(padrão "todos")
  -t, --timestamps Mostrar timestamps
      --até que string Show logs antes de um timestamp (por exemplo, 2013-01-
02T13:23:37) ou relativo (por exemplo, 42m por 42 minutos)
```

De forma resumida, é ver todos os logs que determinado container está gerando, isso salva em análises e buscas por problemas, Exemplo:

```
root@debian:~# docker logs c2a0bb6a7857
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

Dockerfile

Partiu BIR!!!

Dockerfile é um arquivo receita para a construção de constainer's customizados Docker, como exemplo:

File:

```
Dockerfile
```

Contêudo:

```
#Olha o teste
FROM node:latest
ENV PORT=3000
COPY exemplo/ /var/www
WORKDIR /var/www
RUN npm install
ENTRYPOINT npm start
EXPOSE $PORT
```

Primeiro é melhor explicar o que é cada uma dessas linhas, após isso será a explicação da ação **BUILD**:

- Pode se usar # como comentário dentro de um dockerfile

```
#Olha o teste
```

- **FROM** -> escolhe o software que a receita se base e 'lastest' é a versão mais recente;

```
FROM node:latest
```

- **ENV** ->Criando uma variavel;

```
ENV PORT=3000
```

- **COPY** -> Copiar e colar arquivo dentro do projeto;

```
COPY exemplo/ /var/www
```

- **WORKDIR** -> Definir a pasta que a receita inicia;

```
WORKDIR /var/www
```

- **RUN**-> Executar comando;

```
RUN npm install
```

- **ENTRYPOINT** -> Executando ação apos iniciar;

```
ENTRYPOINT npm start
```

- **EXPOSE**->Expor a porta do container;

```
EXPOSE $PORT
```

Explicando um pouco sobre o BUILD

Como dito, a ação **BUILD** é a construção de imagens personalizadas com base em um arquivo receita que por padrão é dito `Dockerfile`, porém o mesmo pode ser alterado a gosto do criador.

Antes de mais nada, um **HELP** para mostrar o que o **BUILD** é capaz:

Original:

```
root@debian:~# docker build --help

Usage:  docker build [OPTIONS] PATH | URL | -

Build an image from a Dockerfile

Options:
      --add-host list           Add a custom host-to-IP mapping (host:ip)
      --build-arg list          Set build-time variables
      --cache-from strings      Images to consider as cache sources
      --cgroup-parent string    Optional parent cgroup for the container
      --compress                Compress the build context using gzip
      --cpu-period int          Limit the CPU CFS (Completely Fair Scheduler)
period
      --cpu-quota int           Limit the CPU CFS (Completely Fair Scheduler)
quota
  -c, --cpu-shares int          CPU shares (relative weight)
      --cpuset-cpus string      CPUs in which to allow execution (0-3, 0,1)
      --cpuset-mems string      MEMs in which to allow execution (0-3, 0,1)
      --disable-content-trust   Skip image verification (default true)
  -f, --file string             Name of the Dockerfile (Default is
'PATH/Dockerfile')
      --force-rm                Always remove intermediate containers
      --iidfile string          Write the image ID to the file
      --isolation string        Container isolation technology
      --label list              Set metadata for an image
  -m, --memory bytes            Memory limit
      --memory-swap bytes       Swap limit equal to memory plus swap: '-1' to
enable unlimited swap
```

```
      --network string          Set the networking mode for the RUN instructions
during build (default "default")
      --no-cache                Do not use cache when building the image
      --pull                    Always attempt to pull a newer version of the
image
  -q, --quiet                   Suppress the build output and print image ID on
success
      --rm                      Remove intermediate containers after a successful
build (default true)
      --security-opt strings    Security options
      --shm-size bytes          Size of /dev/shm
  -t, --tag list                Name and optionally a tag in the 'name:tag' format
      --target string           Set the target build stage to build.
      --ulimit ulimit           Ulimit options (default [])
```

Traduzido:

```
root@debian:~# docker build --help

Uso: docker build [OPÇÕES] PATH | URL | -

Construa uma imagem a partir de um Dockerfile

Opções:
      -lista de host adicionais Adicione um mapeamento personalizado de host-to-IP
(host:ip)
      --build-arg list Definir variáveis de tempo de compilação
      -cache-de strings Imagens a considerar como fontes de cache
      --cgroup-parent cgroup Cgroup Optional parent cgroup for the container
      -compressa Compactar o contexto de compilação usando gzip
      --cpu-período int Limite do período CFS da CPU (Agendador completamente
justo)
      --cpu-quota int Limitar a cota CFS da CPU (Agendador Completamente Justo)
  -c, --cpu-ações int CPU ações (peso relativo)
      --cpuset-cpus cpus cpus em que permitir a execução (0-3, 0,1)
      --cpuset-mems string MEMs em que permitir a execução (0-3, 0,1)
      -desativar-desativar-confiança de conteúdo Pular verificação de imagem
(padrão verdadeiro)
  -f, --arquivo string Nome do Dockerfile (Padrão é 'PATH/Dockerfile')
      -force-rm Sempre remova recipientes intermediários
      --sequência de lábiosA gravar o ID de imagem para o arquivo
      --isolar a tecnologia de isolamento de contêineres
      --lista de rótulos Definir metadados para uma imagem
  -m, --memória bytes limite de memória
      --troca de memória Limite de troca igual à memória mais swap: '-1' para
ativar swap ilimitado
      Sequência de rede -- Configure o modo de rede das instruções RUN durante a
compilação (padrão "padrão")
      -sem cache Não use cache ao construir a imagem
      --puxar Sempre tente puxar uma versão mais recente da imagem
  -q, --silencioso Suprimir a saída de compilação e imprimir iD de imagem no
sucesso
```

```
      -rm Remova recipientes intermediários após uma compilação bem sucedida
(padrão verdadeiro)
      --opções de segurança optam por cadeias de segurança Opções de segurança
      --shm-size bytes Tamanho de /dev/shm
  -t, -tag list Nome e opcionalmente uma tag no formato 'nome:tag'
      --sequência de alvos, defina o estágio de construção do alvo para construir.
      --opções ulimit ulimit Ulimit Ulimit (padrão [])
```

Bem, agora vamos fazer um BUILD simples sobre o contêudo demonstrado:

```
docker build .
```

Olhe a saida disso:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
root@debian:~# docker build .
Sending build context to Docker daemon  50.69kB
Step 1/7 : FROM node:latest
latest: Pulling from library/node
419e7ae5bb1e: Pull complete
848839e0cd3b: Pull complete
de30e8b35015: Pull complete
258fdea6ea48: Pull complete
ca1b0e608d7b: Pull complete
dd8cac1f0c02: Pull complete
a9b903adc613: Pull complete
065afc31ce09: Pull complete
8a2007a51d89: Pull complete
Digest: sha256:ce506ed8986a0c8a364757771679706ebd129fa466165fcc6e2c7dc449a0baac
Status: Downloaded newer image for node:latest
 ---> 40ce906a3734
Step 2/7 : ENV PORT=3000
 ---> Running in 93d935175f46
Removing intermediate container 93d935175f46
 ---> dcd702923b40
Step 3/7 : COPY exemplo/ /var/www
 ---> d35d966a4e06
Step 4/7 : WORKDIR /var/www
 ---> Running in 27361dae7f40
Removing intermediate container 27361dae7f40
 ---> 82ff017d11e7
Step 5/7 : RUN npm install
 ---> Running in 1822ba5b3a93
npm WARN saveError ENOENT: no such file or directory, open '/var/www/package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open '/var/www/package.json'
npm WARN www No description
```

```
npm WARN www No repository field.
npm WARN www No README data
npm WARN www No license field.

up to date in 0.545s
found 0 vulnerabilities

Removing intermediate container 1822ba5b3a93
 ---> c3f362a5e4fd
Step 6/7 : ENTRYPOINT npm start
 ---> Running in b27a0433bb91
Removing intermediate container b27a0433bb91
 ---> 185d8fc71608
Step 7/7 : EXPOSE $PORT
 ---> Running in e06654d31961
Removing intermediate container e06654d31961
 ---> b1d74098cb41
Successfully built b1d74098cb41
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
<none>              <none>              b1d74098cb41        2 minutes ago
944MB
node                latest              40ce906a3734        2 days ago
944MB
root@debian:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
root@debian:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Então, o **BUILD** é faz quase a mesma coisa que o **RUN**, ele entende o arquivo receita e busca os **PULL's** necessários para fazer a imagem, além de que, caso queira alterar algo no arquivo e dar **BUILD** denovo, a imagem resultante só irá alterar as parte modificadas, Ex:

	Dockerfile - 1	**Dockerfile - 2**	**Alterações**
Base	Ubuntu	Cento	X
Aplicação	Apache	Apache	
Customizações	XYZ	XYZ	

...

Veja, se eu der um **BUILD** no Dockerfile - 1, eu terei uma imagem com Ubuntu/apache, más se eu alterar esse arquivo e colocar um Cento OS como base, no momento que der o **BUILD**, ele só vai procurar pela imagem Cento e utilizar o Apache que já tava ali.

Todos os **STEP** são passos para a conclusão da criação da imagem e seguem a quantidade de linhas existentes dentro do **Dockerfile**.

Voltando para a saida do BUILD

Você notou isso?

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
<none>              <none>              b1d74098cb41        2 minutes ago
944MB
node                latest              40ce906a3734        2 days ago
944MB
```

A imagem está `none`, esse cenário acontece quando a imagem não tem referência, para resolver isso, coloque a TAG durante a construção da imagem, veja:

```
docker build . -t teste:latest
```

Olhe a saída:

```
root@debian:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
teste               latest              b1d74098cb41        About an hour ago
944MB
node                latest              40ce906a3734        2 days ago
944MB
```

Como foi falado, a alteração afetou a imagem base e somente colocou a **TAG**.

O `.` é usado quando estamos dando build em um arquivo chamado `Dockerfile`, se tivesemos colocado qualquer outro nome o `.` não iria funcionar.

O `-t` é para nomear a imagem que está sendo construida pelo **BUILD**.

O `teste` é o **REPOSITORY** da imagem e o `latest` é a versão da imagem, notemos também que como foi comentado anteriormente, o **CREATED** está registrado que a imagem foi criada agora, enquando a base `node` que forá utilizada, foi criada a 2 dias atrás.

Docker Hub

Como explicado no inicio do trabalho, o **HUB** é mantido pela Docker Inc e ó o repositório central para as imagens, caso for um incristo na plataforma é possível enviar imagens criadas por você para uso da comunidade e como um backup.

- Entra no site, coloque ou crie sua conta

- Após isso va para o terminal e execute docker login, entre com usuario e senha, apos isso voce cai estar logado

Para subir a imagem no hub:

```
docker push <nome da imagem>
```

Exemplo:

```
docker push <nome do usuario do docker hub>/<nome da imagem>
```

Sobre as camadas dos containers o mesmo vale no push. Ele so sobe as camadas que for precisar e não sobe o que ele notar quee já existe, assim ele sobe so as camadas exenciais e as configuracao;

Para fazer o up da uma imagem

```
docker pull <nome do usuario>/<imagem>
```

Ex:

```
docker pull teste/teste_ubuntu
```

Docker compose

Até agora subimos todos os containers na mão, porém daqui em diante vamos automatizar este processo para que não seja:

- Repetitivo;
- Cansativo;
- Cheio de falhas;
- Humanamente lento e problemático;

Más não é que é ruim subir na mão os containers, é que numa aplicação dos mesmos de forma a fornecer um serviço por exemplo, o administrador tem que saber o que está acontecendo, más não trabalham a todo o instante sobre a plataforma, assim quanto mais automatizado melhor o gerenciamento sobre os containers.

Agora, vemos lembrar la do inicio **#TEXTO**, que afirma-va que um container tem a propósta de funcionar com uma unica aplicação sobre ele, assim não pense que é só colocar toda a sua produção sobre um container e esperar que de tudo certo... tudo bem, se nada de ruim acontecer tudo vai funcionar... más como você está neste manual você sabe, **MERLIN's acontencem**.

Para isso existe a ferramenta Docker Composede que é o orquestrador de containers para automação de tarefas.

Instalando o compose

Siga esse link para a distro que estiver utilizando:

- https://docs.docker.com/compose/install/

Antes de tudo um HELP

- **Original:**

```
root@debian:~/exemplo# docker-compose --help
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compose file
                              (default: docker-compose.yml)
  -p, --project-name NAME     Specify an alternate project name
                              (default: directory name)
  -c, --context NAME          Specify a context name
  --verbose                   Show more output
  --log-level LEVEL           Set log level (DEBUG, INFO, WARNING, ERROR,
CRITICAL)
  --no-ansi                   Do not print ANSI control characters
  -v, --version               Print version and exit
  -H, --host HOST             Daemon socket to connect to

  --tls                       Use TLS; implied by --tlsverify
  --tlscacert CA_PATH         Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH  Path to TLS certificate file
  --tlskey TLS_KEY_PATH       Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
  --skip-hostname-check       Don't check the daemon's hostname against the
                              name specified in the client certificate
  --project-directory PATH    Specify an alternate working directory
                              (default: the path of the Compose file)
  --compatibility             If set, Compose will attempt to convert keys
                              in v3 files to their non-Swarm equivalent
  --env-file PATH             Specify an alternate environment file

Commands:
  build              Build or rebuild services
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove containers, networks, images, and volumes
  events             Receive real time events from containers
  exec               Execute a command in a running container
```

```
  help                Get help on a command
  images              List images
  kill                Kill containers
  logs                View output from containers
  pause               Pause services
  port                Print the public port for a port binding
  ps                  List containers
  pull                Pull service images
  push                Push service images
  restart             Restart services
  rm                  Remove stopped containers
  run                 Run a one-off command
  scale               Set number of containers for a service
  start               Start services
  stop                Stop services
  top                 Display the running processes
  unpause             Unpause services
  up                  Create and start containers
  version             Show the Docker-Compose version information
```

- **Traduzido:**

```
root@debian:~/exemplo# docker-compor --ajuda
Defina e execute aplicativos de vários contêineres com o Docker.

Uso:
  docker-compor [-f <arg>...] [opções] [COMANDO] [ARGS...]
  docker-compor -h|--ajuda

Opções:
  -f, -arquivo ARQUIVO Especifique um arquivo de composição alternativo
                              (padrão: docker-compor.yml)
  -p, --nome do projeto Especifique um nome de projeto alternativo
                              (padrão: nome do diretório)
  -c, --nome do contexto Especifique um nome de contexto
  --verbose Mostrar mais saída
  Nível de registro nível de registro nível de registro nível de registro nível de
registro nível de registro (DEPURG, INFO, AVISO, ERRO, CRÍTICO)
  --no-ansi Não imprima caracteres de controle ANSI
  -v, --versão Imprimir versão e saída
  -H, -host HOST Daemon soquete para conectar-se a

  --tls Usar TLS; implícita por --tlsverify
  --tlscacert CA_PATH Certs trust assinados apenas por este CA
  --tlscert CLIENT_CERT_PATH caminho para arquivo de certificado TLS
  -tlskey TLS_KEY_PATH caminho para arquivo de teclaS TLS
  --tlsverificar o Uso de TLS e verificar o controle remoto
  --skip-hostname-check Não verifique o nome do host do daemon contra o
                              nome especificado no certificado do cliente
  --projeto-diretório PATH Especifique um diretório de trabalho alternativo
                              (padrão: o caminho do arquivo Compor)
  -compatibilidade Se definido, o Compose tentará converter chaves
```

```
                                em arquivos v3 para o seu equivalente não-Swarm
  PATH -env-file Especifique um arquivo de ambiente alternativo

Comandos:
  build            Construir ou reconstruir serviços
  config           Validar e visualizar o arquivo Compor
  create           Criar serviços
  down             Pare e remova contêineres, redes, imagens e volumes
  events           Receba eventos em tempo real de contêineres
  exec             Execute um comando em um contêiner em execução
  help             Obter ajuda em um comando
  images           Liste imagens
  kill             Matar contêineres
  logs             Ver saída de contêineres
  pause            Serviços de pausa
  port             Imprima a porta pública para uma ligação de porta
  ps               Listar contêineres
  pull             Puxar imagens de serviço
  push             Pressione imagens de serviço
  restart          Reiniciar serviços
  rm               Remover recipientes parados
  run              Execute um comando único
  scale            Definir o número de contêineres para um serviço
  start            Iniciar serviços
  stop             Parar serviços
  top              Exibir os processos em execução
  unpause          Serviços de não uso
  up               Criar e iniciar contêineres
  version          Mostre as informações da versão Docker-Compose
```

Compose

O Docker compose tem o mesmo funcionamento que um Dockerfile, são criadas receitas para a montagem de ambientes em vez de uma unica imagem como o Dockerfile, o compose se utiliza de um arquivo .yml de nome docker-compose.yml, exemplo:

- **Arquivo:**

```
docker-compose.yml
```

- **Contêudo:**

```
root@debian:~/exemplo# cat docker-compose.yml
version: '3'
services:
        server1:
                image: ubuntu
```

```
        server2:
                image: ubuntu
```

Para um arquivo **COMPOSE** funcionar, o mesmo deve estar identado corretamente.

Agora, olha o **BUILD** denovo:

```
root@debian:~/exemplo# docker-compose build
Unable to find image 'docker/compose:1.26.2' locally
1.26.2: Pulling from docker/compose
aad63a933944: Pull complete
b396cd7cbac4: Pull complete
0426ec0ed60a: Pull complete
9ac2a98ece5b: Pull complete
Digest: sha256:b60a020c0f68047b353a4a747f27f5e5ddb17116b7b018762edfb6f7a6439a82
Status: Downloaded newer image for docker/compose:1.26.2
server1 uses an image, skipping
server2 uses an image, skipping
root@debian:~/exemplo#
```

O **BUILD** irá procurar pelo documento padrão de receita, o `docker-compose.yml`, com base nele, o mesmo irá iniciar a construção das imagens requeridas, há, lembrando, toda a alteração sobre os arquivos `.yml` deve passar **UP** para refletir no container.

Beleza, com o container's criado, agora só falta subir eles, veja:

```
docker-compose up
```

O resultado vai ser mais ou menos assim:

```
root@debian:~/exemplo# docker-compose up
Creating network "exemplo_default" with the default driver
Pulling server1 (ubuntu:)...
latest: Pulling from library/ubuntu
54ee1f796a1e: Already exists
f7bfea53ad12: Already exists
46d371e02073: Already exists
b66c17bbf772: Already exists
Digest: sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Status: Downloaded newer image for ubuntu:latest
Creating exemplo_server1_1 ... done
Creating exemplo_server2_1 ... done
Attaching to exemplo_server1_1, exemplo_server2_1
exemplo_server1_1 exited with code 0
exemplo_server2_1 exited with code 0
```

Como o container não está fazendo nada, ele simples se fecha, más vamos usar um **COMPOSE** melhor para explicar seu funcionamento.

Exemplo de arquivo docker-compose.yml mais real:

```
version: '3'

#Iniciando o file
services:

        #Criando o Banco
        banco:
                #Usa essa imagem para criar o phpmyadmin
                image: mysql:5.7
                #Libera essa porta
                ports:
                        - "3306:3306"
                #Setando variaveis para o banco
                environment:
                        MYSQL_ROOT_PASSWORD: root
                        MYSQL_DATABASE: projeto

        #Criando o php my admin somente para ter uma forma de adminstrar o mesmo
com mais facilidade
        phpmyadmin:
                #Usa essa imagem para criar o phpmyadmin
                image: phpmyadmin/phpmyadmin:latest
                #Depende do banco e já link eles na mesma rede
                depends_on:
                        - banco
                links:
                        - banco
                #Libera essa porta para conexão
                ports:
                        - "8080:80"
                environment:
                        - PMA_ARBITRARY=1

        #Criando o server de bk
        servidor_servico:
                #Usa essa imagem
                build:
                        context: .
                        dockerfile: Dockerfile
                #Cria o volume no root
                volumes:
                        - ./Sociedade/:/root
                #Depende do banco e já link eles na mesma rede
                depends_on:
                        - banco
                links:
                        - banco
```

```
                #Inicia o comando shell quando subir o banco
                command: bash /root/bk.sh

        #Criando o server de aplicacao
        servidor_aplicacao:
                #Usa essa imagem
                build:
                        context: .
                        dockerfile: Dockerfile
                #Cria o volume no root
                volumes:
                        - ./Sociedade/:/root
                #Depende do banco e já link eles na mesma rede
                depends_on:
                        - banco
                links:
                        - banco
                #Libera essa porta para conexão
                ports:
                        - "5000:5000"
                #Inicia o comando shell quando subir o banco
                command: bash /root/shell.sh
```

Melhor eu explicar o que algumas coisas aqui significam:

O `version:` é a versão do docker-compose utilizado, é algo obrigado a se colocar nos arquivos **YML**, `services:` são os containers que serão utilizados pelo compose durante o **BUILD** e o **UP** dos mesmos, após isso colocamos os nomes dos containers, como exemplo o `banco:`, vamos analisar só esse por um segundo:

```
services:

        #Criando o Banco
        banco:
                #Usa essa imagem para criar o phpmyadmin
                image: mysql:5.7
                #Libera essa porta
                ports:
                        - "3306:3306"
                #Setando variaveis para o banco
                environment:
                        MYSQL_ROOT_PASSWORD: root
                        MYSQL_DATABASE: projeto
```

O **SERVICES** está chamando o container **BANCO** para ser montado, dentro deste container se tem:

- `image:` que é a imagem utilizada para subir esse banco, no caso, já é um container com banco Mysql existente;
- `ports:` que é a liberação das portas do banco de interna para externa;
- `environment` possibilita setar várias ou realizar ações durante a criação do container;

Agora vamos a outro container, vamos ver.... esse:

```
        #Criando o php my admin somente para ter uma forma de adminstrar o mesmo
com mais facilidade
        phpmyadmin:
                #Usa essa imagem para criar o phpmyadmin
                image: phpmyadmin/phpmyadmin:latest
                #Depende do banco e já link eles na mesma rede
                depends_on:
                        - banco
                links:
                        - banco
                #Libera essa porta para conexão
                ports:
                        - "8080:80"
                environment:
                        - PMA_ARBITRARY=1
```

Dessa aqui só vamos ver essas duas opções, o `depends_on` e o `links`;

- `depends_on` é a dependencia de um container para o atual, caso o container estiver falhado em iniciar, esse container que depende dele tambem não irá nem iniciar;
- `links` é mais leve que o `depends_on`, esse cara tem o trabalho de somente fazer a conexão com outro container;

Agora vamos para o ultimo, esse aqui você vai gostar:

```
        #Criando o server de aplicacao
        servidor_aplicacao:
                #Usa essa imagem
                build:
                        context: .
                        dockerfile: Dockerfile
                #Cria o volume no root
                volumes:
                        - ./Sociedade/:/root
                #Depende do banco e já link eles na mesma rede
                depends_on:
                        - banco
                links:
                        - banco
                #Libera essa porta para conexão
                ports:
                        - "5000:5000"
                #Inicia o comando shell quando subir o banco
                command: bash /root/shell.sh
```

Aqui temos os:

- `build` é a forma de se utilizar um Dockerfile que você criou para subir uma imagem no compose, seus complementos são o `context` que irá falar para o compose o caminho do arquivo Dockerfile e o `dockerfile` nem precisa ser dito né;
- `volumes` é o mapeamento de um diretório interno para o diretório dentro do container, exemplo `./Sociedade/:/root`, isso é, do diretório sociedade para o diretório root dentro do container;
- `command` é a ordenação de operações durante o inicio do container, no caso, uma ordenação de execução de um shellScript via Bash;

Agora vamos terminar o compose com uma série de comandos que facilitam seu gerenciamento sobre o mesmo:

```
root@debian:~/exemplo# docker-compose up -d
Starting exemplo_server2_1 ... done
Starting exemplo_server1_1 ... done
```

O `docker-compose up -d` é a criação não taxando a saída ao terminal do usuário, assim você processo no seu terminal sem afetar os containers

- **PS:**

```
root@debian:~/exemplo# docker-compose ps
      Name            Command    State    Ports
-----------------------------------------------
exemplo_server1_1   /bin/bash   Exit 0
exemplo_server2_1   /bin/bash   Exit 0
```

- **PS -A:**

```
root@debian:~/exemplo# docker-compose ps -a
      Name            Command    State    Ports
-----------------------------------------------
exemplo_server1_1   /bin/bash   Exit 0
exemplo_server2_1   /bin/bash   Exit 0
```

Acho que você já deve tar meio careca de ver **PS** neste manual, más saiba que o compose tem os mesmo comandos e com as mesmas funções.

```
root@debian:~/exemplo# docker-compose start
Starting server1 ... done
Starting server2 ... done
```

O `compose start` reinicia os container que estão parados a partir do docker-compose.yml

---

Explicando a telinha

A telinha do compose mostra as seguintes informações:

- **PS -A:**

```
root@debian:~/exemplo# docker-compose ps -a
      Name            Command    State    Ports
-------------------------------------------------
exemplo_server1_1   /bin/bash   Exit 0
exemplo_server2_1   /bin/bash   Exit 0
```

Vamos explicar o que elas são:

Name	Command	State	Ports
Nome do container	Comando que o mesmo está executando	Estado atual do mesmo	Portas que o mesmo está usando

Nova empreitada

Agora, vou alterar um pouco meu arquivo `docker-compose.yml` que irá ficar assim:

```
root@debian:~/exemplo# cat docker-compose.yml
version: '3'
services:
        server1:
                image: nginx
                ports:
                        - 8080:80
```

Se qualquer um que mexe com compose ver isso, ele vai querer me matar por somente estar subindo um container com o mesmo, más... testes.

Se dermos os `docker-compose build` neste momento, o sistema irá reclamar que já existe um server1, veja:

```
root@debian:~/exemplo# docker-compose build
server1 uses an image, skipping
```

Isso porque o **server1** está usando uma outro imagem e ainda existe, para que possamos subir essa nova imagem poderia se criar um novo compose e executar, más vamos matar o antigo e iniciar o novo, para matar o atual, execute o:

```
docker-compose down
```

Isso irá **MATAR TODOS OS CONTAINERS ATUAIS DO COMPOSE**, só execute esse comando quando tiver certeza absoluta e em vez de fazer isso, você pode só matar o container desejado, exemplo:

```
root@debian:~/exemplo# docker-compose kill
Killing exemplo_server1_1 ... done
```

Entenda:

- **DOWN** mata todo mundo;
- **KILL** mata somente os atuais compose;

Upando denovo

Vamos dar um **UP** sobre as alterações do container após retirar as antigas dependências:

```
root@debian:~/exemplo# docker-compose up
Creating network "exemplo_default" with the default driver
WARNING: Found orphan containers (exemplo_server2_1) for this project. If you
removed or renamed this service in your compose file, you can run this command
with the --remove-orphans flag to clean it up.
Pulling server1 (nginx:)...
latest: Pulling from library/nginx
bf5952930446: Pull complete
cb9a6de05e5a: Pull complete
9513ea0afb93: Pull complete
b49ea07d2e93: Pull complete
a5e4a503d449: Pull complete
Digest: sha256:b0ad43f7ee5edbc0effbc14645ae7055e21bc1973aee5150745632a24a752661
Status: Downloaded newer image for nginx:latest
Creating exemplo_server1_1 ... done
Attaching to exemplo_server1_1
server1_1  | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will
attempt to perform configuration
server1_1  | /docker-entrypoint.sh: Looking for shell scripts in /docker-
entrypoint.d/
server1_1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-
ipv6-by-default.sh
server1_1  | 10-listen-on-ipv6-by-default.sh: Getting the checksum of
/etc/nginx/conf.d/default.conf
server1_1  | 10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
server1_1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-
templates.sh
server1_1  | /docker-entrypoint.sh: Configuration complete; ready for start up
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8080->80/tcp
```

```
root@debian:~/exemplo# curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Bem, é isso, notemos que o container subiu normal e que podemos até ver o NGINX funcionando na porta 8080 do HOST;

Descanso

Bem estamos perto do fim e bora dar uma pausa... pronto, há esquecia de avisar, lembra o `ctrl+p+q` que você usa para sair de um container sem matar ele, bem, funciona no compose tambem, caso você estiver ataxado nele, é só dar esse comando para voltar ao terminal, veja:

```
root@debian:~/exemplo# docker-compose up
WARNING: Found orphan containers (exemplo_server2_1) for this project. If you
removed or renamed this service in your compose file, you can run this command
with the --remove-orphans flag to clean it up.
Starting exemplo_server1_1 ... done
Attaching to exemplo_server1_1
server1_1  | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will
attempt to perform configuration
server1_1  | /docker-entrypoint.sh: Looking for shell scripts in /docker-
entrypoint.d/
server1_1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-
ipv6-by-default.sh
server1_1  | 10-listen-on-ipv6-by-default.sh: error: IPv6 listen already enabled
server1_1  | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-
```

```
templates.sh
server1_1  | /docker-entrypoint.sh: Configuration complete; ready for start up
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8080->80/tcp
```

Voltando para finalizar

Daqui por diante é só comando parecido com o docker comum, então é simples de se entender, veja:

```
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8080->80/tcp
root@debian:~/exemplo# docker-compose pause
Pausing exemplo_server1_1 ... done
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State           Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Paused   0.0.0.0:8080->80/tcp
root@debian:~/exemplo# docker-compose start
Starting server1 ... done
root@debian:~/exemplo# docker-compose kill
Killing exemplo_server1_1 ... done
root@debian:~/exemplo# docker-compose ps
      Name                     Command                State     Ports
-------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Exit 137
root@debian:~/exemplo# docker-compose start
Starting server1 ... done
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8080->80/tcp
root@debian:~/exemplo# docker-compose pause
Pausing exemplo_server1_1 ... done
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State           Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Paused   0.0.0.0:8080->80/tcp
root@debian:~/exemplo# docker-compose unpause
Unpausing exemplo_server1_1 ... done
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8080->80/tcp
```

De quebra toma um `pause`, `start`, `kill` e `ùnpause`

- `pause` para os container's atuais;
- `start` inicia esse container parados;
- `kill` como dito, mata esses containers;
- `unpause` faz exatamente o mesmo do `start`;

Temos também o comando `restart`, o mesmo irá reiniciar os containers atualmente criados, veja:

```
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8080->80/tcp
root@debian:~/exemplo# docker-compose pause
Pausing exemplo_server1_1 ... done
root@debian:~/exemplo# docker-compose start
Starting server1 ... done
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State           Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Paused   0.0.0.0:8080->80/tcp
root@debian:~/exemplo# docker-compose restart
Restarting exemplo_server1_1 ... done
root@debian:~/exemplo# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
exemplo_server1_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8080->80/tcp
```

Bem é isso

Obrigado por ter chegado até o fim e espero muito que tenha proveitado sobre esse rápido manual sobre o Docker e o Docker-compose, desculpe pelo erros ortográficos e por fim, volte de vez em quando, vai que eu dou uma atualizado neste manual, bem é isso, agradeço pela atenção e continue com os estudos;