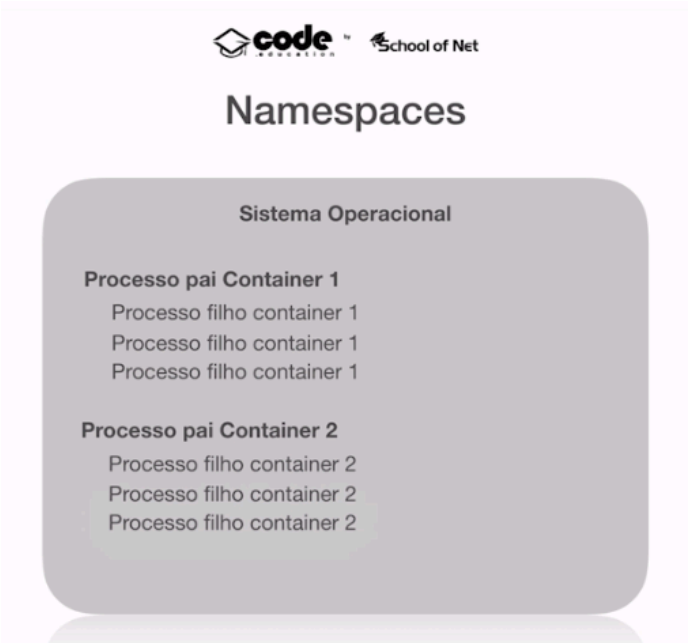
Curso Full Cycle - Módulo: Docker

O que é um container:

É um padrão de unidade de software que empacota código e todas as dependências de uma aplicação fazendo a mesma seja executada rapidamente de forma confiável de um ambiente computacional para outro.

Namespaces:

Namespaces permite realizar o isolamento dos processos.
Ex: processo pai com namespace X e todos os subprocessos deste processo pai, partem desse namespace X



Cgroups :

Criado pela Google, permite delimitar os recursos computacionais de um container

OFS (Overlay File System):

Trabalhar com camadas / layers de configuração da imagem do container. Não precisa de cópias inteiras do OS. As dependências também podem ser compartilhadas, aproveitando o kernel do sistema operacional da máquina que roda o container. O container não tem toda a imagem do OS, e sim só o que ele precisa para rodar.

Imagens:

Em containers, criamos ambientes em camadas, trabalhando com dependências. Se necessário alterar uma camada/layer, é possível alterar apenas ela e construir novamente a imagem, sem afetar as demais. Possui um nome e versão.

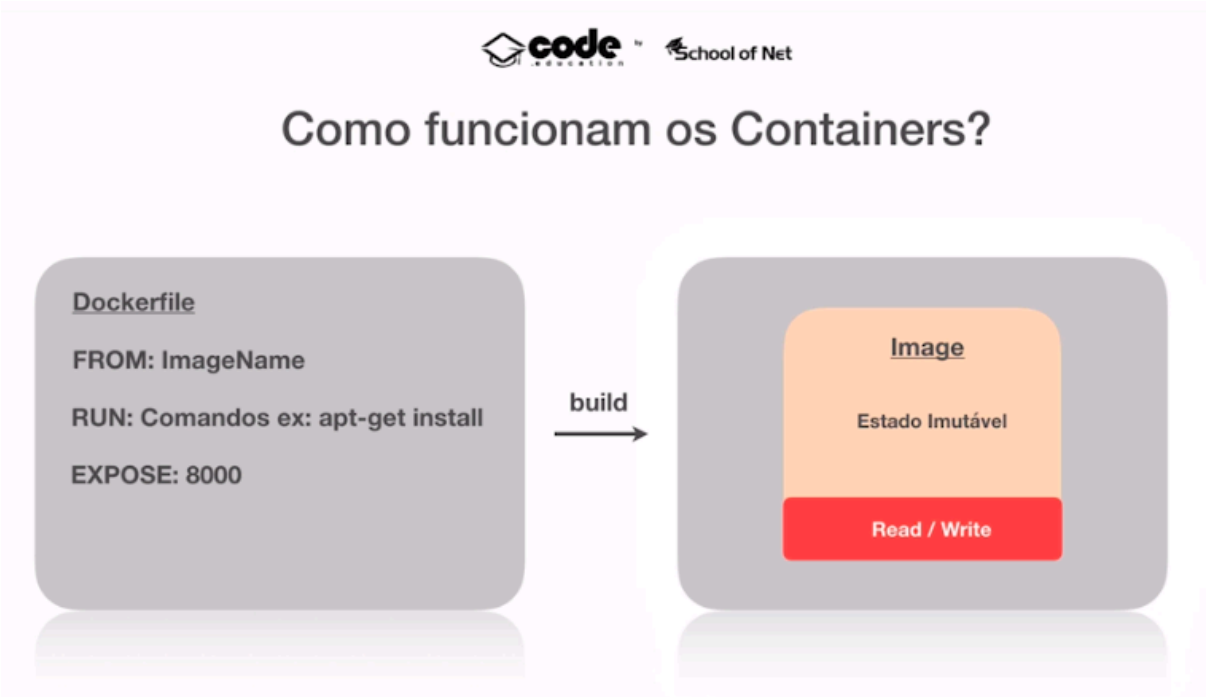
Dockerfile:

Arquivo declarativo que você escreve como vai ser a imagem que você vai construir, partindo de uma imagem existente. Quando executamos o build de um dockerfile, uma imagem é gerada.

sintaxe:

FROM: ImageName (ex: Ubuntu Latest) **PULL de um Image Registry**
RUN: Comandos ex: apt-get install …
EXPOSE: expor portas ex: 8000

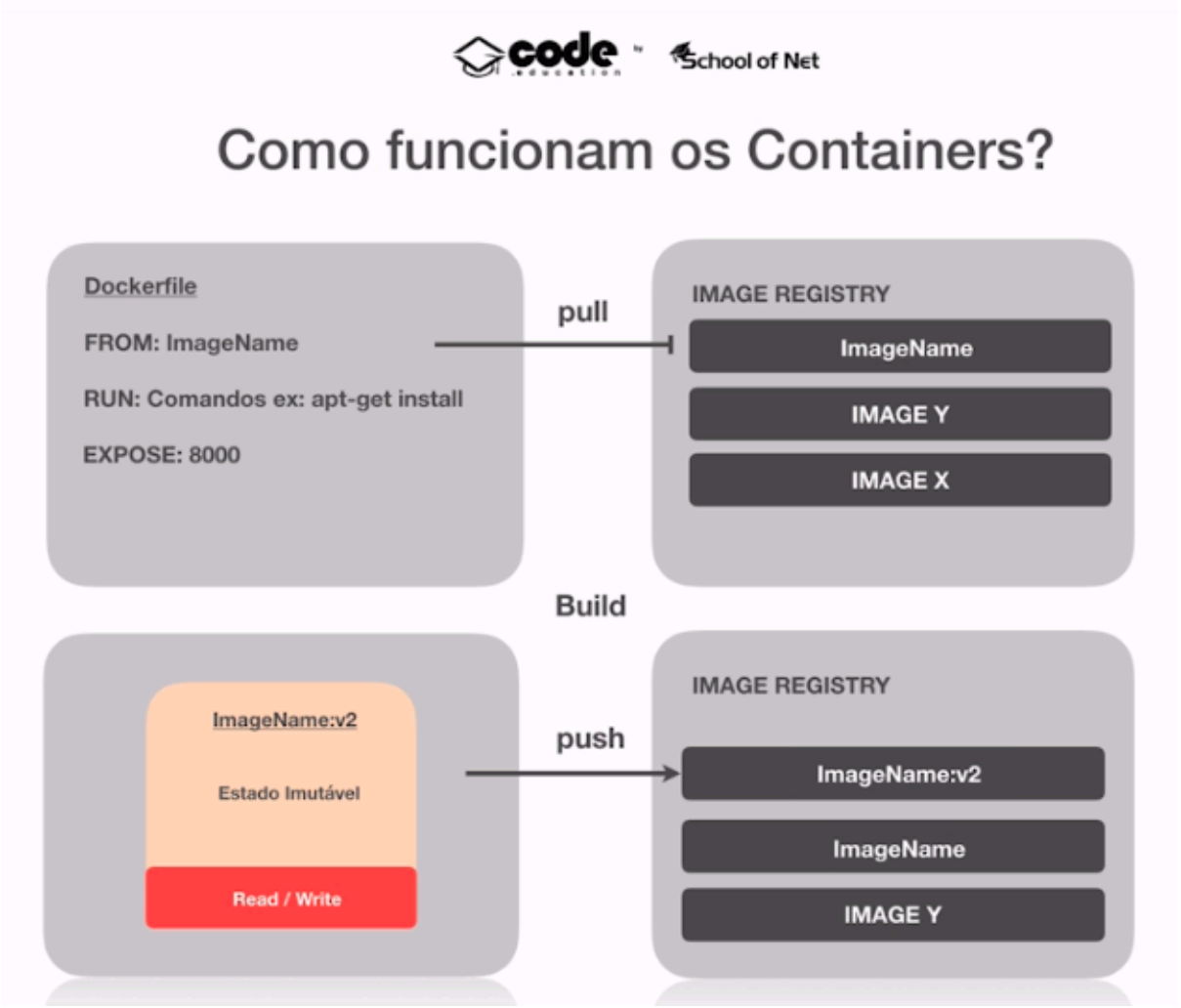
A imagem de um container tem um estado imutável, ou seja, não pode ser editada. Quando entramos no container e alteramos ou escrevemos algo, o Docker cria uma nova camada de Read/Write, e é nela que ficam salvos estes dados. Se caso você matar esse processo, o container é destruído e essa camada de read e write também é perdida.



Outra forma de criar uma imagem, é pegar uma imagem já existente, escrever na camada de Read/Write e dar um Commit, sendo gerada uma nova Imagem.

Onde ficam as imagens?

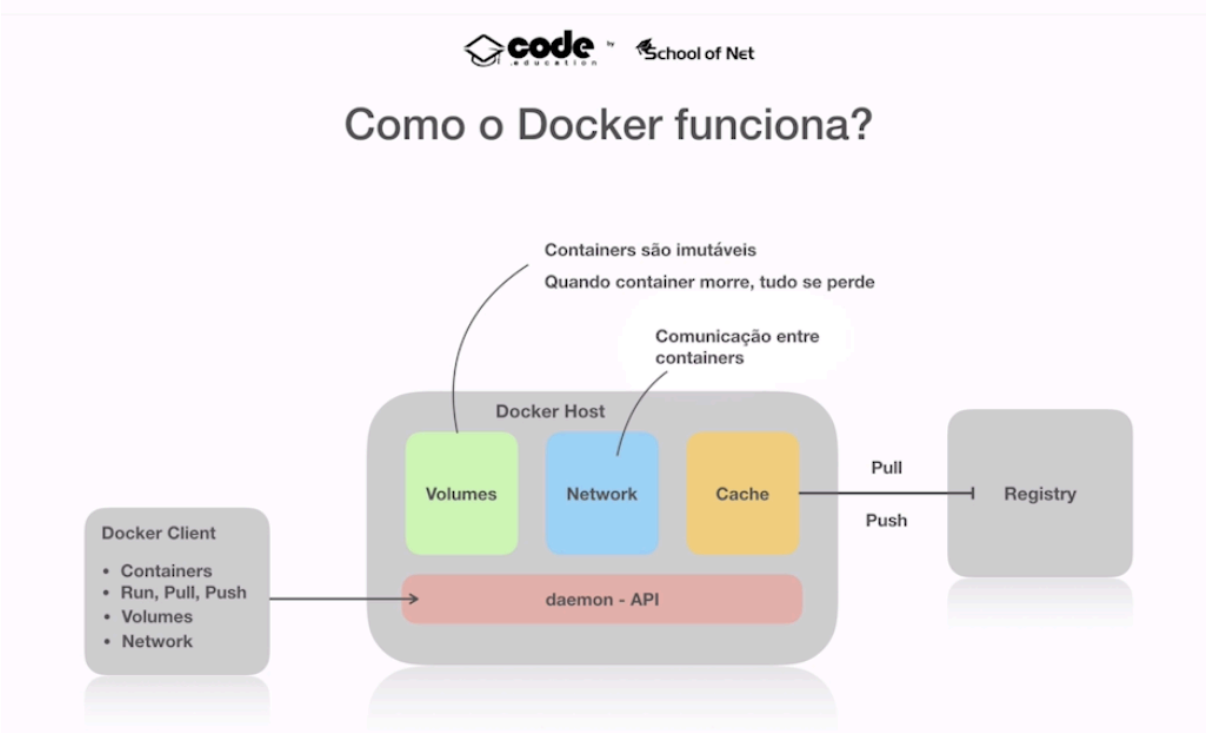
As imagens ficam armazenadas em um **Image Registry**, semelhante ao Github, disponibilizando download destas.



O Docker:

Docker conseguiu unificar os conceitos de Namespaces, CGroups e File System através do **Docker Host**. Este host roda nossa imagem em um container, através de uma **deamon** que disponibiliza uma API.

Para falar com o Docker Host, vai precisar com **Docker Client** (quando digitamos no terminal docker … estamos invocando o docker client) que se comunica com a API do deamon.



Dentro do docker temos também:

- cache: responsável por armazenar as imagens que baixamos do registry e reutilizar caso sejam necessárias
- volumes: como as imagens são imutáveis, os dados que são necessários serem persistidos são salvos em um volume, que nada mais é que uma pasta compartilhada entre o Docker e a máquina que está rodando ele. Se baixar o container ou apagá-lo, o volume continua intacto na pasta fora do container.
- network: permite um gerenciamento de redes para comunicação entre os recursos da aplicação rodando no container.

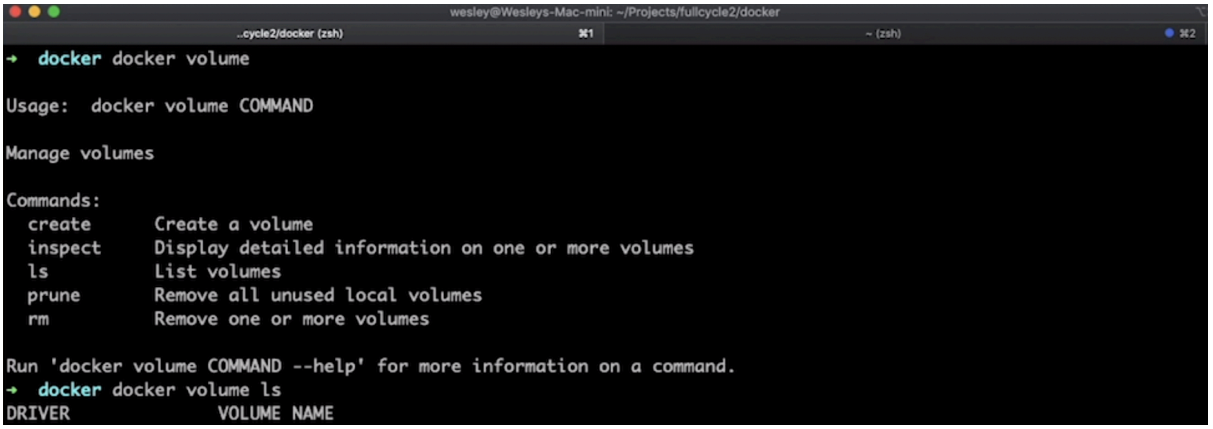
O Docker foi feito para rodar no Linux e é onde ele fica mais performático.

WSL: Subsistema Windows para Linux

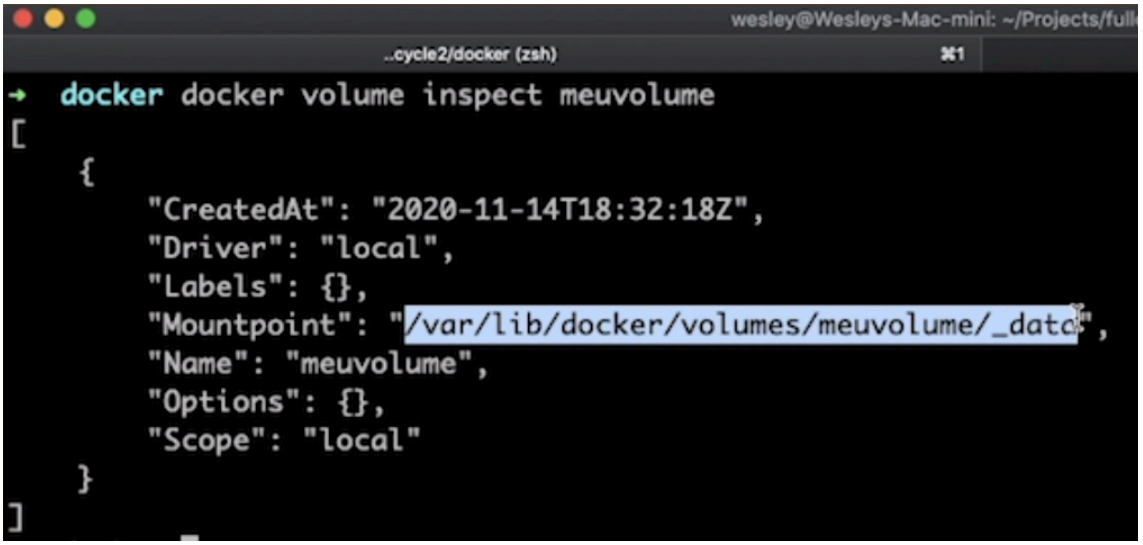
- WSL1: somente via terminal, ambiente virtualizado Linux, não tem o kernel completo, não rodava Docker por esse motivo. Tinha desempenho ruim.
- WSL2: Execução completa do Kernel do Linux, manipulado por terminal. melhor desempenho. Se acessar o C: continua com baixo desempenho, aconselha utilizar as pastas de dentro do WSL2 para melhor desempenho. Agora com suporte ao docker. Usa o Virtual Machine Platform (Uso do HiperV). Windows 10 Home ou Pro, build >= 19.03.

Link util: https://github.com/codeedu/wsl2-docker-quickstart

Volumes



*** comandos do docker volumes**



*** pasta onde está seu volume.**

Quando criado um volume, ele pode ser compartilhado entre várias imagens, ex:

$ docker volume create meuvolume
$ docker run –name nginx -d –mount type=**volume**, source=meuvolume,target:/app nginx
$ docker run –name nginx2 -d –mount type=**volume**, source=meuvolume,target:/app nginx
$ docker run –name nginx3 -d –mount type=**volume**, source=meuvolume,target:/app nginx

$ docker exec -it nginx bash
$ cd /app
$ touch oi

$ docker exec -it nginx2 bash
$ cd /app
$ ls

- verá o arquivo oi que foi criado no container nginx

pode utilizar o -v também:

$ docker run –name nginx4 -d -v meuvolume:/app nginx

Se máquina está lotada, pode limpar os volumes não utilizados:
→ $ docker volume prune

Imagens

Pode baixá-las do image registry do docker-hub: https://hub.docker.com/search?q=

Para criar sua imagem:

docker build -t [seu_user_docker_hub/nm_autor]/[nm_imagem]:[version] [path-onde-esta-dockerfile]

Ex:

docker build -t rodrigospimenta/nginx-com-vim:latest .

Obs: Neste caso usou-se "." ponto final para informar que o arquivo Dockerfile está na própria raiz onde o terminal foi aberto.

Exemplo de arquivo Dockerfile:

Dockerfile ●

Dockerfile > ...

```
1  FROM nginx:latest
2
3  WORKDIR /app
4
5  RUN apt-get update
6  RUN apt-get install vim -y
7
8  COPY html/ /usr/share/nginx/html
```

Dockerfile > ...

```
1  FROM ubuntu:latest
2
3  ENTRYPOINT [ "echo" , "Hello" ] // Código fixo
4
5  CMD [ "World" ] // Codigo variável
```

- Imagem no dockerhub: https://hub.docker.com/layers/226416448/rodrigospimenta/nginx-fullcycle/latest/images/sha256-1c8f8a35090ec779e810ec1f649ea9444b5f059e92ed888b0dfe9ff12f947323?context=repo

Networks

Tipos/drivers de network:

- **bridge**: ponte, quando cria um network no docker e não informa nada, é criada uma bridge. Utilizada normalmente quando quer que um container se comunique facilmente com outro.
- **host**: mescla network do docker com a network do host do docker. Faz com que a network do container, esteja na mesma rede da máquina que está rodando o container. Ex: Estou rodando um mysql na máquina local na porta 3306. Se eu subir uma aplicação no container docker rodando nesta mesma máquina, a aplicação poderá acessar o Mysql. Não precisa utilizar a exposição de portas (comando -p 8080:80 na execução do container)
- **overlay**: não é muito comum. Criar uma rede de comunicação para container que estão rodando em máquinas diferentes. Utilizando no Docker Swarm para comunicação entre os containers gerenciados por ele.
- **macvlan**: também não muito comum. Seta um mac address no container e ele aparecerá como um novo dispositivo plugado na sua rede.
- **nenhum**: container sem rede, rodando de forma isolada.

Os formatos que mais se trabalha são o **bridge** e **host**. O modo **overlay** é mais utilizado quando se utiliza Docker Swarm.

** Obs: o driver **host** não funciona corretamente no mac, pois o docker foi feito para rodar no linux. No mac o docker desktop emula uma máquina para trabalhar com containers, sendo neste caso o Docker Host uma máquina virtual. Neste caso, quando utilizar o driver **host**, ao invés de ser mesclado a network do container com a da sua máquina, ele fundirá a rede do container com a rede da máquina virtual do Docker Host. Isso não ocorre se estiver usando um OS linux ou Windows com WSL2.

Colinha de Comandos

Comando	**Descrição**
docker ps	lista todas os containers
docker ps -a	lista os containers que estão rodando
docker rm [hash/nm_imagem]	remove o container, se adicionar um -f desligar se ela estiver rodando
docker exec -it [hash/nm_imagem] bash	acessa via bash o terminal do container rodando
docker attach [hash/nm_imagem]	acessa via bash o terminal do container rodando
docker stop [hash/nm_imagem]	para a execução da imagem no container
docker start [hash/nm_imagem]	inicia a execução da imagem no container
docker images	lista as imagens baixadas
docker rmi [hash/nm_imagem]	apaga uma imagem local
docker run [nm_imagem:versao]	roda uma imagem, podendo adicionar: ● **--name** : alias para a imagem ● **-d** : detached, libera o terminal após rodar ● **-it** : modo interativo, mantem o terminal aberto, utilizado com bash ● **-p 8080:80** : mapeamento de portas local:container ● **-v ~/docker/html:/user/share/nginx/html** : volume, fazer bind mount de pasta local com container, pode usar "$(pwd)" para pegar o caminho da pasta atual para simplificar, cria pasta se não existir ● **--mount type:bind,source:"$(pwd)/html",target:/usr/share/nginx/html** : modo mais novo de montar volume, não cria a pasta se ela não existir
docker build -t rodrigospimenta/nginx-com-vim:latest .	Criação da imagem seguindo a sintaxe docker build -t nm_autor/nm_imagem:versao [path_onde_esta_o_Dockerfile] ● **-t:** nome da tag ● –no-cache: não gera cache para o build

docker rm $(docker ps -a -q) -f	Pode concatenar comandos, neste caso parar e limpar todos os containers. [docker ps -a -q] lista todos os ids dos containers ativos e inativos. o "-f" força a parada dos ativos para remoção.
sudo docker cp [hash/nm_imagem]:/foo.txt ~/Desktop/foo.txt	Copiar um arquivo de dentro de um container que está rodando para a máquina que está mantendo o container.
sudo docker cp ~/Desktop/to-be-copied.txt [hash/nm_imagem]:/to-be-copied.txt	Copiar um arquivo que está na máquina que mantém um container, para dentro de um container ativo.
docker network	Lista comandos para gerenciamento da rede do docker. podemos adicionar: ● **ls**: listar as networks ● **prune**: remove as redes não utilizadas ● **inspect [nm/id_netwrok]**: traz detalhes da network ● **create -- driver [tipo_rede] [nm_rede]**: cria uma nova rede. ex: docker network -- driver bridge minharede ● **connect [nm_rede] [nm_imagem]**: conecta um container a uma rede ● **disconnect [nm_rede] [nm_imagem]**: desconecta um container de uma rede
ctrl+p seguido de ctrl+q	sair do terminal attached em um container, sem parar a imagem
http://host.docker.internal:[porta]	url que permite de dentro do seu container, acessar algum recurso que está rodando na máquina host do container.
docker-compose up	para subir o docker compose, na mesma pasta ● **-d** : detached ● **--build**: rebuilda seguindo a imagem nova do dockerfile
docker-compose down	para baixar o docker
docker-compose ps	lista os containers de docker-compose que estão rodando
docker-compose build --no-cache	Buildar o docker-compose sem cache
FROM maven:3.6.3-jdk-11 #WORKDIR /app # COPY /caminho/do/projeto/spring /app	CMD ["bash"] - Segura a execução de um Dockerfile para poder acessar via -it no docker run. Ex: docker run -it pepper/tests-app:latest

ENV TZ="America/Sao_Paulo" ENV MAVEN_OPTS="-Xverify:none -Xms128m -Xmx2G" RUN apt-get CMD mvn -v CMD ["bash"]	