

## Relatório - Card 17 - Docker e Containers para Aplicações (III)

Jefferson korte junior

### Seção 1 - Instalação

Na primeira aula do curso, ele nos ensina a baixar o Docker no nosso sistema operacional, e alguns dos comandos básicos que tem.

### Seção 2 - Interface de linha de comando do Docker (CLI)

Já na segunda aula começamos a mexer no Command line interface. segue sempre o mesmo padrão, docker na frente, (objeto) e a (ação):

Interface de linha de comando para controlar a docker engine.

docker {objeto} {ação}

docker {container | image | network | volume | etc } {ls | inspect | rm | create}

\$ docker container rm alpine

\$ docker image ls



Também é possível ir no terminal e digitar **docker --help** que listará todos os comandos possíveis que podemos fazer no docker:

```
C:\Users\jeffe>docker --help
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
run      Create and run a new container from an image
exec     Execute a command in a running container
ps       List containers
build    Build an image from a Dockerfile
pull     Download an image from a registry
push     Upload an image to a registry
images   List images
login    Authenticate to a registry
logout   Log out from a registry
search   Search Docker Hub for images
version  Show the Docker version information
info     Display system-wide information

Management Commands:
ai*      Ask Gordon - Docker Agent
builder  Manage builds
buildx*  Docker Buildx
compose* Docker Compose
container Manage containers
context  Manage contexts
debug*   Get a shell into any image or container
desktop* Docker Desktop commands (Beta)
dev*     Docker Dev Environments
extension* Manages Docker extensions
feedback* Provide feedback, right in your terminal!
image    Manage images
init*    Creates Docker-related starter files for your project
manifest Manage Docker image manifests and manifest lists
network  Manage networks
plugin   Manage plugins
sbom*    View the packaged-based Software Bill Of Materials (SBOM) for an image
system   Manage Docker
trust    Manage trust on Docker images
```

Agora vamos **criar um container**, com o nome teste usando a imagem alpine:

```
C:\Users\jeffe>docker container create --name teste alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
9824c27679d3: Pull complete
Digest: sha256:4bcff63911fcb4448bd4fdacec207030997caf25e9bea4045fa6c8c44de311d1
Status: Downloaded newer image for alpine:latest
6c706736e3a903f39de678949f6d1b6c236e6f02b92a50d59d4a5befd56bd01c
```

Aqui estamos removendo nosso container:

```
C:\Users\jeffe>docker container rm teste
teste
```

### Seção 3 - Manipulando Docker Container através da Docker CLI:

Esse comando cria um container chamado **teste2** usando a imagem do Alpine Linux. Ele já deixa o terminal pronto pra usar com **-it** e o **shell sh**, mas o container ainda não começa a rodar — só fica criado, esperando pra ser iniciado depois.

```
C:\Users\jeffe>docker container create --name teste2 -it alpine sh
e8383c105b3074eaa6fbf8148738ba21b7d87f2f8bc4e26f581abf1b24c99dd9
```

Aqui estamos startando ele:

```
C:\Users\jeffe>docker container start teste2
teste2
```

Aqui estamos entrando dentro do container pelo fato que crie ele com um **terminal interativo**, depois que estou dentro dele estou executando o comando **ls** para **listar** os **diretórios** dentro da raiz

```
C:\Users\jeffe>docker container attach teste2
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ #
```

É falado dentro do curso que os containers são descartáveis, pois eles são criados a partir de imagens imutáveis.

Aqui estou criando um **Arquivo** dentro do meu diretório:

```
C:\Users\jeffe>docker container attach teste2
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # touch jefferson
/ # ls
bin    dev    etc    home   jefferson  media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ #
```

Também é mostrado que é possível sair do container e ainda deixar ele em execução, como criar pastas dentro do container sem estar dentro dele, e como copiar essas pastas tanto do meu sistema para o container como do container para o sistema.

**Ctrl + P, depois Ctrl + Q:** esse atalho desanexa o terminal do container, mantendo o container rodando em segundo plano.

**Aqui estão alguns comandos básicos do Docker:**

## Manipulando containers

Listando containers

```
$ docker container ls -a -s
```

Parando containers

```
$ docker container stop <container>
```

Inicializando containers

```
$ docker container start <container>
```

Copiando containers

```
$ docker container cp <src> <dst>
```

Removendo containers

```
$ docker container rm <nome_id>
```

Anexando-se a containers em execução

```
$ docker container attach <nome_id>
```

Inspeccionando containers

```
$ docker container inspect <nome_id>
```

Renomeando containers

```
$ docker container rename <nome_antigo> <novo_nome>
```

**Mapeamento de volumes:** Volumes são usados pra salvar os dados dos containers, mesmo depois que eles são parados. Na aula, ele mostra um exemplo onde cria pastas do host dentro do container, só pra demonstrar como funciona. Mas também falou que isso pode ser usado pra coisas mais importantes, como guardar dados de um banco ou de outras aplicações.

**O que é uma porta?** Uma porta é um canal de comunicação que permite que os serviços dentro de um contêiner troquem dados com o mundo externo.

```
C:\Users\jeffe>docker container run -d --name nginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
59e22667830b: Pull complete
140da4f89dcb: Pull complete
96e47e70491e: Pull complete
2ef442a3816e: Pull complete
4b1e45a9989f: Pull complete
1d9f51194194: Pull complete
f30ffbee4c54: Pull complete
Digest: sha256:84ec966e61a8c7846f509da7eb081c55c1d56817448728924a87ab32f12a72fb
Status: Downloaded newer image for nginx:latest
83e5ae5d4f284467bcede9b68e6501f79fa0c5a49db339142cd49cfedc90b423

C:\Users\jeffe>
```

Acima estou criando um container com a imagem nginx,

Uma **porta** permite que o serviço executado dentro de um container (como o servidor web Nginx) se comunique com o mundo externo. Para que esse serviço seja acessado fora do ambiente Docker, é necessário **mapear uma porta do seu computador (host)** para uma **porta interna do container**, usando a opção **-p** no momento da criação do container.

O comando completo fica assim:

```
docker container run -d -p 8080:80 --name meu_nginx nginx
```

**docker container run**: inicia a criação e execução de um novo container;

**-d**: executa o container em segundo plano;

**-p 8080:80**: mapeia a porta 8080 do seu computador (host) para a porta 80 do container

**--name meu\_nginx**: dá um nome personalizado para o container, neste caso meu\_nginx;

**nginx**: é a imagem usada (servidor web Nginx).

# Bem-vindo ao nginx!

Se você vir esta página, o servidor web nginx foi instalado com sucesso e Trabalhando. É necessária uma configuração adicional.

Para documentação e suporte on-line, consulte [nginx.org](http://nginx.org).  
O suporte comercial está disponível em [nginx.com](http://nginx.com).

*Obrigado por usar o nginx.*

Se não usar a opção **-p**, o serviço **não será acessível pelo navegador**, pois estará isolado dentro da rede Docker. Daí ele só poderá ser acessado por outros containers conectados à mesma rede.

## Seção 4 - Manipulando Imagens Docker através de CLI

```
C:\Users\jeffe>docker container run --name nginx-allumy -it alpine sh
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # mkdir allumy
/ # cd allumy
/allumy # touch docker
/allumy # ls
docker
/allumy # echo "TESTE CONTAINER PARA IMAGEM" > docker
/allumy # cat docker
TESTE CONTAINER PARA IMAGEM
/allumy #
```

Iniciei a criação de um container utilizando a imagem alpine, Dentro do container, naveguei pelo sistema de arquivos e criei um diretório chamado **allumy**, Em seguida, criei um arquivo chamado docker com o comando **touch**, e escrevi nele a seguinte frase: “TESTE CONTAINER PARA IMAGEM” e Verifiquei o conteúdo.

```
C:\Users\jeffe>docker container commit nginx-allumy nginx-allumy-img
sha256:cd78151b310deda01d4bf37b658b536408491510b0fa9818abbd10a4ddaafafa1
```

Aqui eu criei uma nova imagem Docker a partir do container **nginx-allumy** que estava rodando. Essa imagem guarda tudo o que tinha dentro do container naquele momento, inclusive mudanças que fiz. Dei o nome de **nginx-allumy-img** pra ela, e agora posso usar essa imagem pra criar outros containers iguais depois.

```
C:\Users\jeffe>docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx-allumy-img	latest	cd78151b310d	6 minutes ago	8.31MB
alpine	latest	9234e8fb04c4	2 weeks ago	8.31MB
nginx	latest	2cd1d97f893f	2 weeks ago	192MB
hive-base	latest	8d571bcb5565	5 months ago	1.07GB
hadoop-base	latest	73038d00484e	5 months ago	754MB
airflow-basic	latest	2b6d4516c10e	6 months ago	1.42GB
<none>	<none>	1ee7761cc316	6 months ago	1.42GB
hello-world	latest	74cc54e27dc4	6 months ago	10.1kB
python	3.9-slim	453d3342b002	7 months ago	126MB
python	3.8-slim	b5f62925bd0f	10 months ago	125MB
mysql	5.7	5107333e08a8	19 months ago	501MB
python	3.5-slim	6ffde5f0e2d0	4 years ago	110MB

```
C:\Users\jeffe>
```

Aqui rodamos o comando **docker image ls** para ver quais imagens temos. Podemos observar que a primeira imagem que aparece quando rodei o comando é a imagem que acabei de salvar.

```
C:\Users\jeffe\imagens>docker image save -o nginx-allumy.tar nginx-allumy-img
```

```
C:\Users\jeffe\imagens>
```

exporta a imagem **nginx-allumy-img** para um arquivo **.tar** chamado **nginx-allumy.tar**. Isso permite **salvar a imagem localmente como um arquivo**, que pode ser **compartilhado, versionado ou transferido para outro computador**.

```
C:\Users\jeffe\imagens>docker image load -i nginx-allumy.tar
```

```
2881b26d3114: Loading layer [=====] 4.096kB/4.096kB
```

```
Loaded image: nginx-allumy-img:latest
```

```
C:\Users\jeffe\imagens>
```

Esse comando serve pra importar uma imagem Docker que foi salva antes num arquivo **.tar**. Aqui, ele tá carregando a imagem **nginx-allumy-img** do arquivo **nginx-allumy.tar** e colocando de volta no Docker local. Isso é útil quando a gente quer usar a imagem em outra máquina ou quando tá sem internet.

```
C:\Users\jeffe\imagens>docker image history cd78151b310d
IMAGE          CREATED          CREATED BY          SIZE      COMMENT
cd78151b310d   2 hours ago     sh                  132B      buildkit.dockerfile.v0
<missing>      2 weeks ago     CMD ["/bin/sh"]     0B        buildkit.dockerfile.v0
<missing>      2 weeks ago     ADD alpine-minirootfs-3.22.1-x86_64.tar.gz /... 8.31MB    buildkit.dockerfile.v0

C:\Users\jeffe\imagens>
```

Também é possível ver o histórico da imagem, com o comando acima.

## Breve Resumo sobre os comandos:

1. **docker container run**: Cria e inicia um novo contêiner.
2. **docker container ls**: Lista os contêineres.
3. **docker container stop**: Para um ou mais contêineres em execução.
4. **docker container start**: Inicia um ou mais contêineres parados.
5. **docker container rm**: Remove um ou mais contêineres parados.

1. **docker image pull**: Este comando baixa uma imagem de um registro.
2. **docker image ls**: Lista as imagens locais.
3. **docker image build**: Constroi uma imagem a partir de um Dockerfile.
4. **docker image tag**: Cria um alias para uma imagem.
5. **docker image rm**: Remove uma ou mais imagens locais.

## Seção 5, 6, 7 -

Para criar imagens a partir de um dockerfile que está no mesmo diretório é possível usar esse comando - > **docker image build --no-cache -t meu-ubuntu** .

É possível ver como o Docker está e quanto ele está consumindo do seu disco, com esse comando:

```
C:\Users\jeffe>docker system df
TYPE          TOTAL          ACTIVE          SIZE           RECLAIMABLE
Images        12             3               3.427GB        3.427GB (99%)
Containers    4              1               1.39GB         1.39GB (99%)
Local Volumes 4              1               439MB          439MB (100%)
Build Cache   74             0               1.351GB        1.351GB

C:\Users\jeffe>
```



Para liberação de memória, existe o docker prune:

**\$ docker system prune (pede confirmação)**

**Irá deletar:**

- Todos os containers parados**
- Todos os volumes não utilizados**
- Todas as redes não utilizadas**
- Todas as imagens não utilizadas**

Também é possível fazer isso para cada tipo de objeto:

Limpar todas as imagens que não estão associadas a containers

```
$ docker image prune
```

Limpar todos os containers que não estão em execução

```
$ docker container prune
```

Limpar todos os volumes que não estão associados a containers

```
$ docker volume prune
```

Limpar todas as redes que não estão associadas a containers

```
$ docker network prune
```

## Seção 8 - Revisão e testando seu aprendizado

### Exercício

- Rodar em modo iterativo um container a partir da imagem do ubuntu, usando apt-get instalar o NGINX, inicializar o servidor NGINX.
- Com o comando commit, gerar uma imagem chamada ubuntu-allumy;
- Exportar a imagem para o arquivo ubuntu-allumy.tar através do comando save;
- Remover a imagem ubuntu-allumy;
- Importar a imagem com ubuntu-allumy.tar através do comando load.
- Rodar um container a partir da imagem importada.

ALL

**A)** Rodar em modo Interativo um container a partir da imagem Ubuntu

```
C:\Users\jeffe>docker run -it --name ubuntu-nginx-exercicio ubuntu sh
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
32f112e3802c: Pull complete
Digest: sha256:a08e551cb33850e4740772b38217fc1796a66da2506d312abe51acda354ff061
Status: Downloaded newer image for ubuntu:latest
# |
```

apt-get update para poder instalar o nginx

```
# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:2 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:3 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [23.0 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:5 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [1135 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [19.3 MB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1326 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [1975 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble/restricted amd64 Packages [117 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble/main amd64 Packages [1808 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [331 kB]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1449 kB]
Get:14 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1663 kB]
Get:15 http://archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [45.2 kB]
Get:16 http://archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [2086 kB]
Get:17 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [33.0 kB]
Get:18 http://archive.ubuntu.com/ubuntu noble-backports/main amd64 Packages [48.8 kB]
Fetched 32.0 MB in 1min 39s (324 kB/s)
Reading package lists... Done
# |
```

apt-get install nginx para instalar o nginx

```
# apt-get install nginx
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  iproute2 libatmlt64 libbpf1 libcap2-bin libelf1t64 libmnl0 libpam-cap libxtables12 nginx-common
Suggested packages:
  iproute2-doc python3:any fcgiwrap nginx-doc ssl-cert
The following NEW packages will be installed:
  iproute2 libatmlt64 libbpf1 libcap2-bin libelf1t64 libmnl0 libpam-cap libxtables12 nginx nginx-common
0 upgraded, 10 newly installed, 0 to remove and 3 not upgraded.
Need to get 2025 kB of archives.
After this operation, 5799 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libelf1t64 amd64 0.190-1.1ubuntu0.1 [57.8 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble/main amd64 libbpf1 amd64 1:1.3.0-2build2 [166 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 libmnl0 amd64 1.0.5-2build1 [12.3 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble/main amd64 libxtables12 amd64 1.8.10-3ubuntu2 [35.7 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libcap2-bin amd64 1:2.66-5ubuntu2.2 [34.2 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 iproute2 amd64 6.1.0-1ubuntu6.2 [1120 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble/main amd64 libatmlt64 amd64 1:2.5.1-5.1build1 [22.9 kB]
Get:8 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libpam-cap amd64 1:2.66-5ubuntu2.2 [12.5 kB]
Get:9 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 nginx-common all 1.24.0-2ubuntu7.4 [43.4 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 nginx amd64 1.24.0-2ubuntu7.4 [521 kB]
Fetched 2025 kB in 3s (729 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libelf1t64:amd64.
(Reading database ... 4381 files and directories currently installed.)
Preparing to unpack .../0-libelf1t64_0.190-1.1ubuntu0.1_amd64.deb ...
Unpacking libelf1t64:amd64 (0.190-1.1ubuntu0.1) ...
```

**B) Com o comando commit gerar uma imagem chamada Ubuntu-allumy**

```
C:\Users\jeffe>docker commit ubuntu-nginx-exercicio ubuntu-allumy
sha256:efb5bf164561fafb4d7e5d8fbe4da183988a9e5d79d7314bad5d510f3095cbda
C:\Users\jeffe>
```

**C) Exportar a imagem para o .Tar**

```
C:\Users\jeffe>docker save -o ubuntu-allumy.tar ubuntu-allumy
C:\Users\jeffe>
```

**D) Remover a imagem ubuntu-allumy:**

```
C:\Users\jeffe>docker image rm ubuntu-allumy
Untagged: ubuntu-allumy:latest
Deleted: sha256:efb5bf164561fafb4d7e5d8fbe4da183988a9e5d79d7314bad5d510f3095cbda
Deleted: sha256:ffbc718e53de3f89cfc791a703a88a30b72f3110784e93fcfe51fa08a0f9dfd2
C:\Users\jeffe>
```

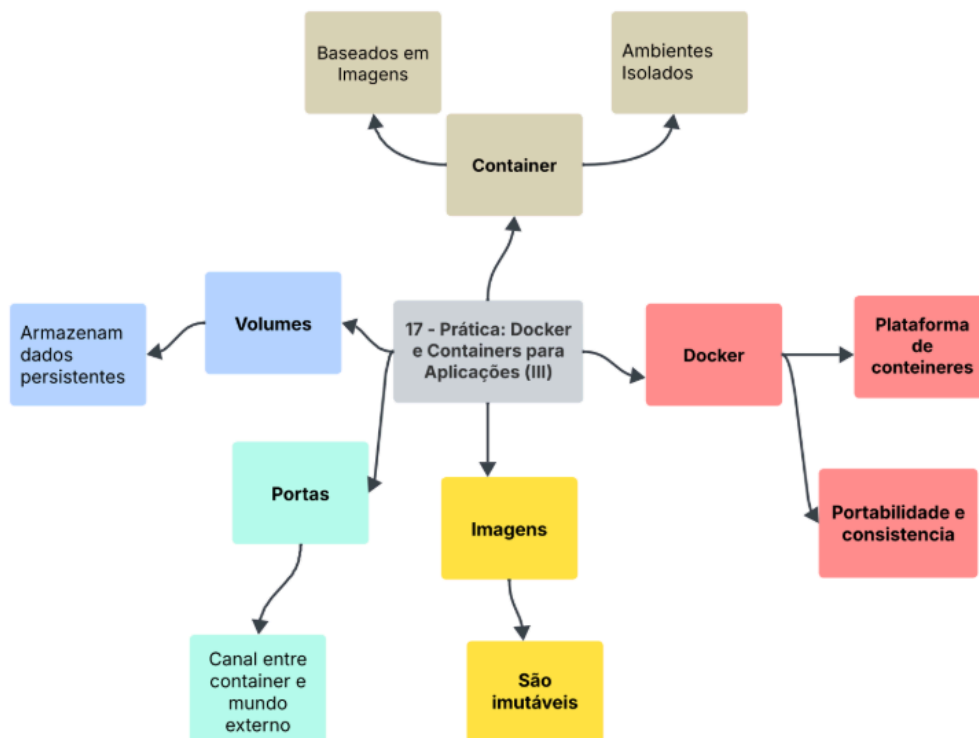
**E) Importar a imagem de volta com Docker Load:**

```
C:\Users\jeffe>docker load -i ubuntu-allumy.tar
768640c785d7: Loading layer [=====>] 58.52MB/58.52MB
Loaded image: ubuntu-allumy:latest
C:\Users\jeffe>
```

**F) Rodar um container a partir da imagem importada:**

```
C:\Users\jeffe>docker run -it --name testeExercicio ubuntu-allumy sh
# ls
bin          boot  etc   lib   media  opt   root  sbin          srv  tmp  var
bin.usr-is-merged  dev  home  lib64  mnt   proc  run   sbin.usr-is-merged  sys  usr
#
```

**Insight Visual Original:**



**Conclusão:** O estudo e as práticas realizadas neste card me proporcionaram uma base sólida sobre os conceitos fundamentais do Docker, como containers, imagens, volumes e mapeamento de portas. Aprendi a criar, manipular, exportar e importar contêineres e imagens, além de compreender como essas ferramentas se aplicam no dia a dia do desenvolvimento.

**Referências:**

Card 17 - Bootcamp - Lamia