

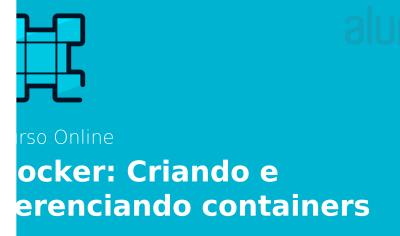
Docker: criando e gerenciando containers

- *Doc created by → Lucas Brito Rodrigues dos Santos*
- *Esta doc foi baseada no curso alura:*

Curso Online Docker: criando e gerenciando containers | Alura

O Docker é a maior ferramenta para o deploy de aplicações e microserviços. Nesse curso você aprenderá como criar imagens e rodar containers com o Docker.

 <https://cursos.alura.com.br/course/docker-criando-gerenciando-containers>



- *Busco demonstrar em laboratórios práticos realizados com base nesse curso as funções e a utilização do docker*

▼ Conhecendo o Docker

▼ Conhecendo o problema

- Temos duas aplicação nginx, java e C#
 - Java → Porta: 80 V : 17

- C# → Porta: 80 V : 9
- NGINX → Porta: 80 V : 1.17.0
- Utilizamos um computador para sustentar a aplicação
 - No caso podemos ter conflito de portas, alterar as versões de maneira prática é possível? Controle de recursos de memória e CPU? Processos e manutenção?
 - Como podemos gerenciar nossa aplicação?
- Uma maneira de resolver seria utilizar uma máquina para cada aplicação, porém o custo será muito maior
- As VMS podem virtualizar computadores em um computador host, isolando as aplicações, o que resolveria a maioria dos problemas → Mas será realmente necessária?
- O container irá isolar a aplicação e suas dependências do sistema operacional da máquina host

▼ Conflitos e versionamento

- Máquinas virtuais são capazes de isolar sistemas, com isso, o controle sobre a aplicação fica mais fácil.
- Containers podem isolar diversas aplicações, facilitando o controle acerca de portas e versões.

▼ Como containers funcionam

- Por que os containers são mais leves?
 - Os containers funcionam diretamente dentro do SO, como processos, assim a carga será menor
- Como garantem o isolamento?
 - Quando o container estiver em execução no SO, temos os Namespaces que fazer os isolamentos dos serviços que rodam dentro deles
 - PID → isolamento dos processos rodando dentro do container
 - NET → isolamento das interfaces de rede

- IPC → isolamento da comunicação entre processos e memória compartilhada
 - MNT → isolamento do sistema de arquivos / pontos de montagem
 - UTS → isolamento do kernel. Age como se o container fosse outro host
- Como funcionam sem instalar um SO?
 - Devido ao UTS → os containers terão um pedaço do kernel do SO porém de forma isolada
 - Como fica a divisão de recursos do sistema?
 - Cgroups → garantem como os consumos dos processos serão realizados
-
-

▼ Containers por baixo dos panos

- Containers podem isolar diversas aplicações, facilitando o controle acerca de portas e versões.

▼ Instalando o Docker no windows

Install Docker Desktop on Windows

Get started with Docker for Windows. This guide covers system requirements, where to download, and instructions on how to install and update.

 <https://docs.docker.com/desktop/install/windows-install/>



- O ideal é utilizar o docker no linux
 - WSL2 → O subsistema de linux para windows deve estar configurado no windows para a utilização do docker de forma mais otimizada
1. Efetue o download
 2. Efetue a instalação padrão e realize a configuração para habilitar no WSL2

3. Após instalação, aceite os termos

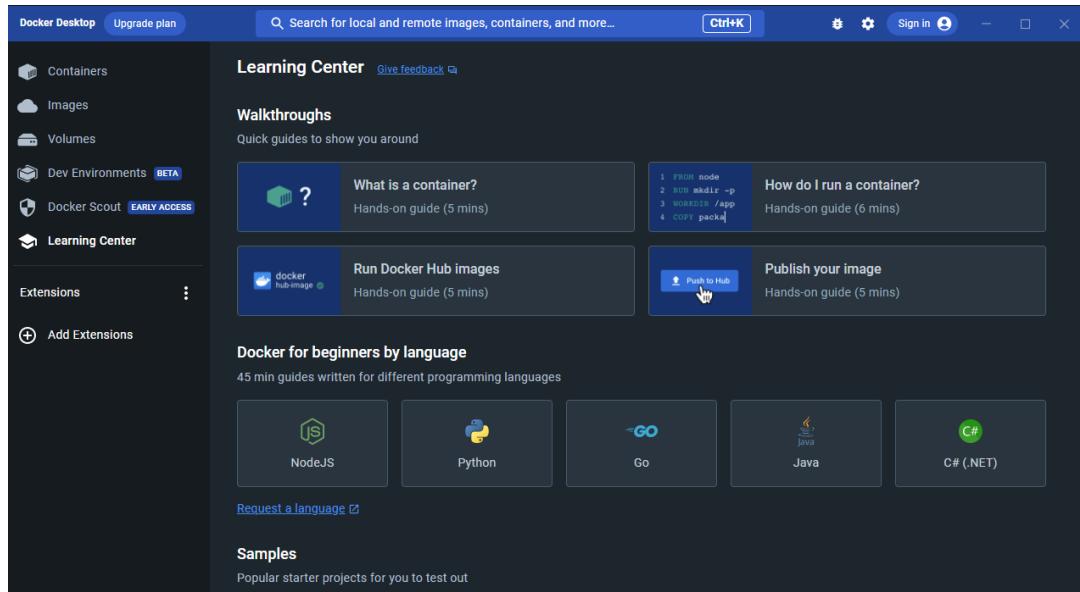
4. Abra o PWSH e rode o comando

- `docker run hello-world`

- Fique atento com os termos de uso do docker no windows para evitar custos

▼ Preparando o ambiente - Windows

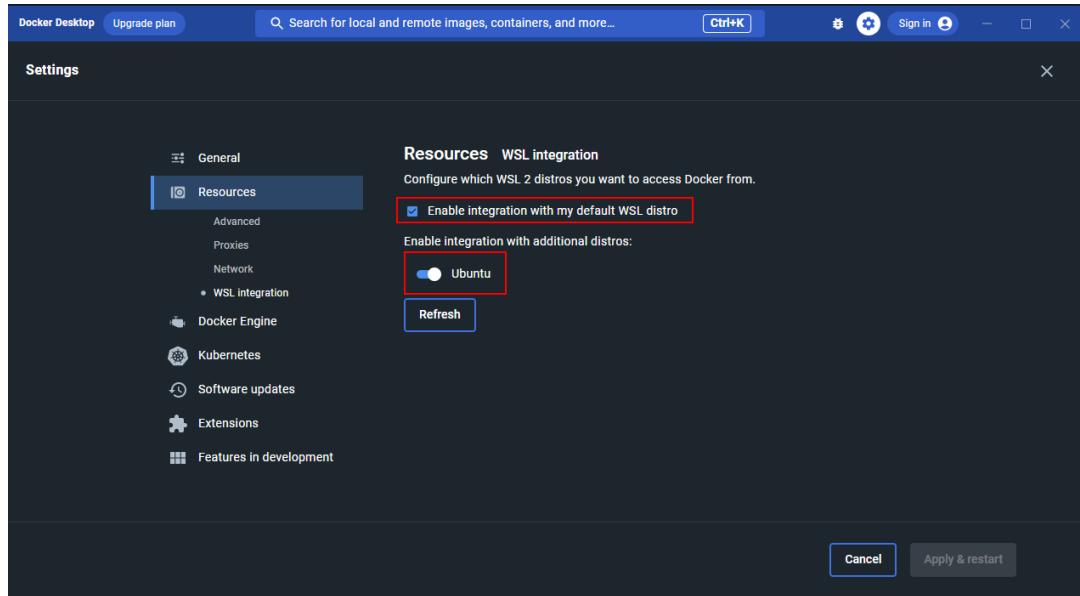
- Instalação do **WSL** (*Windows Subsystem for Linux* ou Subsistema Windows para Linux) no [link](#). O WSL irá garantir um ambiente no qual o Docker irá funcionar;
- Docker Desktop for Windows - Download através do [link](#).
- Após a instalação, crie uma conta gratuita no **Docker Hub** e garanta que esteja logado no Docker Desktop com a mesma. Em alguns casos, o próprio processo de instalação do Docker Desktop irá incluir o cadastro e login.
- Para começar com os comandos do Docker, abra o menu de busca do Windows e inicialize o programa denominado "Ubuntu" (instalado junto com o WSL). Isso irá abrir um bash de linha de comando no qual o curso todo poderá ser desenvolvido e todos os comandos utilizados pelo professor irão funcionar perfeitamente.
- Lembre-se de que o Docker para Windows possui restrições acerca de utilização e custos, sendo preferível usar a versão para Linux.
- No windows o docker possui uma interface de visualização



- O WSL deverá estar habilitado para sua utilização

```
PS C:\Users\LucasSantos> wsl -l -v
  NAME          STATE    VERSION
* Ubuntu        Running   2
  docker-desktop-data  Running   2
  docker-desktop    Running   2
PS C:\Users\LucasSantos> |
```

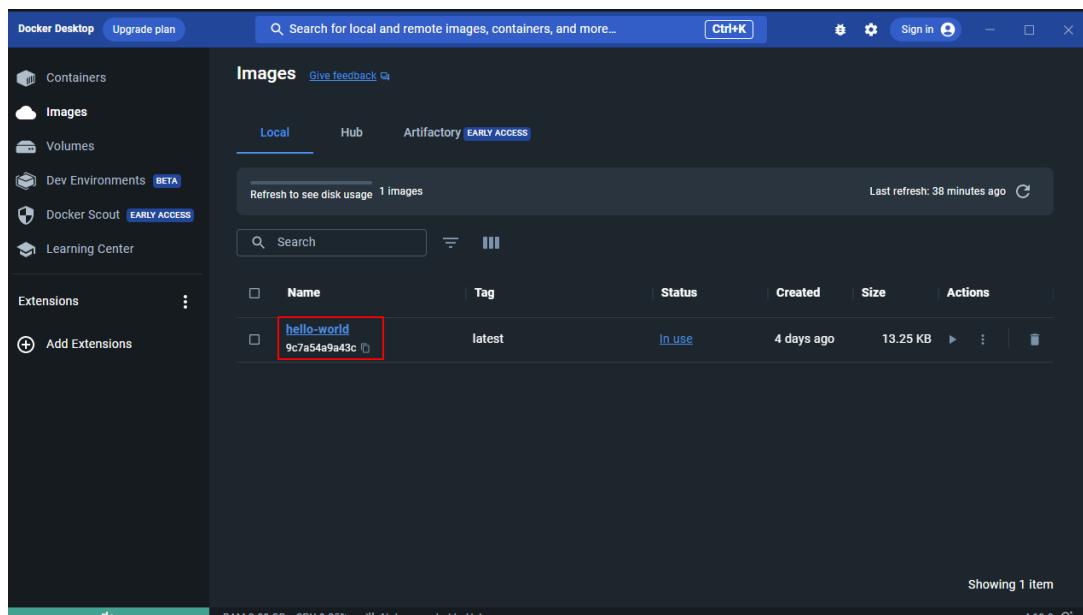
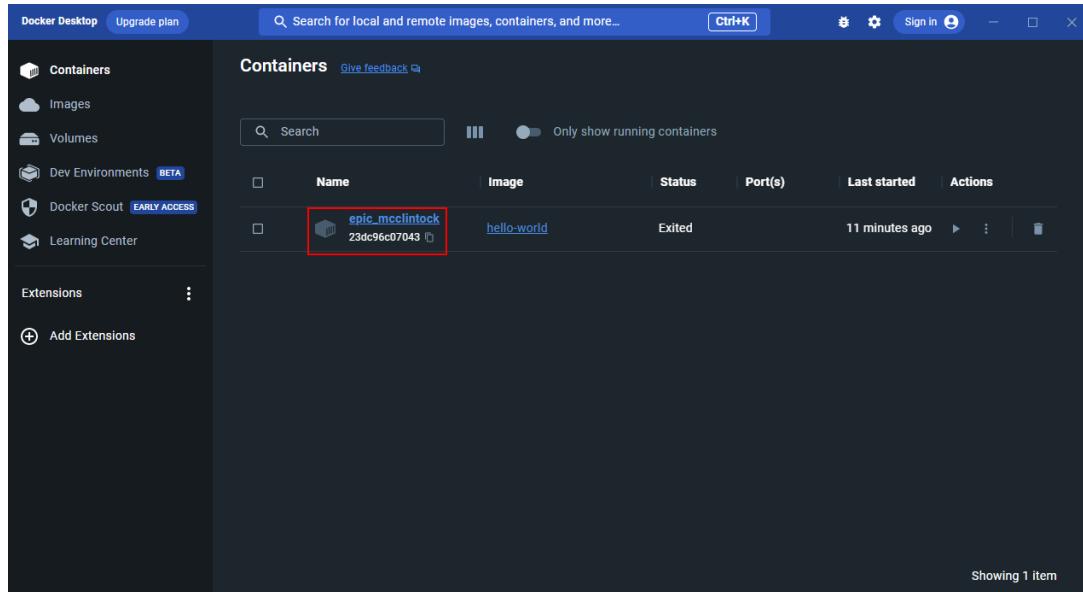
- Caso tivermos um WSL com uma distro (no exemplo ubuntu), podemos configurar para utilizar o docker dentro da distro



```
PS C:\Users\LucasSantos> wsl -l -v
  NAME          STATE      VERSION
* Ubuntu        Running     2
  docker-desktop-data  Running     2
  docker-desktop    Running     2
PS C:\Users\LucasSantos> wsl
lucasvertare@DESKTOP-2AEKE0L:/mnt/c/Users/LucasSantos$ docker --version
Docker version 23.0.6, build ef23cbc
```

- Assim os containers e imagens que formos baixando na própria distro, irá aparecer no docker desktop

```
lucasvertare@DESKTOP-2AEKE0L:/mnt/c/Users/LucasSantos$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
hello-world    latest   9c7a54a9a43c  4 days ago   13.3kB
lucasvertare@DESKTOP-2AEKE0L:/mnt/c/Users/LucasSantos$ docker ps -a
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
23dc96c07043    hello-world  "/hello"    10 minutes ago   Exited (0) 10 minutes ago
lucasvertare@DESKTOP-2AEKE0L:/mnt/c/Users/LucasSantos$
```



▼ Caso a distro instalada for a wsl -v 1 siga os passos deste procedimento

1. Abra o PowerShell como administrador.
2. Liste todas as distribuições WSL instaladas em seu sistema executando o seguinte comando:

```
wsl --list --verbose
```

O resultado exibirá todas as distribuições WSL instaladas em seu sistema, incluindo a versão atual e o local do arquivo VHD.

1. Selecione a distribuição WSL que você deseja converter para WSL2 e execute o seguinte comando:

```
wsl --set-version <Distro> 2
```

Substitua `<Distro>` pelo nome da distribuição WSL que deseja converter. Por exemplo, se você deseja converter a distribuição Ubuntu, execute o seguinte comando:

```
wsl --set-version Ubuntu 2
```

1. Aguarde até que a conversão esteja concluída. O tempo necessário para a conversão pode variar dependendo do tamanho da distribuição e da velocidade do seu computador.

Depois que a conversão estiver concluída, você pode verificar se a distribuição foi convertida com sucesso executando o seguinte comando:

```
wsl --list --verbose
```

O resultado mostrará a versão atual da distribuição WSL, que deve ser 2.

Observe que a conversão de uma distribuição WSL de WSL1 para WSL2 é irreversível e não pode ser desfeita. Certifique-se de fazer backup de seus dados importantes antes de prosseguir com a conversão.

▼ Instalando o Docker no linux

Install Docker Engine on Ubuntu

Jumpstart your client-side server applications with Docker Engine on Ubuntu. This guide details prerequisites and multiple methods to install.

 <https://docs.docker.com/engine/install/ubuntu/>



- Seguindo os passos da documentação é possível realizar a instalação tranquilamente
- Após a instalação precisamos rodas o docker utilizando o “sudo” na frente
- O docker docs sugere uma configuração para permitir rodar o docker sem o sudo na frente

Linux post-installation steps for Docker Engine

Find the recommended Docker Engine post-installation steps for Linux users, including how to run Docker as a non-root user and more.

 <https://docs.docker.com/engine/install/linux-postinstall/>



▼ Preparando o ambiente no linux

- Caso esteja utilizando outra distribuição Linux, é possível consultar no menu à esquerda da página outras distribuições possíveis para instalação do Docker.
- Após instalar, lembre-se de executar o comando `sudo usermod -aG docker $USER` e reiniciar sua sessão para utilizar o Docker sem precisar usar o `sudo`.
 - Docker Ubuntu - Download através do [link](#).

Muitos sistemas atualmente são compostos por diversas aplicações executadas simultaneamente. Diversos problemas podem ocorrer quando muitas aplicações precisam se comunicar. Porém, antes de pensarmos em comunicação de aplicações, é preciso visualizar outras questões importantes para nosso trabalho, como isolamento de contextos e versionamento de aplicações.

- Máquinas virtuais possuem camadas adicionais de virtualização em relação a um container;
- Containers funcionam como processos no host;
- Containers atingem isolamento através de namespaces;
- Os recursos dos containers são gerenciados através de cgroups.

▼ Os primeiros Comandos

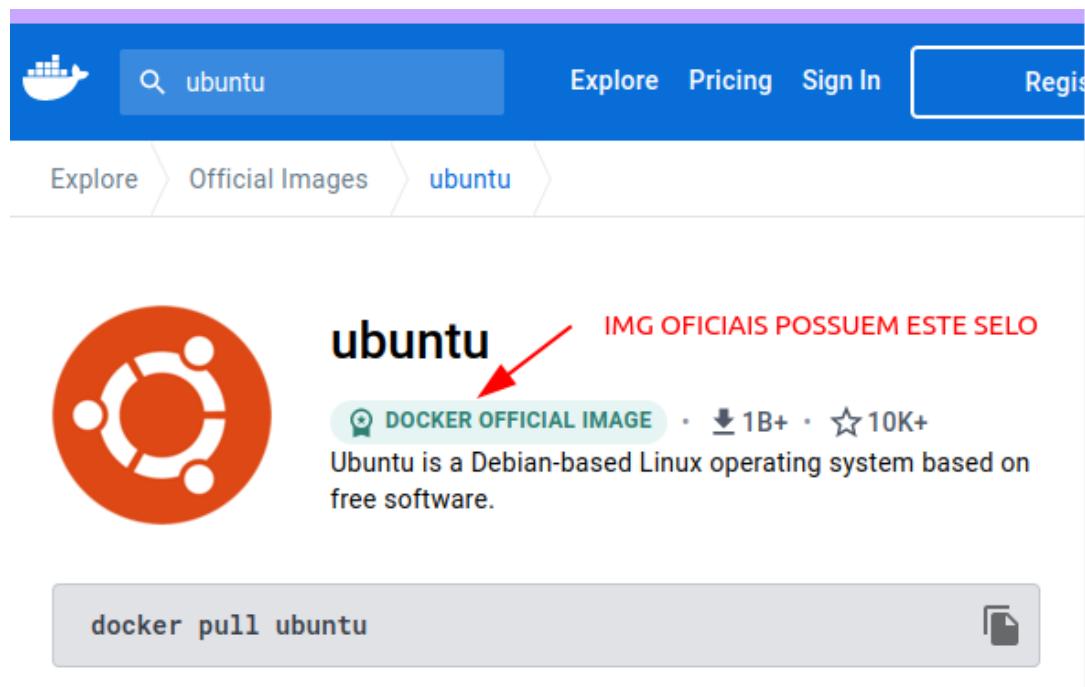
▼ Conhecendo o Docker hub

- Quando rodamos o 'docker run', ele irá buscar as imagens em um repositório de imagens, como o docker hub

Docker Hub Container Image Library | App Containerization

 <https://hub.docker.com/>

- Ou seja não é qualquer imagem que iremos subir usando o 'docker run', serão aquelas que estão no repositório
- Podemos rodar containers baseados em sistemas operacionais como o ubuntu



The screenshot shows the Docker Hub interface. At the top, there's a search bar with 'ubuntu' typed into it, along with navigation links for 'Explore', 'Pricing', 'Sign In', and 'Register'. Below the search bar, the breadcrumb navigation shows 'Explore > Official Images > ubuntu'. The main content area displays the official Ubuntu Docker image. It features the Ubuntu logo (a red circle with a white icon), the word 'ubuntu' in black, and a green button labeled 'DOCKER OFFICIAL IMAGE'. To the right of the image, a red arrow points to the 'DOCKER OFFICIAL IMAGE' button with the text 'IMG OFICIAIS POSSUEM ESTE SELO'. Below the image, a description reads: 'Ubuntu is a Debian-based Linux operating system based on free software.' At the bottom of the screenshot, there's a grey bar containing the command 'docker pull ubuntu' and a download icon.

- Podemos rodar o “docker pull ” para baixar a imagem

```
lk47@linux-labs:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
dbf6a9befcde: Pull complete
Digest: sha256:0b8a884ad7ed20e50a68108895a6e7db54b303ad73b22b6b9ff5f60740dc03e2
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
lk47@linux-labs:~$ docker run ubuntu
lk47@linux-labs:~$
```

Img baixada

Rodando o container

▼ O comando docker run

- Através deste comando, o docker irá executar o container da maneira esperada.

▼ Fluxo de criação de containers

- Após baixar e exutar o container com o `docker run`
- Podemos usar o `docker ps` → Irá mostar os containers em execução

COLUMN	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1	0d306ea72e40	ubuntu	"/bin/bash"	13 minutes ago	Exited (0) 13 minutes ago		hopeful_goldberg

- `docker container ls` → também realiza o mesmo procedimento

COLUMN	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1	6292815bcf5c	hello-world	"/hello"	39 minutes ago	Exited (0) 39 minutes ago		unruffled_austin

- `docker container ls -a` ou `docker ps -a` → mostra todos os container que estiveram ou estão em execução

COLUMN	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1	0d306ea72e40	ubuntu	"/bin/bash"	13 minutes ago	Exited (0) 13 minutes ago		hopeful_goldberg
2	6292815bcf5c	hello-world	"/hello"	39 minutes ago	Exited (0) 39 minutes ago		unruffled_austin
3	fe4c18e3b70b	hello-world	"/hello"	39 minutes ago	Exited (0) 39 minutes ago		frosty_jang

- `Container ID` → mostra o ID unico de cada container
- `IMAGE` → imagem de cada comando
- `COMMAND` → o comando que foi executado ao criar o container
- `CREATED` → quando foi criado
- `STATUS` → exited é por que ele foi executado e saiu
- `PORTS` → mais pra frente
- `NAMES` → nome criado automaticamente quando não definimos
- Logo quando rodamos o `docker run ubuntu`, ele executa o bash, e pronto, ele desliga e é encerrado
 - Para o container ficar em execução um processo deve estar dentro dele
- Portanto se rodar um `docker run ubuntu sleep 1d` → quando subir vai travar o container por 1 dia

```
lk47@linux-labs:~$ docker run ubuntu sleep 1d
```

- Assim rodando um `docker ps` podemos ver o container em execução

```
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED    STATUS        PORTS     NAMES
f251923c1004   ubuntu    "sleep 1d"   7 seconds ago   Up 6 seconds   nervous_lalande
lk47@linux-labs:~$
```

▼ Etapas do run

Procura a imagem localmente -> Baixa a imagem caso não encontre localmente -> Valida o hash da imagem -> Executa o container.

- Este é o fluxo adotado pelo docker para execução de um container.

▼ Outros comandos importantes

- `docker stop (id ou nome container)` → Para o processo em container e consequentemente o container
 - Podemos colocar a flag `-t=0` assim parar de imediato

```
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS      PORTS
ATUS          ubuntu     "sleep 1d"   About a minute ago
              3 seconds   competent_dubinsky
lk47@linux-labs:~$ docker stop -t=0 c2891510b773
c2891510b773
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS      PORTS
NAMES
lk47@linux-labs:~$ 
Repare que através da flag -t=0, estamos informando que a
```

- `docker start (id ou nome do container)` → irá reiniciar o container que foi stopado

```
lk47@linux-labs:~$ docker start c2891510b773
c2891510b773
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS
              PORTS      NAMES
c2891510b773   ubuntu     "sleep 1d"   2 minutes ago   Up 3 se
conds          competent_dubinsky
lk47@linux-labs:~$
```

- `docker exec -it (id ou nome do container) bash` → acessa o container que criamos

```

lk47@linux-labs:~$ docker ps
CONTAINER ID        IMAGE       COMMAND      CREATED      STATUS
              PORTS     NAMES
c2891510b773        ubuntu      "sleep 1d"   3 minutes ago   Up About a minute
                                         competent.dubinsky
lk47@linux-labs:~$ docker exec -it c2891510b773 bash
root@c2891510b773:/# ls
bin  etc  lib32  media  proc  sbin  tmp
boot  home  lib64  mnt  root  srv  usr
dev  lib  libx32  opt  run  sys  var
root@c2891510b773:/# █

```

```

top - 18:45:59 up 1:19, 0 users, load average: 0.57, 0.17, 0.06
Tasks: 3 total, 1 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.0 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 969.5 total, 111.1 free, 258.0 used, 600.4 buff/cache
MiB Swap: 1378.0 total, 1378.0 free, 0.0 used. 562.3 avail Mem

          PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM TIME+ COMMAND
            1 root      20   0   2788  1012   924 S  0.0  0.1  0:00.03 sleep
            7 root      20   0   4624  3744  3180 S  0.0  0.4  0:00.04 bash
           18 root      20   0   7292  3364  2832 R  0.0  0.3  0:00.04 top

```

- `-it` → Interactive
- MNT → permite isolar o arquivo do sistemas
 - Tudo o que fizermos dentro do container estará isolado do sistema operacional
- Se resetarmos o container toda a arvore de processo será reiniciada
- `docker (un)pause (id ou nome do container)` → O container irá pausar/despausar e a arvore processos será mantida

```

lk47@linux-labs:~$ docker pause c2891510b773
c2891510b773
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS
              PORTS      NAMES
c2891510b773   ubuntu     "sleep 1d"    6 minutes ago  Up 3 min
utes (Paused)                           competent_dubinsky
lk47@linux-labs:~$ docker unpause c2891510b773
c2891510b773
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS
              PORTS      NAMES
c2891510b773   ubuntu     "sleep 1d"    6 minutes ago  Up 4 min
utes                           competent_dubinsky
lk47@linux-labs:~$ █

```

- `docker rm (id ou nome do container)` → remove o container
 - Deixando de existir todo o conteúdo será perdido

```

lk47@linux-labs:~$ docker rm c2891510b773
c2891510b773
lk47@linux-labs:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS
              PORTS      NAMES
f251923c1004   ubuntu     "sleep 1d"    47 minutes ago
                Exited (137) 43 minutes ago
0d306ea72e40   ubuntu     "/bin/bash"  About an hour ago
                Exited (0) About an hour ago
6292815bcf5c   hello-world "/hello"    About an hour ago
                Exited (0) About an hour ago
fe4c18e3b70b   hello-world "/hello"    About an hour ago
                Exited (0) About an hour ago
lk47@linux-labs:~$ █

```

- `docker run -it + Image (ubuntu) + comando (bash)` → cria um container interativo
 - Porém saindo do terminal, o container irá morrer

```

lk47@linux-labs:~$ docker run -it ubuntu bash
root@5c60eaaedf21:/# ls
bin etc lib32 media proc sbin tmp
boot home lib64 mnt root srv usr
dev lib libx32 opt run sys var
root@5c60eaaedf21:/# touch test.txt
root@5c60eaaedf21:/# ls
bin etc lib32 media proc sbin test.txt var
boot home lib64 mnt root srv tmp
dev lib libx32 opt run sys usr
root@5c60eaaedf21:/# exit
exit
lk47@linux-labs:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
lk47@linux-labs:~$ 

```

▼ Run vs Exec

- O `docker run` cria um novo container e o executa. O `docker exec` permite executar um comando em um container que já está em execução.

▼ Mapeando portas

- É uma boa prática usar imagens oficiais
- Porém podemos usar imagens não oficiais
- `docker run -d` → Roda o container em background sem travar o terminal
- No caso vamos baixar o dockersample/static-site
 - `docker run -d dockersamples/static-site`

```

lk47@linux-labs:~$ docker run -d dockersamples/static-site
Untagged: dockersamples/static-site:latest
Digest: sha256:da0a8cc61d7d230b797e72157997489db49f316b9b9af3909df10fd28b2dec
Status: Downloaded newer image for dockersamples/static-site:latest
e7454843183f:a995d5ebcf9a04a72f5e40a0a622a5d82106edb81d6f85b803dd
lk47@linux-labs:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e7454843183f dockersamples/static-site "/bin/sh -c 'cd /usr'" 25 seconds ago Up 22 seconds 80/tcp, 443/tcp eloquent_cori
lk47@linux-labs:~$ 

```

- COMMAND → `/bin/sh -c 'cd/usr'` → mantém o container em execução
- PORTS → `80/tcp, 443/tcp` → Porta em que está sendo executada
 - NET → Os containers possuem as interfaces de rede isoladas, portanto mesmo que eu vá no `localhost` no navegador não será encontrado

- No caso se quisermos acessar precisamos expor
- Vamos usar o `docker rm id --force` irá parar e remover o container

```
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE          COMMAND
CREATED       STATUS        PORTS
AMES
e7454843183f  dockersamples/static-site "/bin/sh -c 'cd /usr
"  About a minute ago  Up About a minute  80/tcp, 443/tcp
loquent_cori
lk47@linux-labs:~$ docker rm e7454843183f --force
e7454843183f
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE          COMMAND
CREATED       STATUS        PORTS
NAMES
lk47@linux-labs:~$
```

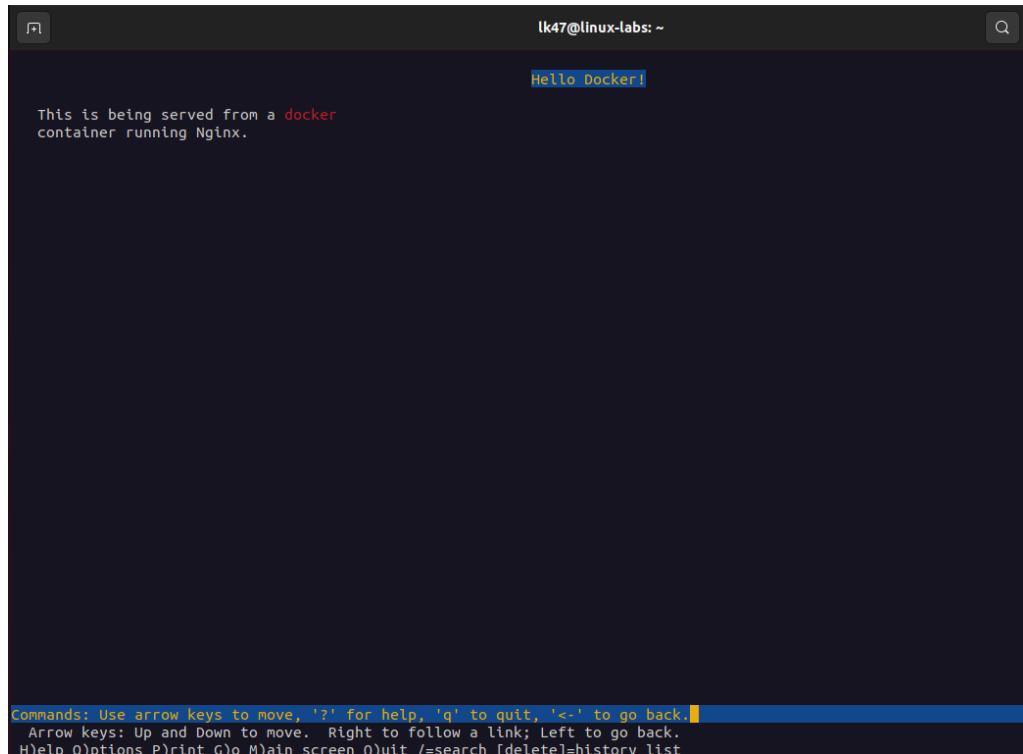
- `docker run -d -P dockersamples/static-site` → vamos add o -P

```
lk47@linux-labs:~$ docker run -d -P dockersamples/static-site
d7c63e3200357443b26e1b207913dc9e5e664d422fe24dd83750db910ec65a
lk47@linux-labs:~$ docker ps
CONTAINER ID   IMAGE          COMMAND
CREATED       NAMES        STATUS        PORTS  Variados
d7c63e320035  dockersamples/static-site "/bin/sh -c 'cd /usr"  5 seconds ago  Up 4 seconds  0.0.0.0:32769->80/tcp, :::32769->
80/tcp, 0.0.0.0:32768->443/tcp, :::32768->443/tcp  distracted_northcutt
lk47@linux-labs:~$
```

- `docker port id` → mostra o mapeamento do host
- Assim podemos acessar o conteúdo do nosso container

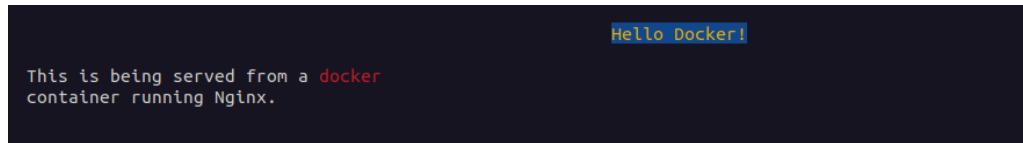
```
lk47@linux-labs:~$ docker port d7c63e320035
80/tcp -> 0.0.0.0:32769
80/tcp -> [::]:32769
443/tcp -> 0.0.0.0:32768
443/tcp -> [::]:32768
lk47@linux-labs:~$
```

```
lk47@linux-labs:~$ lynx http://0.0.0.0:32769
```



- `-p` → com o menos p minusculo podemos especificar a porta

```
lk47@linux-labs:~$ docker run -d -p 8080:80 dockersamples/static-site
98c82e7852832b04e3a4f443afb69baea59cab8c8c4e4263c2687f35091c86166
lk47@linux-labs:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
98c82e785283        dockersamples/static-site   "/bin/sh -c 'cd /usr..."   4 seconds ago      Up 2 seconds       443/tcp, 0.0.0.0:8080->80/tcp
lk47@linux-labs:~$ docker port 98c82e785283
80/tcp -> 0.0.0.0:8080
80/tcp -> [::]:8080
lk47@linux-labs:~$ lynx http://0.0.0.0:8080
```



▼ Visualizando mapemantos

- `docker port`
 - Este comando é responsável por exibir como o mapeamento de portas de um container está sendo feito.
 - **O Docker Hub é um grande repositório de imagens que podemos utilizar;**

- A base dos containers são as imagens;
- Como utilizar comandos acerca do ciclo de vida dos containers, como: `docker start`, para iniciar um container que esteja parado; `docker stop`, para parar um que esteja rodando; `docker pause`, para pausar um container e `docker unpause` para iniciar um container pausado; -Conseguimos mapear portas de um container com as flags `p` e `P`.

▼ Criando e empreendendo imagens

▼ Entendendo imagens

- Imagens → conjunto de camadas quando juntas formam uma imagem
- `docker images ou docker images ls` → mostra imagens

```
lk47@linux-labs:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
ubuntu              latest   3b418d7b466a  8 days ago    77.8MB
hello-world         latest   feb5d9fea6a5  19 months ago  13.3kB
dockersamples/static-site  latest   f589ccde7957  7 years ago   191MB
lk47@linux-labs:~$
```

- `docker inspect + id da imagem` → vemos detalhes da imagem

```
lk47@linux-labs:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
ubuntu              latest   3b418d7b466a  8 days ago    77.8MB
hello-world         latest   feb5d9fea6a5  19 months ago  13.3kB
dockersamples/static-site  latest   f589ccde7957  7 years ago   191MB
lk47@linux-labs:~$ docker inspect f589ccde7957
[
  {
    "Id": "sha256:f589ccde7957fa3ddf76a2eeee4d2f5d687b32176f559b703b6b8cacf6d36bc4",
    "RepoTags": [
      "dockersamples/static-site:latest"
    ],
    "RepoDigests": [
      "dockersamples/static-site@sha256:daa686c61d7d239b7977e72157997489db49f316b9af3909d9f10fd28b2dec"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2016-03-18T10:59:54.367126Z",
    "Container": "5d69bea014449857807d4fe490d635a65d5cf42ed5e1b615bc18ef6a415fa66",
    "ContainerConfig": {
      "Hostname": "esc68db50333",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "ExposedPorts": {
        "443/tcp": {},
        "80/tcp": {}
      },
      "Tty": false,
      "OpenStdin": false,
      "StdinOnce": false,
      "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "NGINX_VERSION=1.9.12-1-jessie"
      ]
    }
  }
]
```

- Layers são as camadas

```

    "Type": "layers",
    "Layers": [
        "sha256:917c0fc99b353c0397a9effdf042d72529de7c452669b1e11b05bec3088c7056",
        "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",
        "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",
        "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",
        "sha256:f0d7d68f89e518cc65ffb6892c4c21f135a9140ffc1911afe86521fff2c9ed17",
        "sha256:3f3324023e75cc3fb9f72fee3a13b39fb5cf46f8c234a7c582c3ae69089911c3",
        "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",
        "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",
        "sha256:b46149abb3e1c05e43a0e2a8c52e78a2cfbe8572b0a5f0234887e0f179258660"
    ],
    "Metadata": {
        "LastTagTime": "0001-01-01T00:00:00Z"
    }
}

```

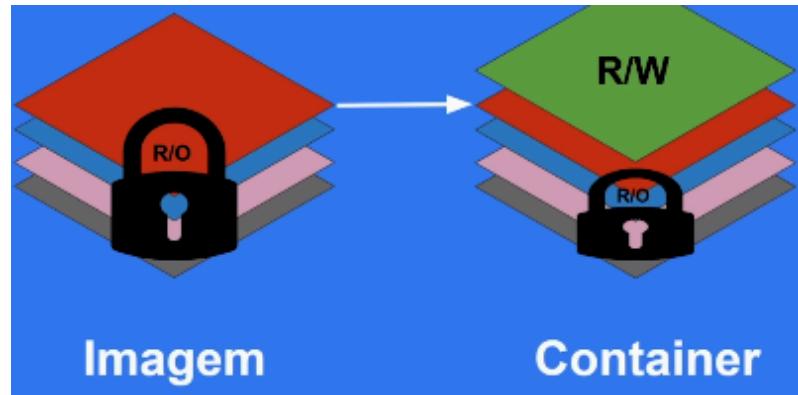
- `docker history + id da imagem` → vemos as camadas da imagem

```

lk47@linux-labs:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
ubuntu              latest   3b418d7b466a  8 days ago   77.8MB
hello-world         latest   feb5d9fea6a5  19 months ago  13.3kB
dockersamples/static-site  latest   f589ccde7957  7 years ago   191MB
lk47@linux-labs:~$ docker history f589ccde7957
IMAGE      CREATED      CREATED BY
f589ccde7957  7 years ago   /bin/sh -c #(nop) CMD ["/bin/sh" "-c" "cd /u... 0B
<missing>  7 years ago   /bin/sh -c #(nop) WORKDIR /usr/share/nginx/h... 0B
<missing>  7 years ago   /bin/sh -c #(nop) COPY file:c8203f6bfe2ff6ba... 8.75kB
<missing>  7 years ago   /bin/sh -c mkdir -p /usr/share/nginx/html 0B
<missing>  7 years ago   /bin/sh -c #(nop) ENV AUTHORITY=Docker 0B
<missing>  7 years ago   /bin/sh -c #(nop) CMD ["nginx" "-g" "daemon ... 0B
<missing>  7 years ago   /bin/sh -c #(nop) EXPOSE 443/tcp 80/tcp 0B
<missing>  7 years ago   /bin/sh -c ln -sf /dev/stdout /var/log/nginx... 22B
<missing>  7 years ago   /bin/sh -c apt-key adv --keyserver hkp://pgp... 65.4MB
<missing>  7 years ago   /bin/sh -c #(nop) ENV NGINX_VERSION=1.9.12-1... 0B
<missing>  7 years ago   /bin/sh -c #(nop) MAINTAINER NGINX Docker Ma... 0B
<missing>  7 years ago   /bin/sh -c #(nop) CMD ["/bin/bash"] 0B
<missing>  7 years ago   /bin/sh -c #(nop) ADD file:b5391cb13172fb513... 125MB
lk47@linux-labs:~$ 

```

- No nosso host pode haver as camadas/imagens que precisamos, isso nos permite já utilizarmos as que estão locais sem necessariamente pegar do docker hub
- Uma imagem é read only, não podemos alterar após criada
- Um container nada mais é de uma camada a mais na imagem, onde podemos escrever as informações enquanto o processo da imagem se encontra em utilização



- Os containers após concluir o processo o mesmo é encerrado e as modificações realizadas na imagem, serão excluidas

▼ Detalhes sobre imagens

- As camadas são a menor unidade que compõem uma imagem.
- `docker history` → Este comando é responsável por exibir quais são as camadas de uma imagem.

▼ Criando a primeira imagem

- Neste procedimento vamos subir um site utilizando o node.js, abaixa o arquivo de exemplo
 - <https://github.com/danielartine/alura-docker/blob/aula-3/app-exemplo.zip?raw=true>
- Para isso precisaremos subir alguns arquivos em uma imagem para subir via container

```
lk47@linux-labs:~$ ls
app-exemplo app-exemplo.zip
lk47@linux-labs:~$ cd app-exemplo/
lk47@linux-labs:~/app-exemplo$ ls
index.html index.js main.css node_modules package.json
lk47@linux-labs:~/app-exemplo$
```

- Portanto vamos criar a imagem dockerfile dentro da pasta

```
lk47@linux-labs:~/app-exemplo$ touch dockerfile
lk47@linux-labs:~/app-exemplo$ ls
dockerfile index.html index.js main.css node_modules package.json
lk47@linux-labs:~/app-exemplo$
```

- Dentro do arquivo dockerfile, vamos especificar os seguintes comandos

```
FROM node:14
#Especifica que a imagem que estamos criando está sendo com base em uma já existente no dockerhub, no caso é a node versão 14
WORKDIR /app-node
#Este comando está especificando a pasta que será criada na imagem (Pasta de trabalho), no caso onde os arquivos serão atrelados
COPY . .
# Este comando está copiando todos os arquivos da pasta atual (no caso app-exemplo), para a pasta de trabalho que será criada na imagem (app-node)
RUN npm install
# Este comando vai rodar e instalar o node
ENTRYPOINT npm start
# Este comando irá iniciar o node, assim definimos que, ao executar o container gerado por esta imagem, o comando executado será o npm start.
```

```
lk47@linux-labs:~/app-exemplo$ cat dockerfile
FROM node:14
WORKDIR /app-node
COPY . .
RUN npm install
ENTRYPOINT npm start
lk47@linux-labs:~/app-exemplo$
```

- Ainda dentro da pasta pelo terminal, iremos rodar o comando
 - `docker build -t <seu-nome-de-usuario-do-docker-hub>/app-node:1.0 .`
 - Sendo a flag `-t` para etiquetar a imagem com um nome
 - E o `.` no final serve para referenciar todos os arquivos dentro da pasta 'app-exemplo'

```

lk47@linux-labs:~/app-exemplo$ docker build -t lk47-labs/app-node:1.0 .
[+] Building 79.6s (9/9) FINISHED
--> [internal] load build definition from dockerfile                                0.2s
--> => transferring dockerfile: 114B                                              0.0s
--> [internal] load .dockerignore                                                 0.2s
--> => transferring context: 28                                                 0.0s
--> [internal] load metadata for docker.io/library/node:14                           2.8s
--> [1/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c5      69.9s
--> => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c59     0.1s
--> => sha256:1d12470fa4662a2a5cb50378dc8ea228c1735747db 7.51kB / 7.51kB  0.0s
--> => sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbc 776B / 776B  0.0s
--> => sha256:2cafa3fb0b6529ee4726b4f599ec27ee557ea3dea7 2.21kB / 2.21kB  0.0s
--> => sha256:2ff1d7c41c74a25258bfaf0b8adb0a727f84518 50.45MB / 50.45MB 11.2s
--> => sha256:3d2201bd995cccf12851a50820de03d34a17011dc 10.00MB / 10.00MB 3.8s
--> => sha256:b253aaefaea7e0671bb60008df01de101a38a045ff7 7.86MB / 7.86MB 5.0s
--> => sha256:1de76e268b103d05fa8960e0f77951ff54b912b6 51.88MB / 51.88MB 12.1s
--> => sha256:d9a8df5894511ce28a05e2925a75e8a4acbd06 191.85MB / 191.85MB 29.8s
--> => sha256:6f51ee005deac0d99898e41b8ce60ebf250ebe1a31 4.19kB / 4.19kB 11.6s
--> => sha256:f5f32ed3c3f278edda4fc571c880b5277355a29ae 35.24MB / 35.24MB 19.6s
--> => extracting sha256:2f1d7c41c74a25258bfaf0b8adb0a727f84518f55f65c 12.0s
--> => sha256:0c8cc2f24a4dc64e602e086fc9446b0a541e8acd9 2.29MB / 2.29MB 12.8s
--> => sha256:0d27a8e861329007574c6766fb946d48e20d2c8e964e873 450B / 450B 13.2s
--> => extracting sha256:b253aaefaea7e0671bb60008df01de101a38a045ff7bc656 1.7s
--> => extracting sha256:3d2201bd995cccf12851a50820de03d34a17011dcbb9ac9f 1.4s
--> => extracting sha256:1de76e268b103d05fa8960e0f77951ff54b912b63429c34f 7.3s
--> => extracting sha256:d9a8df5894511ce28a05e2925a75e8a4acbd0634c39ad73 22.4s
--> => extracting sha256:6f51ee005deac0d99898e41b8ce60ebf250ebe1a31a0b03f 0.0s
--> => extracting sha256:f5f32ed3c3f278edda4fc571c880b5277355a29ae8f52b52c 5.5s
--> => extracting sha256:0c8cc2f24a4dc64e602e086fc9446b0a541e8acd9ad72d2 0.2s
--> => extracting sha256:0d27a8e861329007574c6766fb946d48e20d2c8e964e873 0.0s
--> [internal] load build context                                                 0.4s
--> => transferring context: 1.15MB                                             0.2s
--> [2/4] WORKDIR /app-node                                                    1.0s
--> [3/4] COPY . . . . .                                                       0.8s
--> [4/4] RUN npm install                                                       4.2s
--> => exporting to image                                                       0.4s
--> => exporting layers                                                       0.3s

```

- Feito isso podemos ver a imagem através do `docker images`

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
lk47-labs/app-node	1.0	ba9da2de3783	2 minutes ago	913MB
ubuntu	latest	3b418d7b466a	8 days ago	77.8MB
hello-world	latest	feb5d9fea6a5	19 months ago	13.3kB
dockersamples/static-site	latest	f589ccde7957	7 years ago	191MB

- Com a imagem pronta agora vamos subir o container

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
lk47-labs/app-node	1.0	ba9da2de3783	5 minutes ago	913MB
ubuntu	latest	3b418d7b466a	8 days ago	77.8MB
hello-world	latest	feb5d9fea6a5	19 months ago	13.3kB
dockersamples/static-site	latest	f589ccde7957	7 years ago	191MB

Para rodar em background

No arquivo da aplicação está especificando a porta 3000, portanto vamos pontuar a mesma vamos usar a 8081 pois a 8080 está em uso

```

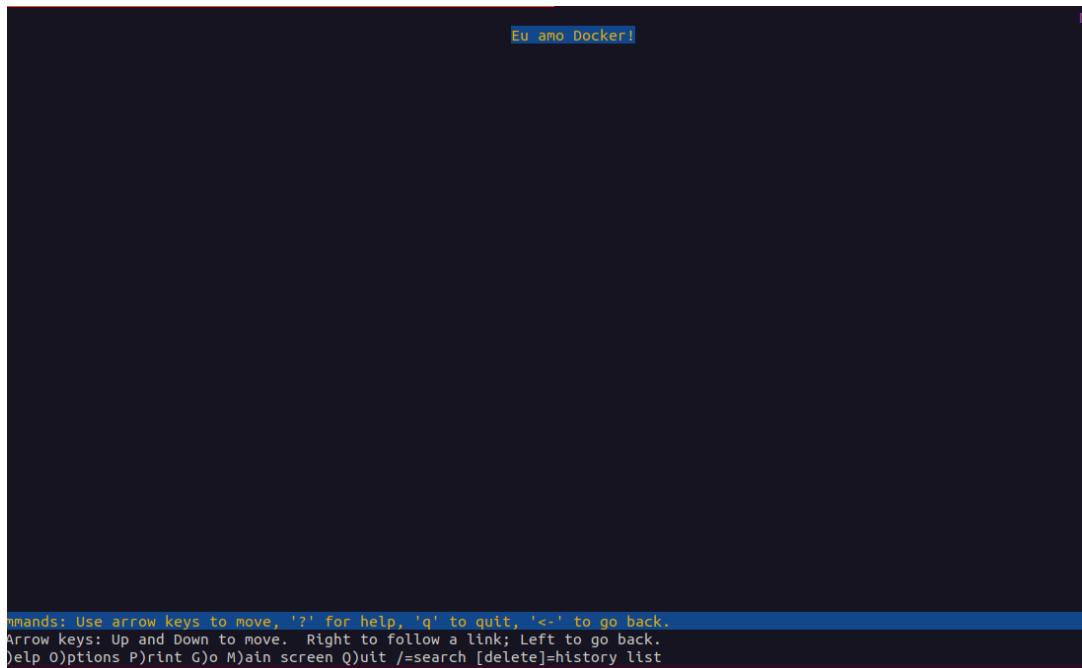
lk47@linux-labs:~/app-exemplo$ docker images
REPOSITORY          TAG           IMAGE ID        CREATED         SIZE
lk47-labs/app-node  1.0           ba9da2de3783   5 minutes ago  913MB
ubuntu              latest        3b418d7b466a   8 days ago    77.8MB
hello-world         latest        feb5d9fea6a5   19 months ago   13.3kB
dockersamples/static-site  latest        f589ccde7957   7 years ago    191MB
lk47@linux-labs:~/app-exemplo$ ^C
lk47@linux-labs:~/app-exemplo$ docker run -d -p 8081:3000 lk47-labs/app-node:1.0

```

- Validando agora diretamente o container no local host

```
lk47@linux-labs:~/app-exemplo$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
NAMES
9219af5819cb lk47-labs/app-node:1.0 "/bin/sh -c 'npm sta..." 48 seconds ago Up 46
000/tcp wonderful_bouman
lk47@linux-labs:~/app-exemplo$ docker port 9219af5819cb
3000/tcp -> 0.0.0.0:8081
3000/tcp -> [::]:8081
lk47@linux-labs:~/app-exemplo$ lynx 0.0.0.0:8081
```

- Vemos que a aplicação node está em funcionamento



▼ Instruções no Dockerfile

- A instrução `FROM` é usada para definirmos uma imagem como base para a nossa.
- Desta maneira, podemos adicionar à nossa imagem conteúdos que utilizaremos de maneira mais prática.

▼ Incrementando a imagem

- `docker stop $(docker container ls -q)` → para todos os containers
 - `-q` → flag para pegar o id de todos os containers em execução

```
lk47@linux-labs:~/app-exemplo$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
9219af5819cb lk47-labs/app-node:1.0 "/bin/sh -c 'npm sta..." 4 hours ago Up 4 hours 0.0.0.0:8081->3000/tcp, :::8081->3000/tcp wonderful_bouman
lk47@linux-labs:~/app-exemplo$ docker stop $(docker container ls -q)
9219af5819cb
lk47@linux-labs:~/app-exemplo$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
lk47@linux-labs:~/app-exemplo$
```

- Vamos rodar novamente a imagem que criamos, mas sem alterar a porta usando o comando `docker run -d lk47-labs/app-node:1.0`

```
lk47@linux-labs:~/app-exemplo$ docker run -d lk47-labs/app-node:1.0
3b991955a7a75c355c2cb5e1f081a8d89edcc7d51e6e88d6a1e5c7838e17859
lk47@linux-labs:~/app-exemplo$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
3b991955a7a7 lk47-labs/app-node:1.0 "/bin/sh -c 'npm sta..." 10 seconds ago Up 9 seconds
lk47@linux-labs:~/app-exemplo$
```

- Podemos ver que não é mostrado nada na coluna de portas, porém sabemos que a aplicação está definida no código para rodar na porta 3000
- Podemos incrementar a imagem para fazer com que o container exponha a porta que está sendo exposta na porta 3000

```
lk47@linux-labs:~/app-exemplo$ cat dockerfile
FROM node:14
EXPOSE 3000 ← Esta linha está apontando que a aplicação está exposta na porta 3000
WORKDIR /app-node
COPY . .
RUN npm install
ENTRYPOINT npm start
lk47@linux-labs:~/app-exemplo$
```

- Feito isso rodamos o `docker build -t lk47-labs/app-node:1.1` para criar uma nova imagem

```
Start a build
lk47@linux-labs:~/app-exemplo$ docker build -t lk47-labs/app-node:1.1 .
[+] Building 5.4s (9/9) FINISHED
--> [internal] load build definition from dockerfile
--> => transferring dockerfile: 126B
--> [internal] load .dockerrunignore
--> => transferring context: 2B
--> [internal] load metadata for docker.io/library/node:14
--> [1/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
--> [internal] load build context
--> => transferring context: 10.46kB
--> => CACHED [2/4] WORKDIR /app-node
--> => [3/4] COPY . .
--> => [4/4] RUN npm install
--> => exporting to image
--> => exporting layers
--> => writing image sha256:1c7d90ee804f51e20b9e4cbd9fc5140be0e7904544b3490cd9e712fd7658995a
--> => naming to docker.io/lk47-labs/app-node:1.1
lk47@linux-labs:~/app-exemplo$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
lk47-labs/app-node 1.1 1c7d90ee804f 6 seconds ago 913MB
lk47-labs/app-node 1.0 ba9da2de3783 4 hours ago 913MB
ubuntu latest 3b418d7b466a 9 days ago 77.8MB
hello-world latest febd59fread65 19 months ago 13.3kB
dockersamples/static-site latest f589ccde7957 7 years ago 191MB
lk47@linux-labs:~/app-exemplo$ docker run -d lk47-labs/app-node:1.1
a1987fb763cc73f29c11f07786873f09e249540fd35237afeef2871d708e4c0a9
lk47@linux-labs:~/app-exemplo$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a1987fb763cc lk47-labs/app-node:1.1 "/bin/sh -c 'npm sta..." 6 seconds ago Up 5 seconds 3000/tcp heuristic_chebyshev
3b991955a7a7 lk47-labs/app-node:1.0 "/bin/sh -c 'npm sta..." 9 minutes ago Up 9 minutes tender_khorana
lk47@linux-labs:~/app-exemplo$
```

- Podemos ver que a porta irá ficar visivel
- Porém podemos efetuar mais alterações, para ficar mais parametrizado na criação da imagem através de uma variavel de ambiente
- Primeiramente vamos alterar no arquivo index.js do node

```
const express = require('express')
let app = express();

app.use(express.static("."));

app.get("/", (req,res)=>{
    res.sendFile(__dirname + '/index.html') Aqui estamos declarando qual
} )                                     porta o app ficara exposto

app.listen("3000", ()=>{
    console.log("Server is listening on port 3000")
})
```

```
const express = require('express')
let app = express();

app.use(express.static("."));

app.get("/", (req,res)=>{
    res.sendFile(__dirname + '/index.html')
} )

app.listen(process.env.PORT, ()=>{ No caso vamos deixar dessa maneira
    console.log("Server is listening on port 3000") estamos definindo no app que o
} )                                     mesmo leia uma variavel de ambiente
                                            chamada PORT
```

- Agora vamos alterar o dockerfile, para receber esse parametrô para a imagem
 - Vamos usar um **ARG** para declarar a porta, porém isso só servirá para a imagem em sí
 - Para que seja alocado ao container usamos o **ENV**
 - Alteramos o valor do **EXPOSE** e **ENV** para o mesmo da variavel

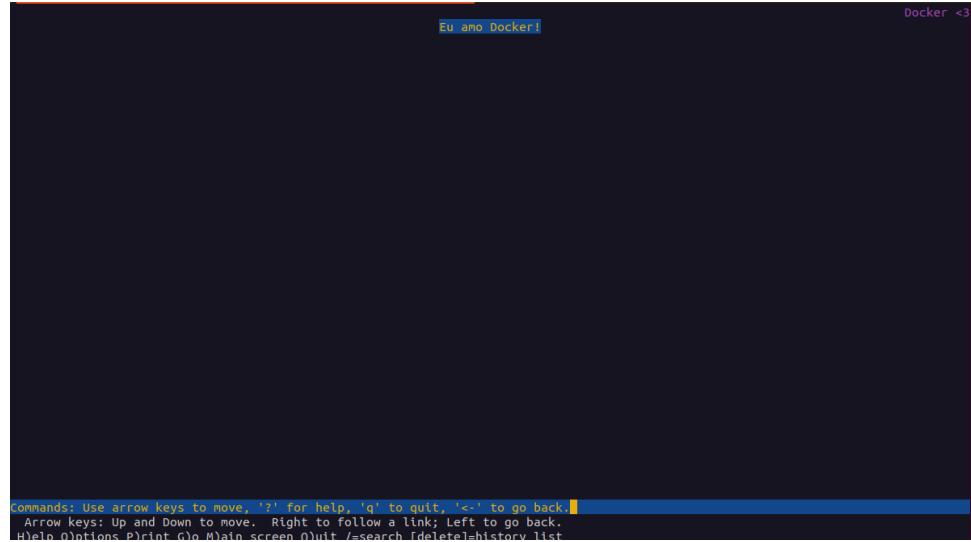
```
lk47@linux-labs:~/app-exemplo$ cat dockerfile
FROM node:14
WORKDIR /app-node
ARG PORT BUILD=6000
ENV PORT=$PORT BUILD
EXPOSE $PORT BUILD
COPY . .
RUN npm install
ENTRYPOINT npm start
lk47@linux-labs:~/app-exemplo$
```

- Vamos buildar novamente

```
lk47@linux-labs:~/app-exemplo$ docker build -t lk47-labs/app-node:1.2 .
[+] Building 6.2s (9/9) FINISHED
--> [internal] load build definition from dockerfile
--> => transferring dockerfile: 176B
--> [internal] load .dockerignore
--> => transferring context: 2B
--> [internal] load metadata for docker.io/library/node:14
--> [3/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
--> [internal] load build context
--> => transferring context: 16.78kB
--> => CACHED [2/4] WORKDIR /app-node
--> [3/4] COPY .
--> [4/4] RUN npm install
--> exporting to image
--> => exporting layers
--> => writing image sha256:a1a49b39ebf5c59e92a2f9185df072b9aa3b9dc361bf333b4ded021888b60db
--> => naming to docker.io/lk47-labs/app-node:1.2
lk47@linux-labs:~/app-exemplo$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
lk47-labs/app-node 1.2 1a149b39ebf 7 seconds ago 913MB
lk47-labs/app-node 1.1 1c7d90ee804f 23 minutes ago 913MB
lk47-labs/app-node 1.0 ba9da2de3783 5 hours ago 913MB
ubuntu latest 3b418d7b466a 9 days ago 77.8MB
hello-world latest febd59fe6a5 19 months ago 13.3KB
dockersamples/static-site latest f589ccde7957 7 years ago 191MB
lk47@linux-labs:~/app-exemplo$ docker run -d -n
lk47@linux-labs:~/app-exemplo$ docker run -d lk47-labs/app-node:1.2
556bf2c6869d0cf4dfaafdf3932ee4b6beec5b86b890c2663788d73a07d086a3
lk47@linux-labs:~/app-exemplo$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
556bf2c6869d lk47-labs/app-node:1.2 "/bin/sh -c 'npm sta..." 3 seconds ago Up 2 seconds 6000/tcp blissful_stonebraker
a1987fb763ct lk47-labs/app-node:1.1 "/bin/sh -c 'npm sta..." 24 minutes ago Up 24 minutes 3000/tcp heuristic_chebyshev
3b991955a7a7 lk47-labs/app-node:1.0 "/bin/sh -c 'npm sta..." 33 minutes ago Up 33 minutes tender_khorana
lk47@linux-labs:~/app-exemplo$
```

- E podemos ver a porta exposta conforme configurada
- Se subirmos novamente um container expondo em uma das portas da nossa maquina (usando o `-p`)

```
lk47@linux-labs:~/app-exemplo$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
281c901f5c4c lk47-labs/app-node:1.2 "/bin/sh -c 'npm sta..." 12 seconds ago Up 11 seconds 0.0.0.0:9090->6000/tcp, :::9090
6000/tcp amazing_knuth
556bf2c6869d lk47-labs/app-node:1.2 "/bin/sh -c 'npm sta..." 2 minutes ago Up 2 minutes 6000/tcp
a1987fb763ct lk47-labs/app-node:1.1 "/bin/sh -c 'npm sta..." 27 minutes ago Up 27 minutes 3000/tcp
heuristic_chebyshev
3b991955a7a7 lk47-labs/app-node:1.0 "/bin/sh -c 'npm sta..." 36 minutes ago Up 36 minutes
tender_khorana
lk47@linux-labs:~/app-exemplo$ docker port 281c901f5c4c
6000/tcp -> 0.0.0.0:9090
6000/tcp -> [::]:9090
lk47@linux-labs:~/app-exemplo$ lynx 0.0.0.0:9090
```



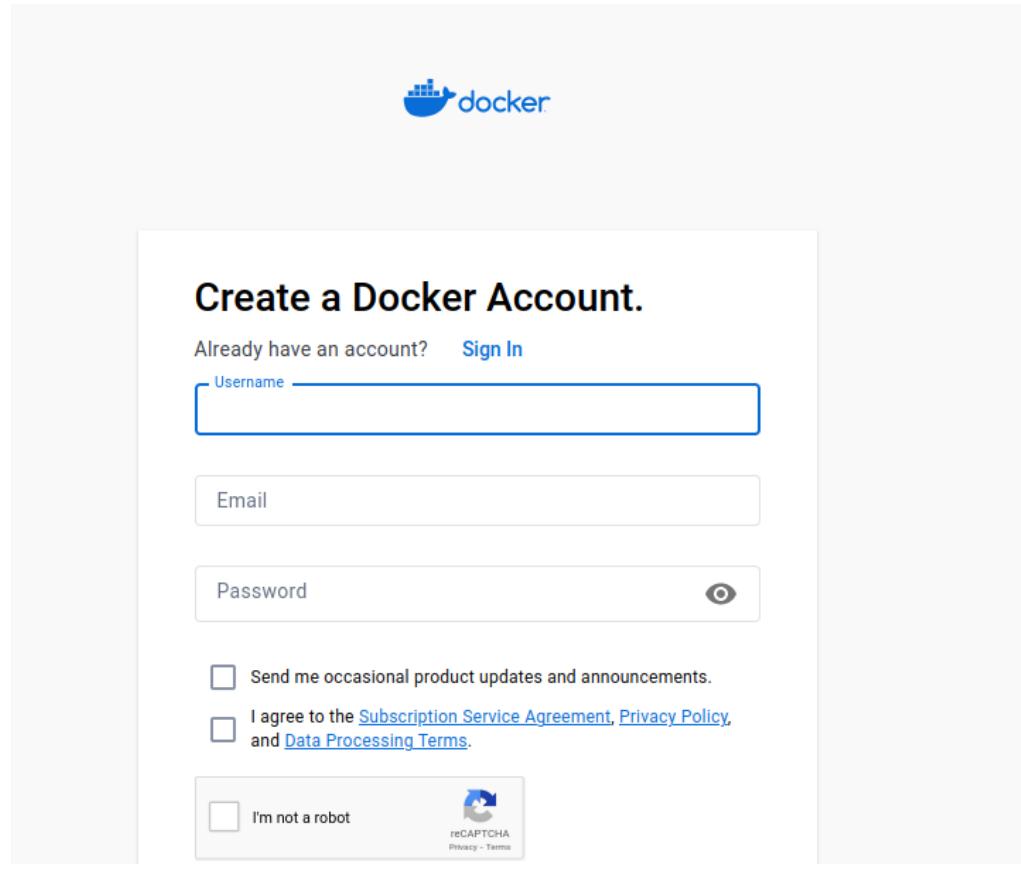
- Teremos o acesso da aplicação

▼ ARG vs ENV

- A instrução `ARG` carrega variáveis apenas no momento de build da imagem, enquanto a instrução `ENV` carrega variáveis que serão utilizadas no container.
 - Desta maneira, é possível diferenciar o que é necessário para a etapa de build e para a etapa de execução.

▼ Subindo a imagem para o Docker Hub

- Antes de tudo é necessário criar uma conta no docker hub



Docker Hub

 <https://hub.docker.com/signup>

- Feito isso seguimos no terminal
 - Usamos o comando `docker login -u <username> -p <senha>`
- ```
lk47@linux-labs:~$ docker login -u luc4sbr1to -p [REDACTED]
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /home/lk47/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
lk47@linux-labs:~$
```
- Ou podemos usar só o parametro `-u` e a senha é solicitada em seguida porém não ficará a mostra
  - Após conectarmos, precisaremos alterar o nome das imagens pois as mesmas devem ficar com o nome que está em nosso repositorio

<nome\_repositorio>/<nome\_da\_imagem>

- Usamos o docker tag <nome da imagem> <novo nome> → Faz uma cópia da imagem com o novo nome

```
lk47@linux-labs:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
lk47-labs/app-node 1.2 1a149b39e0bf 20 minutes ago 913MB
lk47-labs/app-node 1.1 1c7d90ee804f 44 minutes ago 913MB
lk47-labs/app-node 1.0 ba9da2de3783 5 hours ago 913MB
ubuntu latest 3b418d7b466a 9 days ago 77.8MB
hello-world latest feb5d9fea6a5 19 months ago 13.3kB
dockersamples/static-site latest f589ccde7957 7 years ago 191MB
lk47@linux-labs:~$ docker tag lk47-labs/app-node:1.0 luc4sbr1to/app-node:1.0
lk47@linux-labs:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
lk47-labs/app-node 1.2 1a149b39e0bf 21 minutes ago 913MB
lk47-labs/app-node 1.1 1c7d90ee804f 44 minutes ago 913MB
lk47-labs/app-node 1.0 ba9da2de3783 5 hours ago 913MB
luc4sbr1to/app-node 1.0 ba9da2de3783 5 hours ago 913MB
ubuntu latest 3b418d7b466a 9 days ago 77.8MB
hello-world latest feb5d9fea6a5 19 months ago 13.3kB
dockersamples/static-site latest f589ccde7957 7 years ago 191MB
lk47@linux-labs:~$
```

- Agora para subir as imagens usamos o `docker push`

```
lk47@linux-labs:~$ docker push luc4sbr1to/app-node:1.0
The push refers to repository [docker.io/luc4sbr1to/app-node]
adf195beeabc: Pushed
bb04cd8f3184: Pushed
6cf0c89e5007: Pushed
0d5f5a015e5d: Mounted from library/node
3c777d951de2: Mounted from library/node
f8a91dd5fc84: Mounted from library/node
cb81227abde5: Mounted from library/node
e01a454893a9: Mounted from library/node
c45660adde37: Mounted from library/node
fe0fb3ab4a0f: Mounted from library/node
f1186e5061f2: Mounted from library/node
b2dba7477754: Mounted from library/node
1.0: digest: sha256:f80c6ae066eb0ef2bef85b28c9263c88be1eb2f511c9f6e66add560c535b00bd size: 2839
lk47@linux-labs:~$ docker push luc4sbr1to/app-node:1.1
The push refers to repository [docker.io/luc4sbr1to/app-node]
b32287939b92: Pushed
0d255f40189a: Pushed
6cf0c89e5007: Layer already exists
0d5f5a015e5d: Layer already exists
3c777d951de2: Layer already exists
f8a91dd5fc84: Layer already exists
cb81227abde5: Layer already exists
e01a454893a9: Layer already exists
c45660adde37: Layer already exists
fe0fb3ab4a0f: Layer already exists
f1186e5061f2: Layer already exists
b2dba7477754: Layer already exists
1.1: digest: sha256:b285e0083c5b4230950b7ed967e5ca263a6be3d896b391e32b16728f0bdf5f51 size: 2839
lk47@linux-labs:~$
```

Com este comando estamos subindo a imagem para o nosso dockerhub

Subindo outras versões da imagem, podemos ver que as camadas já existentes não irão subir, facilitando subir versões diferentes

**Docker commands**

```
docker push luc4sbr1to/app-node:tagname
```

**Tags**

| Tag | OS     | Type  | Pulled        | Pushed            |
|-----|--------|-------|---------------|-------------------|
| 1.2 | Ubuntu | Image | ---           | a few seconds ago |
| 1.1 | Ubuntu | Image | 6 minutes ago | 10 minutes ago    |
| 1.0 | Ubuntu | Image | 6 minutes ago | 10 minutes ago    |

[See all](#) [Go to Advanced Image Management](#)

**Automated Builds**

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

[Upgrade](#)

- **Imagens são imutáveis, ou seja, depois de baixadas, múltiplos containers conseguirão reutilizar a mesma imagem;**
- **Imagens são compostas por uma ou mais camadas. Dessa forma, diferentes imagens são capazes de reutilizar uma ou mais camadas em comum entre si;**
- **Podemos criar nossas imagens através de Dockerfiles e do comando `docker build`;**
- **Para subir uma imagem no Docker Hub, utilizamos o comando `docker push`.**

## ▼ Persistindo dados

### ▼ O problema de persistir dados

- Vamos remover todos os containers e todas as imagens
  - `docker rm $(docker container ls -q)`
    - Caso não haja container em execução podemos usar a flag `-a`

```
lk47@linux-labs:~$ docker rm $(docker container ls -a -q)
281c901f5c4c
556bf2c6869d
a1987fb763cc
3b991955a7a7
9219af5819cb
98c82e785283
5c60eaaedf21
f251923c1004
lk47@linux-labs:~$
```

- o `docker rmi $(docker image ls -aq)`

- Caso fale de dependências utilizar a flag `--force`

```
lk47@linux-labs:~$ docker rmi $(docker image ls -aq)
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:0b8a884ad7ed20e50a68108895a6e7db54b303ad73b22b6b9ff5f60740dc03e2
Deleted: sha256:3b418d7b466ac6275a6abfc86febe4422ff6eaafaf444a294f82d3bf5f173ce74
Deleted: sha256:b8a36d10656ac19dd9b6ef3107f76820663717708fc37ce299925c36d1b1d157
Untagged: dockersamples/static-site:latest
Untagged: dockersamples/static-site@sha255c...daa686c61d7d239ff77e72157997489db49f316b9b9af3909d9f10fd28b2dec
Deleted: sha256:f589ccde7957fa3ddf76a2eeee4df2f5d687b32176f559b703b6b8cacfd36bc4
Deleted: sha256:068bb05c1f415c0f505428477910b6a672c1439421e3bd958db7d0a8b3dfb671
Deleted: sha256:42d2184b31e555191b329ab06997f82fe8c51991f19017e990d6739848cf
Deleted: sha256:b0dfb0190c842cadda115b029h4157cf98595f4a090fc2a1c4-5if5f5f749e0ca6
Deleted: sha256:1a31d9de2c5959dfc9ced01e0575b119a125c7206a77b68c0dfbeec35629f1
Deleted: sha256:5ace35615df0450258a03e074a0f7996d8aa30d5eae7cea779d08e4be4
Deleted: sha256:02f990df226000c138600f059b06672a0c42ce1dd5769c294f443368c1de64
Deleted: sha256:894fdee79cd8b48c0b0bac6664ad032c7672c80a1d81ac2d0a5505d0e116
Deleted: sha256:917c0fc99b35c0397a9effd042d72529de7c452609b1e1b05bec3088c7056
Error response from daemon: conflict: unable to delete 1a149b39e0bf (must be forced) - image is referenced in multiple repositories
Error response from daemon: conflict: unable to delete 1a149b39e0bf (must be forced) - image is referenced in multiple repositories
Error response from daemon: conflict: unable to delete 1c7d90ee804f (must be forced) - image is referenced in multiple repositories
Error response from daemon: conflict: unable to delete 1c7d90ee804f (must be forced) - image is referenced in multiple repositories
Error response from daemon: conflict: unable to delete ba9da2de3783 (must be forced) - image is referenced in multiple repositories
Error response from daemon: conflict: unable to delete ba9da2de3783 (must be forced) - image is referenced in multiple repositories
lk47@linux-labs:~$ docker rmi $(docker image ls -aq) --force
Untagged: lk47-labs/app-node:1.2
Untagged: luc4sbrito/app-node:1.2
Untagged: luc4sbrito/app-node@sha256:e26639e6dd6b7798a2226f16388121a230b951debd8e455202ce34e14efcff6
Deleted: sha256:1a149b39e0bf5c9e92a2f9185df072b9aaaf3b9dc361bf333b4ded021888b60db
Untagged: lk47-labs/app-node:1.1
Untagged: luc4sbrito/app-node:1.1
Untagged: luc4sbrito/app-node@sha256:b285e0083c5b4230950b7ed967e5ca263a6be3d896b391e32b16728f0bdf5f51
Deleted: sha256:1c7d90ee804f51e20b9e4cbdcfc5140be0e7904544b3490cd9e712f7658995a
Untagged: lk47-labs/app-node:1.0
Untagged: luc4sbrito/app-node:1.0
Untagged: luc4sbrito/app-node@sha256:f80c6ae66e0b6f2bef85b28c9263c88be1eb2f511c9f6e66add560c535b00bd
Deleted: sha256:b9ad2de3783226ed82091b1c94838f8ba73d57b6f7474470d7222c2a0ba40
Error response from daemon: No such image: 1a149b39e0bf:latest
Error response from daemon: No such image: 1c7d90ee804f:latest
Error response from daemon: No such image: ba9da2de3783:latest
```

```
lk47@linux-labs:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
lk47@linux-labs:~$
```

- Vamos subir um ubuntu no modo interativo

- o `docker run -it ubuntu bash`

- Rodando o `docker ps -s` → Podemos ver o tamanho do container e o tamanho virtual

```
lk47@linux-labs:~$ docker ps -s
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES SIZE
f4d223a1adb8 ubuntu "bash" 14 seconds ago Up 12 seconds eager_feistel 0B (virtual 77.8MB)
lk47@linux-labs:~$
```

- Se rodarmos o `docker history` podemos ver que o tamanho da imagem que seria o tamanho virtual

```
lk47@linux-labs:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ubuntu latest 3b418d7b466a 10 days ago 77.8MB
lk47@linux-labs:~$ docker history 3b418d7b466a
IMAGE CREATED CREATED BY SIZE COMMENT
3b418d7b466a 10 days ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0B
<missing> 10 days ago /bin/sh -c #(nop) ADD file:2fc6364d149ecc7f... 77.8MB
<missing> 10 days ago /bin/sh -c #(nop) LABEL org.opencontainers...
<missing> 10 days ago /bin/sh -c #(nop) LABEL org.opencontainers...
<missing> 10 days ago /bin/sh -c #(nop) ARG LAUNCHPAD_BUILD_ARCH 0B
<missing> 10 days ago /bin/sh -c #(nop) ARG RELEASE 0B
lk47@linux-labs:~$
```

container

SIZE

- Dentro do container vamos rodar o `apt-get update`

```
root@f4d223a1ad0b:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1010 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [1083 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [925 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [41.2 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1356 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [1137 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1166 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [46.6 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [25.6 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [49.4 kB]
Fetched 27.1 MB in 6s (4542 kB/s)
Reading package lists... Done
root@f4d223a1ad0b:/#
```

- Se validarmos o tamanho novamente podemos ver que ocorreu alguns registros

```
lk47@linux-labs:~$ docker ps -s
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES SIZE
f4d223a1ad0b ubuntu "bash" 2 minutes ago Up 2 minutes eager_feistel 43.8MB (virtual 122MB)
lk47@linux-labs:~$ docker history 3b418d7b466a
IMAGE CREATED CREATED BY SIZE COMMENT
3b418d7b466a 10 days ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0B
<missing> 10 days ago /bin/sh -c #(nop) ADD file:2fc6364d149ecc7f... 77.8MB
<missing> 10 days ago /bin/sh -c #(nop) LABEL org.opencontainers...
<missing> 10 days ago /bin/sh -c #(nop) LABEL org.opencontainers...
<missing> 10 days ago /bin/sh -c #(nop) ARG LAUNCHPAD_BUILD_ARCH 0B
<missing> 10 days ago /bin/sh -c #(nop) ARG RELEASE 0B
lk47@linux-labs:~$
```

SIZE

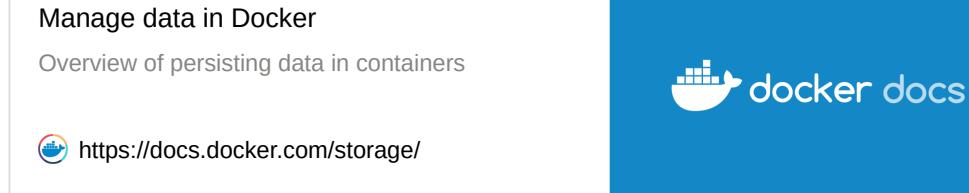
- No caso é a img  $77.8 + 43.8 = 121.6$  porém é arredondado para 122 (pois o decimal é maior que 5)

- Portanto podemos entender que a camada R/W foi modificada (que no caso é a camada do container)
- Se recriarmos o container, vemos que a camada zerou, portanto podemos ver que a camada R/W foi zerada

```
lk47@linux-labs:~$ docker run -it ubuntu bash
root@ffa1893a9d13:/#
```

| CONTAINER ID | IMAGE  | COMMAND | CREATED        | STATUS        | PORTS | NAMES                | SIZE                |
|--------------|--------|---------|----------------|---------------|-------|----------------------|---------------------|
| ffa1893a9d13 | ubuntu | "bash"  | 12 seconds ago | Up 11 seconds |       | descending_heyrovsky | 0B (virtual 77.8MB) |

- Para persistir os dados do container podemos usar os volumes que são dos seguintes tipos:
  - bind mount → Entre o file system do nosso SO e o container fazendo com que persista a informação no host
  - Volume → que é gerenciado pelo docker
  - tmpfs → que seria temporário



## ▼ Tipos de persistência

- **Volumes**
  - Com volumes, é possível escrever os dados em uma camada persistente.
- **Bind mounts**
  - Com bind mounts, é possível escrever os dados em uma camada persistente baseado na estrutura de pastas do host.

## ▼ Utilizando bind mounts

- Ligação de um ponto de montagem com nosso SO e um diretório no container
- Em nosso SO vamos criar uma pasta chamada `volume-docker`

```
lk47@linux-labs:~$ mkdir volume-docker
lk47@linux-labs:~$ ls
app-exemplo app-exemplo.zip volume-docker
```

- `docker run -it -v /home/<nome do usuário>/volume-docker:/app ubuntu bash`

```
lk47@linux-labs:~$ docker run -it -v /home/lk47/volume-docker:/app ubuntu bash
root@d28a6662810db:/#
```

- Com este comando com a flag `-v` estou especificando que os dados salvo no diretório `/app` no container sejam persistidos no diretório `volume-docker` em nosso SO
- Dentro do container podemos ver o diretório `'app'`
  - Vamos salvar um arquivo dentro dessa pasta

```
root@ffac51cbde57:/# ls
app bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@ffac51cbde57:/# cd app/
root@ffac51cbde57:/app# touch arquivo-persistente
root@ffac51cbde57:/app# ls
arquivo-persistente
root@ffac51cbde57:/app#
```

- Se checarmos a pasta `volume-docker` em nosso SO estará lá

```
lk47@linux-labs:~$ cd volume-docker/
lk47@linux-labs:~/volume-docker$ ls
arquivo-persistente
```

- Vamos criar novamente o container com o mesmo caminho, se entramos na pasta `'app'` podemos ver o arquivo ainda lá

```
lk47@linux-labs:~/volume-docker$ docker run -it -v /home/lk47/volume-docker:/app ubuntu bash
root@d0825d087fd0:/# cd app/
root@d0825d087fd0:/app# ls
arquivo-persistente
root@d0825d087fd0:/app#
```

- Outra forma de criarmos volume do tipo bind mount é utilizar o comando

- `docker run -it --mount type=bind,source=/home/<nome do usuario>/volume-docker,target=/app ubuntu bash`

```
lk47@linux-labs:~/volume-docker$ docker run -it --mount type=bind,source=/home/lk47/volume-docker,target=/app ubuntu bash
root@bcea4979f37a:/# ls
app bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@bcea4979f37a:/# cd app/
root@bcea4979f37a:/app# ls
arquivo-persistente
root@bcea4979f37a:/app#
```

- `--mount type=` → especifica o tipo de volume + `source=` especifica o caminho no host + `target=` especifica o diretório no container

### ▼ Criando um bind

- `docker run --mount type=bind,source=/home/diretorio,target=/app nginx`
  - *Com esta sintaxe, criaremos um bind mount para o container baseado na imagem do nginx.*

### ▼ Utilizando volumes

- Por mais que sejam persistidas no host, o próprio docker fará o gerenciamento
- `docker volume ls` → lista os volumes

```
lk47@linux-labs:~$ docker volume ls
DRIVER VOLUME NAME
lk47@linux-labs:~$
```

- `docker volume create <nome do volume>` → cria um volume

```
lk47@linux-labs:~$ docker volume create volume-docker
volume-docker
lk47@linux-labs:~$ docker volume ls
DRIVER VOLUME NAME
local volume-docker
lk47@linux-labs:~$
```

- Agora iremos criar um container novamente
  - `docker run -it -v <nome do volume>:/app ubuntu bash` → onde a flag `-v` direciona o volume e a pasta do container onde os dados serão persistidos

- Vamos criar um arquivo no container
- Recriar o container e o arquivo permanecerá lá

```
lk47@linux-labs: $ docker run -it -v volume-docker:/app ubuntu bash
root@93bf1d998bac:/# ls
app bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@93bf1d998bac:/# cd app/
root@93bf1d998bac:/app# touch arquivo-persistente
root@93bf1d998bac:/app# ls
arquivo-persistente
root@93bf1d998bac:/app# exit
exit
lk47@linux-labs: $ docker run -it -v volume-docker:/app ubuntu bash
root@b2855e892da1:/# cd app/
root@b2855e892da1:/app# ls
arquivo-persistente
root@b2855e892da1:/app#
```

- Os dados são gravados na pasta `/var/lib/docker`
  - Lá há um diretório chamado `volumes` que é onde fica os volumes criados pelo docker
  - Indo em `_data` podemos ver os arquivos do container

```
lk47@linux-labs:~$ sudo su - ← Necessario entrar em modo user root
root@linux-labs:~# cd /var/lib/docker
root@linux-labs:/var/lib/docker# ls
buildkit containers engine-id image network overlay2 plugins runtimes swarm tmp volumes
root@linux-labs:/var/lib/docker# cd volumes/
root@linux-labs:/var/lib/docker/volumes# ls
backingFsBlockDev metadata.db volume-docker
root@linux-labs:/var/lib/docker/volumes# cd volume-docker/_data/
root@linux-labs:/var/lib/docker/volumes/volume-docker/_data# ls
arquivo-persistente
root@linux-labs:/var/lib/docker/volumes/volume-docker/_data#
```

- Assim conseguimos gerenciar os volumes pelo próprio docker ao invés de ser pelo root
- Podemos também usar o comando
- `docker run -it --mount source=<nome do volume>, target=/app ubuntu bash`
  - Esta é a outra forma de executar o container com volume, sem precisar utilizar o `type`

```
lk47@linux-labs:~$ docker run -it --mount source=volume-docker,target=/app ubuntu bash
root@7ccad5e4fd36:/# ls
app bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@7ccad5e4fd36:/# cd app/
root@7ccad5e4fd36:/app# ls
arquivo-persistente
root@7ccad5e4fd36:/app#
```

- Podemos criar um volume automaticamente também
  - `docker run -it --mount source=<nome do novo volume>, target=/app ubuntu bash`

```
lk47@linux-labs:~$ docker run -it --mount source=volume-docker2,target=/app ubuntu bash
root@f7684ce2316a:/# ls
app bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@f7684ce2316a:/# exit
exit
lk47@linux-labs:~$ docker volume ls
DRIVER VOLUME NAME
local volume-docker
local volume-docker2
```

## ▼ Vantagens de volumes

- **Volumes são gerenciados pelo Docker e independem da estrutura de pastas do sistema.**
  - Desta maneira, a persistência de dados independe de como as pastas do sistema estão estruturadas.

## ▼ Utilizando tmpfs

- No caso o tmpfs utiliza uma flag própria `--tmpfs=<pasta de persistencia de volumes do container>`
  - `docker run -it --tmpfs=/app ubuntu bash`

```
lk47@linux-labs:~$ docker run -it --tmpfs=/app ubuntu bash
root@3b49d4f9a3d5:/# ls
app bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@3b49d4f9a3d5:/#
```

- Quando rodamos este comando a pasta será temporária
- Se criarmos um arquivo novamente

```
root@3b49d4f9a3d5:/# cd app/
root@3b49d4f9a3d5:/app# ls
arquivo-persistente
root@3b49d4f9a3d5:/app#
```

- E recriarmos novamente o container os dados não irão se persistir

```
lk47@linux-labs:~$ docker run -it --tmpfs=/app ubuntu bash
root@adf70dbd3190:/# cd app/
root@adf70dbd3190:/app# ls
root@adf70dbd3190:/app#
```

- Pois o tmpfs serve apenas para armazenamento de dados temporários especificando a pasta de destino, serve para guardar informações sensíveis

- Outra forma de executar este comando é

- `docker run -it --mount type=tmpfs,destination=/app ubuntu bash`

```
lk47@linux-labs: $ docker run -it --mount type=tmpfs,destination=/app ubuntu bash
root@932d37db4b7c:/# ls
app bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@932d37db4b7c:/#
```

- Quando containers são removidos, nossos dados são perdidos;
- Podemos persistir dados em definitivo através de volumes e bind mounts;
- Bind mounts dependem da estrutura de pastas do host;
- Volumes são gerenciados pelo Docker;
- Tmpfs armazenam dados em memória volátil.

## ▼ Comunicação através de redes

**Para fins didáticos neste modulo utilizei o docker no WSL v2 no windows**

### ▼ Conhecendo a rede bridge

- Primeiramente vamos subir dois container usando o `docker run -it ubuntu bash`

```
lucasvertare@DESKTOP-2AEKE0L:~$ docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally Container1
latest: Pulling from library/ubuntu
dbf6a9befcde: Pull complete
Digest: sha256:df64a3b4296d8c9b62aa3309984f8620b98d87e47492599ee20739e8eb54fbf
Status: Downloaded newer image for ubuntu:latest
root@5da0d5d7d41a:/# |
```

```
lucasvertare@DESKTOP-2AEKE0L:~$ docker run -it ubuntu bash Container2
root@5debd1616094:/#
```

- Vamos utilizar outro terminal para validar outros pontos
- Podemos ver que há dois containers em processo `docker ps`

| CONTAINER ID | IMAGE  | COMMAND | CREATED            | STATUS            | PORTS | NAMES              |
|--------------|--------|---------|--------------------|-------------------|-------|--------------------|
| 5debd1616094 | ubuntu | "bash"  | About a minute ago | Up About a minute |       | descending_leavitt |
| 5da0d5d7d41a | ubuntu | "bash"  | About a minute ago | Up About a minute |       | intelligent_buck   |

- Se utilizarmos o comando `docker inspect + id do container` podemos ver vários detalhes do container em sí em formato JSON

```
lucasvertare@DESKTOP-2AEKEBL:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
5debd1616094 ubuntu "bash" About a minute ago Up About a minute
5da0d5d7d41a ubuntu "bash" About a minute ago Up About a minute
lucasvertare@DESKTOP-2AEKEBL:~$ docker inspect 5debd1616094
[{"Id": "5debd1616094bc47fabc762b5f47a7fdf44409dd3885a078cf6752fed7609540",
 "Created": "2023-05-09T18:28:19.069252173Z",
 "Path": "bash",
 "Args": [],
 "State": {
 "Status": "running",
 "Running": true,
 "Paused": false,
 "Restarting": false,
 "OOMKilled": false,
 "Dead": false,
 "Pid": 31457,
 "ExitCode": 0,
 "Error": "",
 "StartedAt": "2023-05-09T18:28:19.499407291Z",
 "FinishedAt": "0001-01-01T00:00:00Z"
 },
 "Image": "sha256:3b418d7b466ac6275a6bfcb0c86fbe4422ff6ea0af444a294f82d3bf5173ce74",
 "ResolvConfPath": "/var/lib/docker/containers/5debd1616094bc47fabc762b5f47a7fdf44409dd3885a078cf6752fed7609540/resolv.conf",
 "HostnamePath": "/var/lib/docker/containers/5debd1616094bc47fabc762b5f47a7fdf44409dd3885a078cf6752fed7609540/hostname",
 "HostsPath": "/var/lib/docker/containers/5debd1616094bc47fabc762b5f47a7fdf44409dd3885a078cf6752fed7609540/hosts",
 "LogPath": "/var/lib/docker/containers/5debd1616094bc47fabc762b5f47a7fdf44409dd3885a078cf6752fed7609540/logs",
 "Name": "/condescending_leavitt",
 "RestartCount": 0,
 "Driver": "overlay2",
 "Platform": "linux",
 "MountLabel": "",
 "ProcessLabel": "",
 "AppArmorProfile": "",
 "ExecIDs": null}
```

- Em network temos um conjunto chamado bridge que possui diversas configurações

```

 },
 "NetworkSettings": {
 "Bridge": "",
 "SandboxID": "f86c80f26f0a7b855a62865bde2948bb27b096a6bde314f3632ffcbef151381c",
 "HairpinMode": false,
 "LinkLocalIPv6Address": "",
 "LinkLocalIPv6PrefixLen": 0,
 "Ports": {},
 "SandboxKey": "/var/run/docker/netns/f86c80f26f0a",
 "SecondaryIPAddresses": null,
 "SecondaryIPv6Addresses": null,
 "EndpointID": "2e83d2233a439e921da23715bf7744aac6520b46c13c06cdcd57e2b5da71d9c9",
 "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": "ab6c68523a7b6cf826204aaa8a48a0ba2dc7fb74849dc9b1deb9a2b14905e1b4",
 "EndpointID": "2e83d2233a439e921da23715bf7744aac6520b46c13c06cdcd57e2b5da71d9c9",
 "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
 }
 }
 }
]
}

```

Container2

- Comparando o inspect network dos dois container podemos validar que ambos se encontram em uma mesma rede em comum

```

 "NetworkSettings": {
 "Bridge": "",
 "SandboxID": "b9da29b51ac47e35ccc5c7dee317c133c52ea56ea3aa94f9bb787ab00426dcf2",
 "HairpinMode": false,
 "LinkLocalIPv6Address": "",
 "LinkLocalIPv6PrefixLen": 0,
 "Ports": {},
 "SandboxKey": "/var/run/docker/netns/b9da29b51ac4",
 "SecondaryIPAddresses": null,
 "SecondaryIPv6Addresses": null,
 "EndpointID": "8cf2d29d02ba25541a59f3e1e9171bd1e707b70bcd746e22319c70f4f9c6ab483",
 "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": "ab6c68523a7b6cf826204aaa8a48a0ba2dc7fb74849dc9b1deb9a2b14905e1b4",
 "EndpointID": "8cf2d29d02ba25541a59f3e1e9171bd1e707b70bcd746e22319c70f4f9c6ab483",
 "Gateway": "172.17.0.1",
 "IPAddress": "172.17.0.2", 172.17.0.2 ←
 "IPPrefixLen": 16,
 "IPv6Gateway": "",
 "GlobalIPv6Address": "",
 "GlobalIPv6PrefixLen": 0,
 "MacAddress": "02:42:ac:11:00:02",
 "DriverOpts": null
 }
 }
 }
}

```

Container1

- No caso o próprio docker configura dessa forma ao subir um container
- Portanto se dermos um ping no de cada container, veremos que ambos irão se comunicar normalmente
  - No caso precisamos instalar o ping → `apt-get update` e `apt-get install iputils-ping`

```

root@5da0d5d7d41a:/# apt-get update && apt-get install iputils-ping Container1
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [904 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [597 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [39.6 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [869 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [893 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [907 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [44.9 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1139 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [49.4 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [25.6 kB]
Fetched 25.8 MB in 32s (806 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
 libcap2-bin libpam-cap
The following NEW packages will be installed:
 iputils-ping libcap2-bin libpam-cap
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 76.8 kB of archives.
After this operation, 280 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 libcap2-bin amd64 1:2.44-1build3 [26.0 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 iputils-ping amd64 3:20211215-1 [42.9 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/main amd64 libpam-cap amd64 1:2.44-1build3 [7932 B]
Fetched 76.8 kB in 2s (49.8 kB/s)
debcfg: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libcap2-bin.
(Reading database ... 4395 files and directories currently installed.)

```

```

root@5da0d5d7d41a:/# ping 172.17.0.3 Container1
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data. Pingando o Container2
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.470 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.090 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.074 ms
64 bytes from 172.17.0.3: icmp_seq=5 ttl=64 time=0.067 ms
64 bytes from 172.17.0.3: icmp_seq=6 ttl=64 time=0.073 ms
64 bytes from 172.17.0.3: icmp_seq=7 ttl=64 time=0.063 ms
^C
--- 172.17.0.3 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6211ms
rtt min/avg/max/mdev = 0.063/0.130/0.470/0.138 ms
root@5da0d5d7d41a:#

```

- Se utilizarmos o comando `docker network ls` podemos ver as redes existentes em nosso docker

| NETWORK ID   | NAME   | DRIVER | SCOPE |
|--------------|--------|--------|-------|
| ab6c68523a7b | bridge | bridge | local |
| ca29d135d9ea | host   | host   | local |
| 87237b1b5390 | none   | null   | local |

- No caso nossos dois container foram colocados nessa rede padrão chamada de bridge que utiliza o driver bridge

## ▼ Redes do Docker

- **Redes padrão já criadas pelo Docker.**
  - **HOST**
  - **NONE**
  - **BRIDGE**

## ▼ Criando uma rede bridge

- No caso o docker já cria 3 redes padrões, Host, None e bridge, cada uma com seus respectivos drivers
- No caso vamos criar uma rede do tipo bridge, que seria aquela responsável pela comunicação entre os containers
- Vamos usar o comando `docker network create --driver bridge minha-bridge`
  - Rodando um `docker network ls` já podemos ver a rede que acabamos de criar

```
lucasvertare@DESKTOP-2AEKE0L:~$ docker network create --driver bridge minha-bridge
9c8de904a6a13307521ffe3986333c1b94187f320a59f2bd3ea0157e52023b2c
lucasvertare@DESKTOP-2AEKE0L:~$ docker network ls
NETWORK ID NAME DRIVER SCOPE
ab6c68523a7b bridge bridge local
ca29d135d9ea host host local
9c8de904a6a1 minha-bridge bridge local
87237b1b5390 none null local
lucasvertare@DESKTOP-2AEKE0L:~$ |
```

- Vamos apagar todos os containers existentes utilizando o comando `docker container rm $(docker ps -aq) --force`

```
lucasvertare@DESKTOP-2AEKE0L:~$ docker container rm $(docker ps -aq) --force
5debd1616094
5da0d5d7d41a
23dc96c07043
```

- Após removermos vamos criar um container , mas dessa vez especificando o nome dos containers com a flag `--name` e a rede na qual criamos `--network` , em name colocaremos “pong”
  - `docker run -d --name pong --network minha-bridge ubuntu sleep 1d`

- Este container vai rodar em segundo plano com sleep de 1d para manter em execução

```
lucasvertare@DESKTOP-2AEKE0L:$ docker run -d --name pong --network minha-bridge ubuntu sleep 1d
51c84dc5c954934dbc48cf6656d48282601d0f3dee9cab9577aa51d2f0ab9b74
lucasvertare@DESKTOP-2AEKE0L:$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
51c84dc5c954 ubuntu "sleep 1d" 3 seconds ago Up 2 seconds pong
de5cc2db9d4e ubuntu "bash" 2 minutes ago Up 2 minutes ubuntu1
```

- Em seguida vamos criar um outro no acessaremos através do -it

- `docker run -it --name ubuntu1 --network minha-bridge ubuntu bash`
- E vamos instalar o ping → `apt-get update` e `apt-get install iputils-ping -y`

```
lucasvertare@DESKTOP-2AEKE0L:$ docker run -it --name ubuntu1 --network minha-bridge ubuntu bash
root@de5cc2db9d4e:/# apt-get update && apt-get install iputils-ping -y
Get:1 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [906 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [108 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [40.4 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [398 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [599 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1142 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [45.7 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [891 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [403 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [49.4 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [25.6 kB]
Fetched 24.8 MB in 16s (1555 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
```

- Validando com o `docker inspect` podemos ver que ambos se encontram na mesma rede (No caso a que nós criamos “minha-bridge”)

```
"Networks": {
 "minha-bridge": {
 "IPAMConfig": null,
 "Links": null,
 "Aliases": [
 "51c84dc5c954"
],
 "NetworkID": "9c8de904a6a13307521ffe3986333c1b94187f320a59f2bd3ea0157e52023b2c",
 "EndpointID": "273fdd3539c5710a545ac4e55b81abc47229d9fc4887798dec1b92b9cb4253e",
 "Gateway": "172.18.0.1",
 "IPAddress": "172.18.0.2",
 "IPPrefixLen": 16,
 "IPv6Gateway": "",
 "GlobalIPv6Address": "",
 "GlobalIPv6PrefixLen": 0,
 "MacAddress": "02:42:ac:12:00:02",
 "DriverOpts": null
 }
},
```

Container pong

```

 "Networks": {
 "minha-bridge": {
 "IPAMConfig": null,
 "Links": null,
 "Aliases": [
 "de5cc2db9d4e"
],
 "NetworkID": "9c8de904a6a13307521ffe3986333c1b94187f320a59f2bd3ea0157e52023b2c",
 "EndpointID": "7efcfffe23503d52b947058e7bfe0e4a492549a0df0d1d49c000617871a10a22",
 "Gateway": "172.18.0.1",
 "IPAddress": "172.18.0.3",
 "IPPrefixLen": 16,
 "IPv6Gateway": "",
 "GlobalIPv6Address": "",
 "GlobalIPv6PrefixLen": 0,
 "MacAddress": "02:42:ac:12:00:03",
 "DriverOpts": null
 }
 }
 }
}

```

Container ubuntu1

- Agora vamos dar um `ping pong` através do container ubuntu1

```

root@de5cc2db9d4e:/# ping pong
PING pong (172.18.0.2) 56(84) bytes of data.
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=1 ttl=64 time=2.02 ms
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=4 ttl=64 time=0.067 ms
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=5 ttl=64 time=0.064 ms
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=6 ttl=64 time=0.064 ms
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=7 ttl=64 time=0.062 ms
64 bytes from pong.minha-bridge (172.18.0.2): icmp_seq=8 ttl=64 time=0.065 ms
^C
--- pong ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7282ms
rtt min/avg/max/mdev = 0.056/0.307/2.015/0.645 ms
root@de5cc2db9d4e:/#

```

- Podemos ver que a rede está funcional, pois ping entre hostname está ocorrendo normalmente

## ▼ As redes none e host

- **None**
  - None se refere a nenhuma interface de rede
  - Se rodarmos um container configurado para o driver none como
`docker run -d --network none ubuntu sleep 1d`
  - Em seguida rodar um docker inspect, podemos ver que toda a configuração de network se encontra como none
  - Isso significa que o container está sem nenhuma interface de rede

```

lucasvertare@DESKTOP-2AEKE0L:~$ docker run -d --network none ubuntu sleep 1d
7d003229f8458458b75a5fdb47eb10cf53a48d58b15233c5903bf42e1c088e9
lucasvertare@DESKTOP-2AEKE0L:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
7d003229f845 ubuntu "sleep 1d" 7 seconds ago Up 6 seconds friendly_williams
lucasvertare@DESKTOP-2AEKE0L:~$ docker inspect 7d003229f845

```

```

"Networks": {
 "none": {
 "IPAMConfig": null,
 "Links": null,
 "Aliases": null,
 "NetworkID": "87237b1b53906b50ad69f3c806fb835ab8e53ebb6fd580325e08a173c31a2758",
 "EndpointID": "0c4624404e1fd2b471c90efebd3bc3f748259562adfd558726cf9e005578e65f",
 "Gateway": "",
 "IPAddress": "",
 "IPPrefixLen": 0,
 "IPv6Gateway": "",
 "GlobalIPv6Address": "",
 "GlobalIPv6PrefixLen": 0,
 "MacAddress": "",
 "DriverOpts": null
 }
}
}

```

- Host

- No caso se refere a rede do host em sí
- Se rodarmos o container novamente `docker run -d --network host`
  - No caso essa aplicação em na versão 1.1 está exposta na porta 3000
- Se validarmos um inspect irá mostrar a que está na rede do host

```

lucasvertare@DESKTOP-2AEKE0L:~$ docker run -d --network host luc4sbr1to/app-node:1.0
39f4243756617815dac7cd03cfaac783ae4504a292919169b47836c34cf6c7f
lucasvertare@DESKTOP-2AEKE0L:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
39f424375661 luc4sbr1to/app-node:1.0 "/bin/sh -c 'npm sta..." 23 seconds ago Up 22 seconds exciting_ramanujan
7d003229f845 ubuntu "sleep 1d" 3 minutes ago Up 3 minutes friendly_williams
lucasvertare@DESKTOP-2AEKE0L:~$ docker inspect 39f424375661

```

```

"Networks": {
 "host": {
 "IPAMConfig": null,
 "Links": null,
 "Aliases": null,
 "NetworkID": "ca29d135d9ea64d4954596080ff12f8b9fbbf10c7fbfaf78383aaa919eb4eb98",
 "EndpointID": "e9ddc2cab37e289e7247d886d5cad227029d826ff460b344074e8f7df46f87",
 "Gateway": "",
 "IPAddress": "",
 "IPPrefixLen": 0,
 "IPv6Gateway": "",
 "GlobalIPv6Address": "",
 "GlobalIPv6PrefixLen": 0,
 "MacAddress": "",
 "DriverOpts": null
 }
}

```

- Se validarmos no localhost:3000 podemos identificar que está funcionando normalmente, sem precisarmos declarar nada ao subir

o container (com a flag `-p`)



▼ Host vs None

- A rede host remove o isolamento entre o container e o sistema, enquanto a rede none remove a interface de rede. Estes são os propósitos de utilizar estas redes.

▼ Comunicando aplicação e banco

- Primeiramente vamos utilizar a rede que criamos “minha-bridge”
  - Caso não tenha criado `docker network create --driver bridge minha-bridge`
  - No caso não criamos

```
lucassvertare@DESKTOP-2AEKE0L:~$ docker network ls
NETWORK ID NAME DRIVER SCOPE
a320c7eb4e65 bridge bridge local
ca29d135d9ea host host local
ab418cf8f1df minha-bridge bridge local
87237b1b5390 none null local
lucassvertare@DESKTOP-2AEKE0L:~$
```

- Vamos baixar as imgs
  - De banco, no caso o mongo
    - `docker pull mongo:4.4.6`
  - E uma img já com o app configurado no caso :

- `docker pull aluradocker/alura-books:1.0`

- A aplicação vai rodar na porta 3000

```
lucasvertare@DESKTOP-2AEKE0L:~$ docker pull mongo:4.4.6
4.4.6: Pulling from library/mongo
Digest: sha256:6efa052039903e731e4a5550c68a13c4869ddc93742c716332883fd9c77eb79b
Status: Image is up to date for mongo:4.4.6
docker.io/library/mongo:4.4.6
lucasvertare@DESKTOP-2AEKE0L:~$ docker pull aluradocker/alura-books:1.0
1.0: Pulling from aluradocker/alura-books
Digest: sha256:ed754b9994539e1b9114ecd8686e2e09fdcfeb5e6f1aef71c6e7cdc63afeb08
Status: Image is up to date for aluradocker/alura-books:1.0
docker.io/aluradocker/alura-books:1.0
lucasvertare@DESKTOP-2AEKE0L:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ubuntu latest 3b418d7b466a 2 weeks ago 77.8MB
gcr.io/cicd-conciliacaopdv/ubuntu2 latest 3b418d7b466a 2 weeks ago 77.8MB
gcr.io/cicd-conciliacaopdv/ubuntu latest 3b418d7b466a 2 weeks ago 77.8MB
mongo 4.4.6 61ea24dc52c6 22 months ago 423MB
aluradocker/alura-books 1.0 ebff169e5013 5 years ago 691MB
lucasvertare@DESKTOP-2AEKE0L:~$ |
```

- No caso essa aplicação estará buscando um banco chamado “meu-mongo”, portanto precisamos declarar o nome do container dessa forma

- `docker run -d --network minha-bridge --name meu-mongo mongo:4.4.6`

```
lucasvertare@DESKTOP-2AEKE0L:~$ docker run -d --network minha-bridge --name meu-mongo mongo:4.4.6
ad774f58ce0e11ca058aeaf7f768e9004dbdd31b54bf1f77172cdcc9e0615c97b
lucasvertare@DESKTOP-2AEKE0L:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
ad774f58ce0e mongo:4.4.6 "docker-entrypoint.s..." 7 seconds ago Up 6 seconds 27017/tcp meu-mongo
lucasvertare@DESKTOP-2AEKE0L:~$ |
```

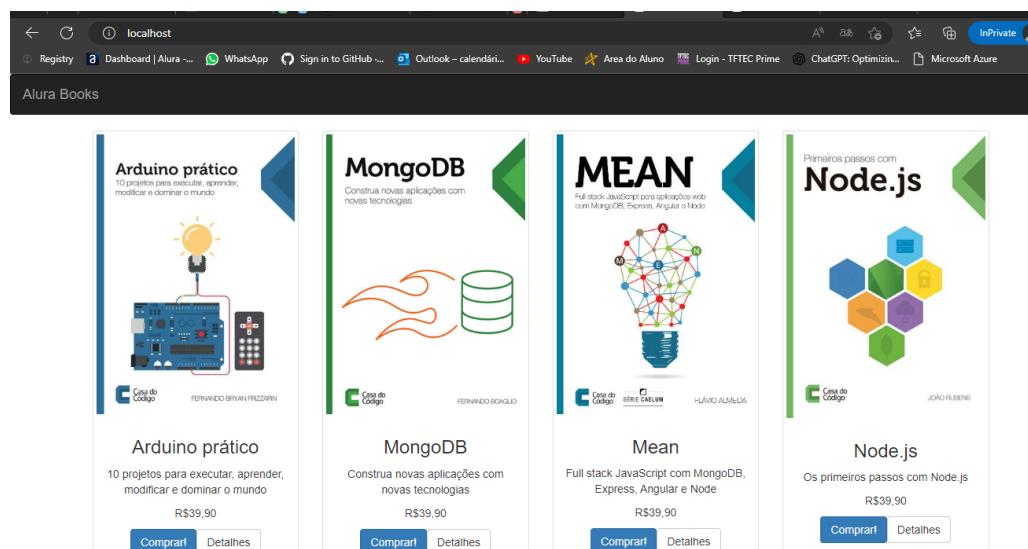
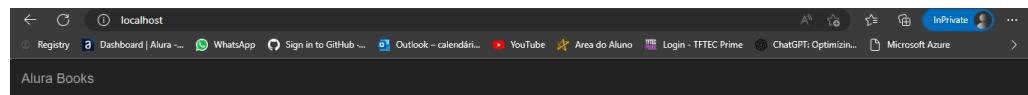
- Portanto agora vamos subir o container da nossa aplicação (lebrando de usar a flag de porta para liberar em nosso host)

- `docker run -d -p 80:3000 --network minha-bridge --name alurabooks aluradocker/alura-books:1.0`

```
lucasvertare@DESKTOP-2AEKE0L:~$ docker run -d -p 80:3000 --network minha-bridge --name alurabooks aluradocker/alura-books:1.0
7d688972e38bc33a650853f7b5c3343520bed402673ccb2a8f688502d051c29
lucasvertare@DESKTOP-2AEKE0L:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
7d688972e38 aluradocker/alura-books:1.0 "npm start" 5 seconds ago Up 4 seconds 0.0.0.0:80->3000/tcp alurabooks
ad774f58ce0e mongo:4.4.6 "docker-entrypoint.s..." 2 minutes ago Up 2 minutes 27017/tcp meu-mongo
lucasvertare@DESKTOP-2AEKE0L:~$ |
```

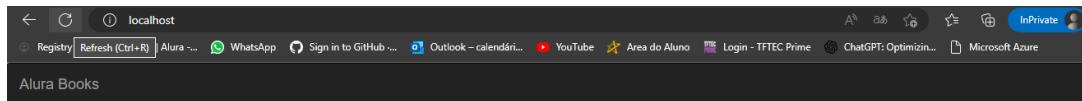
- Com os dois containers em execução na mesma rede vamos validar no navegador se está funcional :

- `localhost:80` e em outra aba o `localhost:80 /seed` para popular o banco

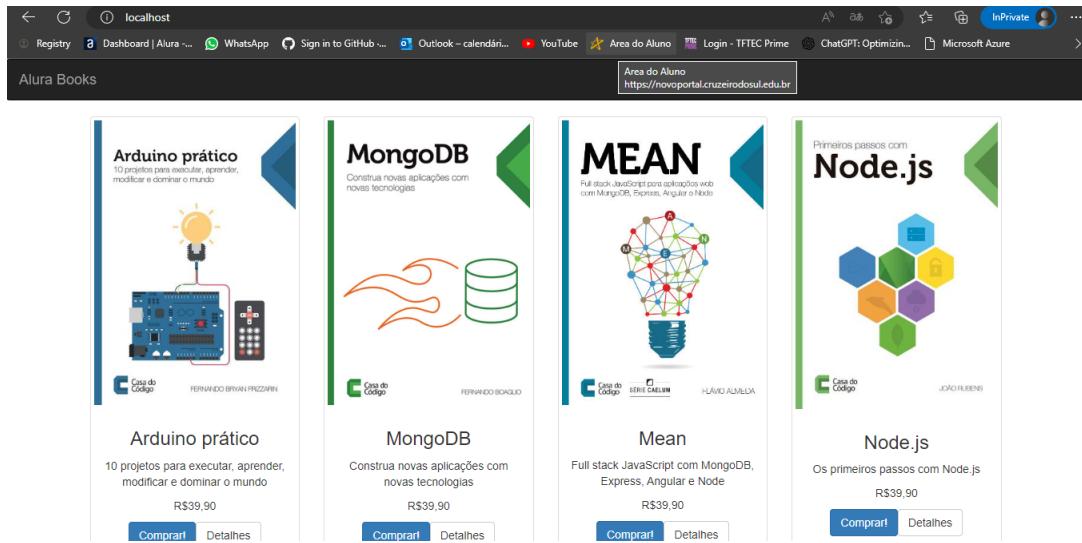


- Se rodarmos um `docker stop meu-mongo` podemos ver que os dados na aplicação irão sumir e se dermos um `start` irá recarregar

```
lucassvertare@DESKTOP-2AEKE0L:~$ docker stop meu-mongo
meu-mongo
lucassvertare@DESKTOP-2AEKE0L:~$ |
```



```
lucassvertare@DESKTOP-2AEKE0L:~$ docker start meu-mongo
meu-mongo
lucassvertare@DESKTOP-2AEKE0L:~$ |
```



- Sendo assim, dessa forma é realizada a comunicação entre containers para o funcionamento de uma aplicação

## ▼ Coordenando containers

### ▼ Conhecendo o docker compose

- Como vimos podemos fazer com que containers se comuniquem um com os outros, porém conforme nossa aplicação vai ficando mais

complexa, cada vez com mais serviços, fica mais difícil termos um controle sobre os containers que estamos criando

- Dessa forma a melhor maneira de coordenarmos todos os nossos containers é através do docker compose que possui exatamente está função
  - Ele irá nos auxiliar a executar, a compor, diversos containers em um mesmo ambiente, através de um único arquivo.
- No caso o arquivo que utilizaremos para compor nossa aplicação é o arquivo yml (ou yaml) que é um tipo de estrutura que vamos seguir baseado em indentação dos nossos serviços
  - Definiremos a versão, serviços, a parte de rede, mas através de um único arquivo
- **Caso você esteja no Windows nesse momento, quando você instalar o Docker você já tem nesse momento o Docker Compose.**
- No linux será necessário instalar seguindo a doc [Install the Compose plugin | Docker Documentation](#)

```
sudo apt-get update
sudo apt install docker-compose
```

```
lk47@Latitude-E6410:~$ sudo apt install docker-compose
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
 python3-attr python3-distutils python3-docker python3-dockerpty
 python3-docopt python3-dotenv python3-jsonschema python3-lib2to3
 python3-persistent python3-setuptools python3-texttable python3-websocket
Suggested packages:
 python-attr-doc python-jsonschema-doc python-setuptools-doc
Recommended packages:
 docker.io
The following NEW packages will be installed:
 docker-compose python3-attr python3-distutils python3-docker
 python3-dockerpty python3-docopt python3-dotenv python3-jsonschema
 python3-lib2to3 python3-persistent python3-setuptools python3-texttable
 python3-websocket
0 upgraded, 13 newly installed, 0 to remove and 4 not upgraded.
Need to get 988 kB of archives.
After this operation, 5.239 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

## Docker Compose overview

Learn how to use Docker Compose to define and run multi-container applications with this detailed introduction to the tool.

 <https://docs.docker.com/compose/>



### ▼ Utilização do docker compose

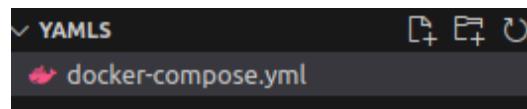
- **O Docker Compose irá resolver o problema de executar múltiplos containers de uma só vez e de maneira coordenada, evitando executar cada comando de execução individualmente.**
  - Este é o principal objetivo do Docker Compose.

### ▼ Definindo os serviços

- Vamos subir a aplicação que subimos anteriormente através do docker-compose
- Vamos criar uma pasta em nosso desktop para que possamos salvar os arquivos YAML, em seguida vamos abrir o VScode para melhor visualização de nossos códigos

```
Lk47@Latitude-E6410:~$ mkdir Desktop/YAMLS
Lk47@Latitude-E6410:~$ cd Desktop/YAMLS/
Lk47@Latitude-E6410:~/Desktop/YAMLS$ code .
```

- Vamos criar um arquivo chamado docker-compose.yml



- No arquivo iremos configurar da seguinte forma :

```
#versão do YAML
version: "3.9"
#As programas que iremos rodar nos containers
services:
 #No container do banco de serviço vamos definir a img, nome, e rede
 mongodb:
 image: mongo:4.4.6
```

```

 container_name: meu-mongo
 networks:
 - compose-bridge
#No container da aplicação, mesmos parametros, porém com a porta
alurabooks:
 image: aluradocker/alura-books:1.0
 container_name: alura-books
 networks:
 - compose-bridge
 ports:
 - "80:3000"
#Vamos criar a rede bridge em serviços também
network:
 compose-bridge:
 drive: bridge

```

```

EXPLORER ... Welcome docker-compose.yml
YAMLs docker-compose.yml
```
version: "3.9"
services:
  mongodb:
    image: mongo:4.4.6
    container_name: meu-mongo
    networks:
      - compose-bridge
  alurabooks:
    image: aluradocker/alura-books:1.0
    container_name: alura-books
    ports:
      - "80:3000"
    networks:
      - compose-bridge
networks:
  compose-bridge:
    driver: bridge
```

```

- Após isso, salvamos o arquivo, e executamos através do terminal o comando `docker-compose up`

```

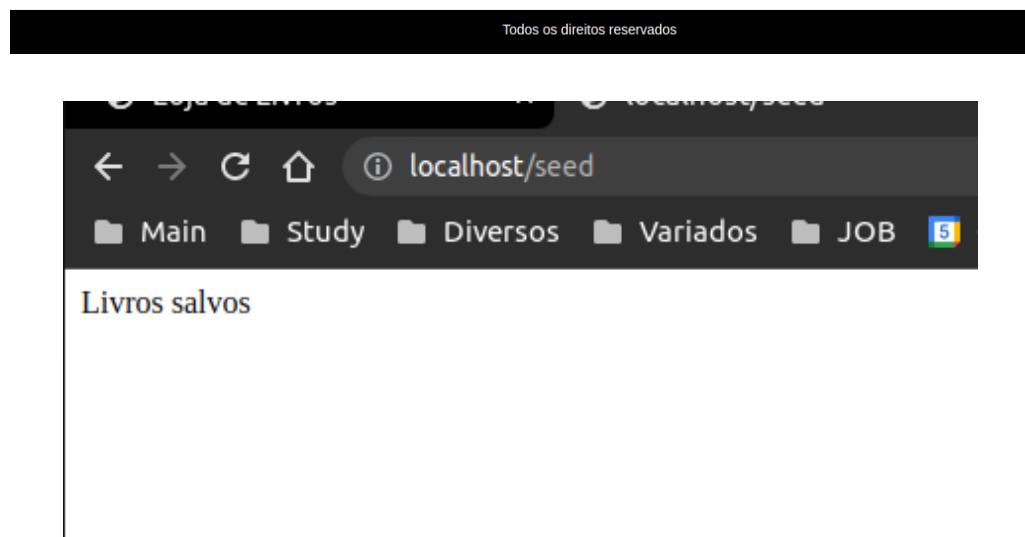
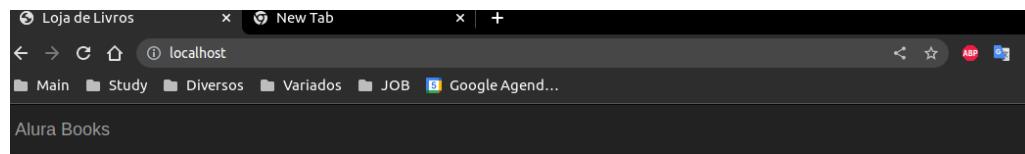
lk47@Latitude-E6410:~/Desktop/YAML$ ls
docker-compose.yml
lk47@Latitude-E6410:~/Desktop/YAML$ docker-compose up
Creating alura-books ... done
Creating meu-mongo ... done
Attaching to meu-mongo, alura-books
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.846+00:00"},"s":"I", "c":"CONTROL", "id":23285, "a
abling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'}
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.868+00:00"},"s":"W", "c":"ASIO", "id":22601, "a
configured during NetworkInterface startup"
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.868+00:00"},"s":"I", "c":"NETWORK", "id":4648601, "a
Open unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpen
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.870+00:00"},"s":"I", "c":"STORAGE", "id":4615611, "a
starting","attr":{"pid":1,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"0612c1b1645b"
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.870+00:00"},"s":"I", "c":"CONTROL", "id":23403, "a
fo","attr":{"buildInfo":{"version":"4.4.6","gitVersion":"72e66213c2c3eab37d9358d5e78ad7f5c1d0d0d7","open
018","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu1804","distarch":"x86_64","targ
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.870+00:00"},"s":"I", "c":"CONTROL", "id":51765, "a
g System","attr":{"os":{"name":"Ubuntu","version":"18.04"}}}
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.870+00:00"},"s":"I", "c":"CONTROL", "id":21951, "a
set by command line","attr":{"options":{"net":["bindIp":"*"]}}}
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.875+00:00"},"s":"I", "c":"STORAGE", "id":22297, "a
e XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.o
:["startupWarnings"]}
meu-mongo | {"t":{"$date":"2023-05-12T10:01:03.875+00:00"},"s":"I", "c":"STORAGE", "id":22315, "a
wiredTiger","attr":{"config":{"create,cache_size=1379M,session_max=33000,eviction=(threads_min=4,threads_
(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=1000
le_minimum=250),statistics_log=(wait=0),verbose=[recovery_progress,checkpoint_progress,compact_progress]
alura-books | npm info it worked if it ends with ok
alura-books | npm info using npm@5.3.0
alura-books | npm info using node@v8.2.1
alura-books | npm info lifecycle alura-docker@1.0.0-prestart: alura-docker@1.0.0
alura-books | npm info lifecycle alura-docker@1.0.0-start: alura-docker@1.0.0
alura-books |
alura-books | > alura-docker@1.0.0 start /var/www
alura-books | > node server.js
alura-books |
meu-mongo | {"t":{"$date":"2023-05-12T10:01:04.889+00:00"},"s":"I", "c":"STORAGE", "id":22430, "a

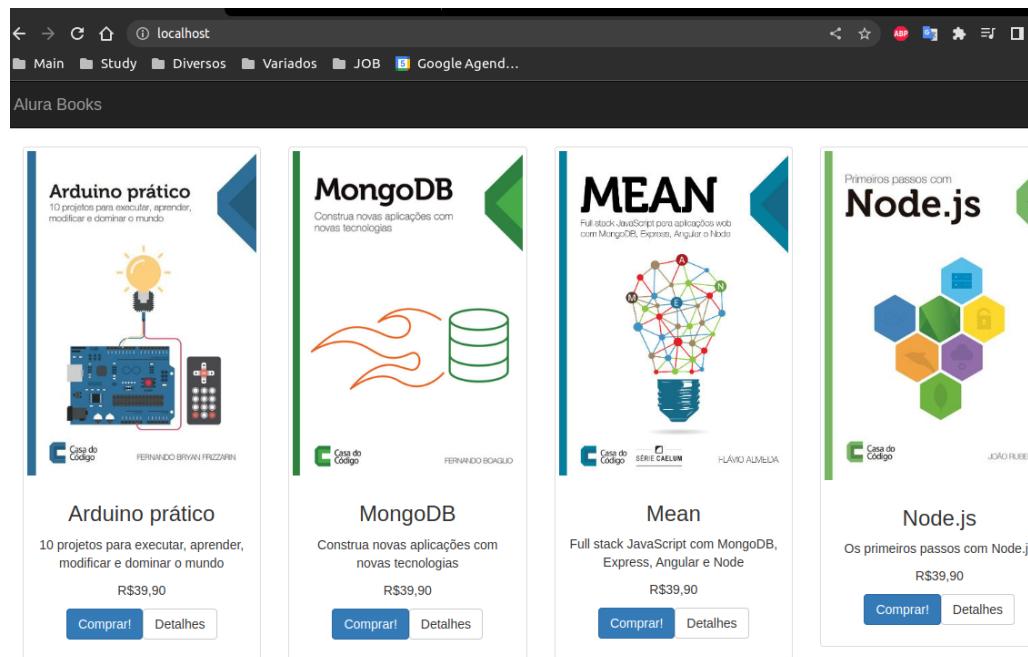
```

- Após rodar o comando podemos ver com o `docker ps` os container já em execução

| CONTAINER ID  | IMAGE                       | COMMAND                  | CREATED       | STATUS       | PORTS                                 |
|---------------|-----------------------------|--------------------------|---------------|--------------|---------------------------------------|
| 465710fc0c1f9 | aluradocker/alura-books:1.0 | "npm start"              | 2 minutes ago | Up 2 minutes | 0.0.0.0:80->3000/tcp, :::80->3000/tcp |
| 0612c1b1645b  | mongo:4.4.6                 | "docker-entrypoint.s..." | 2 minutes ago | Up 2 minutes | 27017/tcp                             |
| meu-mongo     |                             |                          |               |              |                                       |

- Se formos na pagina localhost, abrimos uma nova aba localhost/seed, voltarmos para a primeira e darmos um F5





- Nossa aplicação irá funcionar normalmente
- Dessa forma podemos controlar e coordenar a criação de containers através de um único arquivo

#### ▼ Parâmetros para serviços

- **Nome do container**
  - O nome do container é um parâmetro que pode ser definido com o Docker Compose.
- **Rede que o container irá se conectar**
  - A rede é um parâmetro que pode ser definido com o Docker Compose.
- **Imagen a ser utilizada**
  - A imagem é um parâmetro que pode ser definido com o Docker Compose.

#### ▼ Complementando o compose

## The Compose file

 <https://docs.docker.com/compose/compose-file/03-compose-file/>



- Podemos colocar complementos no arquivo do compose, assim podemos parametrizar quais containers e em qual ordem iremos subir os mesmos
  - Lembrando que ele não irá esperar a aplicação dentro do container estar pronto
- No caso vamos utilizar o `depends_on`, no arquivo de configuração do composer, vamos add na linha está sintaxe no container da aplicação:

```
version: "3.9"
services:
 mongodb:
 image: mongo:4.4.6
 container_name: meu-mongo
 networks:
 - compose-bridge
 alurabooks:
 image: aluradocker/alura-books:1.0
 container_name: alura-books
 ports:
 - "80:3000"
 networks:
 - compose-bridge
 depends_on:
 - mongodb
 networks:
 compose-bridge:
 driver: bridge
```

No caso a aplicação alurabooks irá depender do mongodb para subir

- Vamos salvar o arquivo e rodar novamente a execução do container `docker-compose up`, e poderemos ver que o docker irá trazer todos os containers em uma ordem, primeiro o banco e depois da aplicação

```
meu-mongo |
er message","att
meu-mongo |
er message","att
recovery timest
meu-mongo |
er message","att
oldest timestamp
meu-mongo |
er message","att
snapshot min: 1
meu-mongo |
er opened","attr
meu-mongo |
er recoveryTimes
meu-mongo |
logging setting
meu-mongo |
p monitor starti
meu-mongo |
ontrol is not en
meu-mongo |
trol is enabled
meu-mongo |
zing full-time c
meu-mongo |
,"attr":{"addr
meu-mongo |
,"attr":{"addr
meu-mongo |
onnections","att
alura-books |
ent` option if u
alura-books |
meu-mongo |
cepted","attr":{
meu-mongo |
,"attr":{"remot
x","architecture
alura-books |
alura-books |
15
```

- Podemos rodar o compose no modo background com o -d `docker-compose up -d`

```
lk47@Latitude-E6410:~/Desktop/YAML$ docker-compose up -d
starting meu-mongo ... done
starting alura-books ... done
lk47@Latitude-E6410:~/Desktop/YAML$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
 NAMES
6224513db795 aluradocker/alura-books:1.0 "npm start" About a minute ago Up 2 seconds 0.0.0.0:80->3000/tcp, ::80->3000/tcp
tcp alura-books
0612c1b1645b mongo:4.4.6 "docker-entrypoint.s..." 6 hours ago Up 3 seconds 27017/tcp
meu-mongo
lk47@Latitude-E6410:~/Desktop/YAML$
```

- E para encerrar a aplicação e excluir os containers, podemos rodar o docker-compose down `docker-compose down`

```
lk47@Latitude-E6410:~/Desktop/YAML$ docker-compose down
Stopping alura-books ... done
Stopping meu-mongo ... done
Removing alura-books ... done
Removing meu-mongo ... done
Removing network yaml_compose-bridge
lk47@Latitude-E6410:~/Desktop/YAML$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
lk47@Latitude-E6410:~/Desktop/YAML$
```

- **O Docker Compose é uma ferramenta de coordenação de containers;**
- **Como instalar o Docker Compose no Linux;**
- **Como iniciar containers em conjunto com o comando `docker-compose up`;**
- **Como criar um arquivo de composição e definir instruções de containers, redes e serviços.**