ABRIR DOCKER NO POWERSHELL

docker run hello-world

O docker possui uma estrutura bem ampla que pode ser vista com:
docker run --help

O docker busca imagens no **docker hub** caso não a encontre localmente, se a imagem existir ele irá baixar, caso contrário ele dirá que a imagem não existe. O docker hub pode ser visto em:
https://hub.docker.com/

Imagens oficiais possuíram um selo
docker pull ubuntu
docker run ubuntu

Para listar os containers em execução usa-se:
docker ps ou *docker container ls*

Para listar os containers que foram criados usa-se:
docker os -a ou *docker container ls -a*

docker run ubuntu sleep 1d

Para encerrar um container usa-se:
docker stop [id] ou *[container name]*

Para encerrar todos os containers usa-se:
docker stop $(docker container ls -q)

Para encerrar forçadamente um container usa-se:
docker stop –t=0 [id] ou *[container name]*

Para iniciar um container usa-se:
docker start [id] ou *[container name]*

Para pausar e continuar um container usa-se:
docker pause [id] ou *[container name]*
docker unpause [id] ou *[container name]*

Para remover um container usa-se:
docker rm [id] ou *[container name]*

Para executar um comando interativo dentro de um container usa-se:
docker exec -it [id] ou *[container name] bash*

Para rodar um container no modo interativo usa-se:
docker run -it [id] ou *[container name] ubuntu bash*

Comandos do Linux:
cd
touch meu-arquivo-exemplo.txt
Ls
apt-get update

Porém na nossa máquina esse arquivo não existe, porque são sistemas de arquivos isolados, graças ao *namespace* de MNT. É o *file system* que está completamente isolado.

Nós podemos fazer diversas coisas agora nesse Ubuntu, porque ele vai estar devidamente isolado do nosso sistema. Nós poderíamos executar vários comandos, conforme nossa demanda, e vai estar tudo isolado do nosso sistema original. Links para estudo:

https://bikramat.medium.com/namespace-vs-cgroup-60c832c6b8c8

https://etcd.dev/2021/09/20/containers-o-que-sao-namespaces-e-cgroups/

PORTAS

Dockerhub -> *dockersamples/static-site*

docker pull [image name]

Para rodar uma imagem sem travar *(detached)* o terminal, usa-se:
docker run -d [image name]

Testando o site-> localhost:80 – **Por que não funcionou?!**

Graças aos namespaces, as portas dos containers são isoladas das portas que estão na minha máquina.

Para parar e remover uma imagem de uma vez só, usa-se:
docker rm [id] ou *[container name] --force*

Para rodar uma imagem com mapeamento automático das portas, usa-se:
docker run –d –P [image name]

Para visualizar o mapeamento das portas, usa-se:
docker port [id] ou *[container name]*

Para rodar uma imagem com mapeamento manual das portas, usa-se:
docker run –d –p [porta da sua máquina]:[porta da imagem] [image name]

IMAGENS

Para ver as imagens localmente, usa-se:
docker images

Para ver os detalhes de uma imagem localmente, usa-se:
docker inspect [image id]

Para ver o histórico de camadas de uma imagem localmente, usa-se:
docker history [image id]

Resumindo, uma imagem é um conjunto de camadas empilhadas para formar determinada regra de execução de um container.

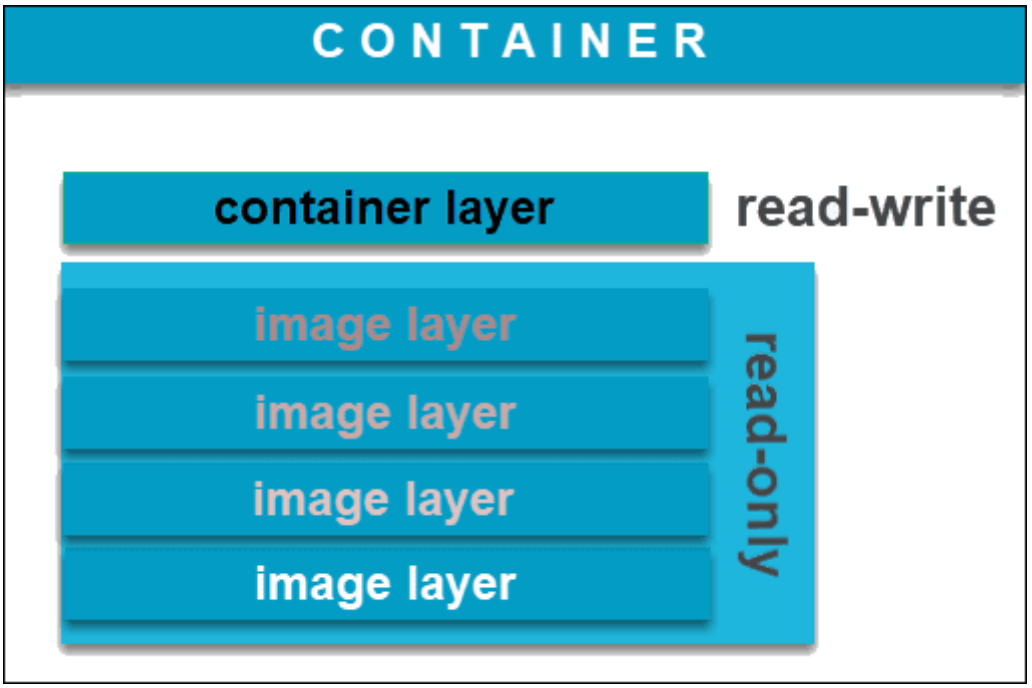
Quando fizemos nosso `docker run` pela primeira vez ou simplesmente um `docker pull` para não executar o container, mas para só baixar a imagem, nós fazemos o download das nossas imagens, das nossas camadas. Mas pode ser que, por exemplo, no nosso *host* nós já tenhamos algumas das camadas que queremos. Então no momento em que fizermos um `pull` ou um `run`, que vai fazer um `pull` consequentemente, nós vamos fazer simplesmente download só das camadas que necessitamos.

Então o Docker é inteligente o suficiente para reutilizar essas camadas para compor novas imagens. Com isso, conseguimos ter uma performance muito boa nesse sentido, já que não precisaremos ter informações duplicadas ou triplicadas. Porque nós conseguimos reutilizar as camadas em outras imagens.

Lembrando que um container é read only.

Mas como conseguimos escrever na imagem do *UBUNTU* se a imagem que gera o nosso container é apenas para leitura, ou *read only*? Se ela é bloqueada para escrita como é que o container consegue escrever informação dentro dela?

Porque no fim das contas, quando criamos o container, o container nada mais é do que uma imagem com uma camada adicional de *read-write*, de leitura e escrita.



https://phoenixnap.com/kb/docker-image-vs-container

Para criar uma imagem usaremos um arquivo *dockerfile* (baixar exemplo no github)
Dentro da pasta o arquivo de conter:

```
FROM node:14
WORKDIR /app-node
COPY . .
RUN npm install
ENTRYPOINT npm start
```

Subindo a imagem:
(PRECISA ESTAR NO MESMO DIRETORIO DO DOCKERFILE) docker build -t huilson/app-node:1.0 .

docker images
docker ps
docker run –d –p 8080:3000 huilson/app-node:1.0

APRIMORANDO

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

```
FROM node:14
WORKDIR /app-node
ARG PORT_BUILD=80
ENV PORT=$PORT_BUILD
EXPOSE $PORT_BUILD
COPY . .
RUN npm install
ENTRYPOINT npm start
```

docker build -t huilson/app-node:1.1 .
docker ps

Subindo a imagem no Dockerhub

docker login -u [user name]
password:

Opcional

Docker tag [nome da image] [nome da image novo]
Exemplo: *Docker tag huilson/app-node:1.0 userDockerHub/app-node:1.0*

docker push [nome da image]