# Relatório 16: Docker e Containers para Aplicações

**Davi Bezerra Barros**

**Container:** Um container é um ambiente isolado onde uma aplicação e todas as suas dependências são hospedadas, e podem ser executado em qualquer máquina que tenha um mecanismo para rodar container, como o Docker, sem precisar se preocupar com as especificidades do sistema operacional

**Imagem:** É o "template" que contém as instruções para a criação de um container, e é composta por múltiplas camadas empilhadas, cada uma especificando uma característica do ambiente. A imagem também contém os códigos binários, runtimes, dependências, filesytem e sistema operacional necessários para rodar uma aplicação quando instanciada em um container.

**Comandos de manupulação de containers**

- **Docker container ls:** Lista os containers
- **Docker container stop < container >:** Para container em execução
- **Docker container start** : inicializa o container
- **Docker container cp < src > < dst >:** copia o container
- **Docker container rm < nome_id >:** remove um container
- **Docker container attach < nome_id >:** se anexa ao container em execução
- **Docker container inspect < nome*id* >:** inspeciona as informações sobre o container
- **Docker container rename < nome_antigo> < nome_novo>:** Renomeia o container

**Filesystem**

```
PS C:\Users\davi.barros> docker container start -ia teste
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr
    var
/ # touch DAVI
/ # ls
DAVI   bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp
    usr    var
/ # echo "teste" > DAVI
/ # cat teste
cat: can't open 'teste': No such file or directory
/ # cat DAVI
teste
/ # exit
```

***Modo iterativo:*** O modo iterativo permite que o usuário acesse e interaja com um container enquanto ele está sendo executado, para fazer as mudanças que forem necessárias.

Criando containers iterativamente:

```
PS C:\Users\davi.barros> docker container ls -a
CONTAINER ID   IMAGE     COMMAND     CREATED              STATUS                          PORTS     NAMES
81f9e13af6c9   alpine    "/bin/sh"   12 seconds ago       Created                                   lucid_me
ninsky
d98a565c7470   alpine    "sh"        About a minute ago   Exited (0) About a minute ago             teste
PS C:\Users\davi.barros> docker container rm 0
Error response from daemon: No such container: 0
PS C:\Users\davi.barros> docker container rm d9
d9
PS C:\Users\davi.barros> docker container ls -a
CONTAINER ID   IMAGE     COMMAND     CREATED              STATUS    PORTS     NAMES
81f9e13af6c9   alpine    "/bin/sh"   About a minute ago   Created             lucid_meninsky
PS C:\Users\davi.barros> docker container rename 81f9 meucontainer
PS C:\Users\davi.barros> docker container ls -a
CONTAINER ID   IMAGE     COMMAND     CREATED              STATUS    PORTS     NAMES
81f9e13af6c9   alpine    "/bin/sh"   About a minute ago   Created             meucontainer
PS C:\Users\davi.barros>
```

***EXEC:*** O comando **Exec** permite que o usuário execute comandos no container ja em execução, sem a necessidade de entrar em seu ambiente.

Executando comandos com **exec**:

```
Mem: 882056K used, 3066520K free, 3220K shrd, 3692K buff, 301472K cached
CPU:   0% usr   0% sys   0% nic  99% idle   0% io   0% irq   0% sirq
Load average: 0.00 0.00 0.00 1/288 12
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    1     0 root     S     1724   0%   0   0% sh
    7     0 root     R     1624   0%   7   0% top
```

```
PS C:\Users\davi.barros> docker container exec teste cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      65b610a67600
PS C:\Users\davi.barros>
```

Copiando um diretório para o container:

```
PS C:\Users\davi.barros> mkdir Alluny


    Diretório: C:\Users\davi.barros


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        05/08/2024     15:55                Alluny
```

```
PS C:\Users\davi.barros\Alluny> cd
PS C:\Users\davi.barros\Alluny> cd..
PS C:\Users\davi.barros> docker container cp Alluny teste:/
Successfully copied 1.54kB to teste:/
PS C:\Users\davi.barros> docker container exec teste ls
Alluny
bin
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
PS C:\Users\davi.barros> docer container attach teste
```

Copiando do container para um diretório do sistema:

```
PS C:\Users\davi.barros\bkp> docker container cp teste:/Alluny/Davi.txt "$HOME\bkp\"
Successfully copied 2.05kB to C:\Users\davi.barros\bkp\
PS C:\Users\davi.barros\bkp> ls


    Diretório: C:\Users\davi.barros\bkp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        05/08/2024     16:16             14 Davi.txt


PS C:\Users\davi.barros\bkp>
```

**Mapeando volumes**

Ao deletar um container, todos os dados contidos nele são deletados também. Para evitar isso, os containers usam volumes para armazenar as informações e salvar seu estado, o que é muito útil para preservar logs, arquivos de configuração, datasets, etc..

```
PS C:\Users\davi.barros\bkp> docker container run -v C:\Users\davi.barros/bkp/:/bkp/ --name testevolume -it alpine sh
/ # ls
bin    dev    home   media  opt    root   sbin   sys    usr
bkp    etc    lib    mnt    proc   run    srv    tmp    var
/ # cd bkp
/bkp # ls
davi.txt
/bkp # cat davi.txt
♦♦teste teste
/bkp #
```

**Mapeamento de portas**

Entendendo o processo de mapeamento de portas para fazer deploy de aplicações. As portas do processo podem ser mapeadas para uma porta especifica do hospedeiro, permitindo o acesso de outros hosts da rede ao server.

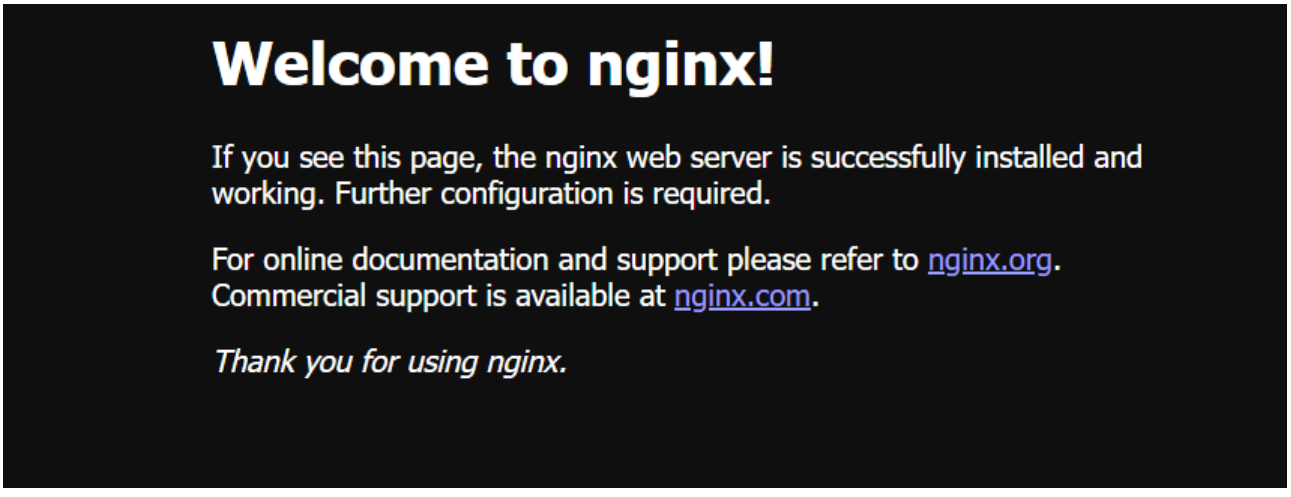
Criando um container com uma imagem do server Ngnix:

```
PS C:\Users\davi.barros> docker container run -v C:\Users\davi.barros\bkp/:/bkp/ --
name testevolume2 -it alpine sh
/ # ls
bin    dev    home   media  opt    root   sbin   sys    usr
bkp    etc    lib    mnt    proc   run    srv    tmp    var
/ # cd bpk
sh: cd: can't cd to bpk: No such file or directory
/ # dc bkp
dc: input error: Is a directory
/ # cd bkp
/bkp # ls
davi.txt
/bkp # exit
PS C:\Users\davi.barros> docker rm testevolume2
testevolume2
PS C:\Users\davi.barros> docker container run -d --name nginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
e4fff0779e6d: Pull complete
9c4e4e507b08: Pull complete
85664e7b25d6: Pull complete
e9d00ba15cc0: Pull complete
ca3474dd2ca6: Pull complete
472851929ad7: Pull complete
ed60582f2661: Pull complete
Digest: sha256:98f8ec75657d21b924fe4f69b6b9bff2f6550ea48838af479d8894a852000e40
Status: Downloaded newer image for nginx:latest
1ce3735025687076826l9ff6b0e2e10e6bb7bc7fd5afbb7c9b780d35e5db6e4a
PS C:\Users\davi.barros> docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PO
RTS     NAMES
1ce373502568   nginx     "/docker-entrypoint.…"   9 seconds ago   Up 8 seconds   80
/tcp    nginx
PS C:\Users\davi.barros> docker container inspect ngnix
```

Acessando o server a partir do ip fornecido pelo comando inspect:

```
PS C:\Users\davi.barros> docker container inspect nginx
```

```
                "Gateway": "172.17.0.1",
                "IPAddress": "172.17.0.2",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "DNSNames": null
            }
```




Criando um novo container mapeando a porta 80 do host na porta 80 do container:

```
docker container run -d -p 80:80 ngnix ngnix
```

```
            "SandboxKey": "/var/run/docker/netns/892b9e3785ee",
            "Ports": {
                "80/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "80"
                    }
                ]
```

**Gerando imagens a partir de containers**

Assim como volumes são utilizados para guardar os dados gerados por um container, imagens podem ser geradas para salvar seu estado. Isso facilita o processo de desenvolvimento ao permitir o compartilhamento e versionamento da aplicação. A criação de uma imagem base pode servir como ponto de partida para a criação de novos containers, como classes de objetos em programação orientada a objetos.

- Dockerfiles: Arquivo de texto que contém todas as instruções para construir uma imagem, define os comandos que o Docker irá executar sequencialmente para criar o ambiente de execução.

Gerando imagem a partir de um container:

```
PS C:\Users\davi.barros> docker container ls -a
CONTAINER ID   IMAGE     COMMAND   CREATED             STATUS                         PORTS     NAMES
2a7657357579   alpine    "sh"      About an hour ago   Exited (0) About an hour ago             nginx-allumy
PS C:\Users\davi.barros> docker container commit nginx-allumy nginx-allumy-img
sha256:6aaf65aa7d28035ada625269ace2a1a145ec80d2f86937025ae9f396c7b9ec3d
PS C:\Users\davi.barros> docker image ls
REPOSITORY         TAG       IMAGE ID       CREATED          SIZE
nginx-allumy-img   latest    6aaf65aa7d28   12 seconds ago   7.8MB
alpine             latest    324bc02ae123   3 weeks ago      7.8MB
nginx              latest    900dca2a61f5   7 weeks ago      188MB
hello-world        latest    d2c94e258dcb   15 months ago    13.3kB
PS C:\Users\davi.barros>
```

Criando um novo container a partir da imagem gerada:

```
PS C:\Users\davi.barros> docker container run -it --rm nginx-allumy-img sh
/ # ls
allumy  dev     home    media   opt     root    sbin    sys     usr
bin     etc     lib     mnt     proc    run     srv     tmp     var
/ # cat docker
cat: can't open 'docker': No such file or directory
/ # docker
sh: docker: not found
/ # cd allumy
/allumy # ls
docker
/allumy # cat docker
Teste container para imagem
/allumy #
```

- **Arquivo TAR:** *Tape ARchive*, agrupa diversos arquivos em um único arquivo, facilitando o backup e armazanamento. É o formato em que são exportadas as imagens utilizadas em containers.

Salvando a imagem como um arquivo TAR:

```
PS C:\Users\davi.barros\imagens> docker image save -o nginx-allumy.tar nginx-allum
y-img
PS C:\Users\davi.barros\imagens> ls


    Diretório: C:\Users\davi.barros\imagens


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        15/08/2024     15:16        8100352 nginx-allumy.tar
```

Carregando a imagem a partir de um arquivo .tar:

```
PS C:\Users\davi.barros\imagens> docker image load -i nginx-allumy.tar
dc241c3190f5: Loading layer  4.096kB/4.096kB
Loaded image: nginx-allumy-img:latest
PS C:\Users\davi.barros\imagens> docker image ls
REPOSITORY         TAG       IMAGE ID       CREATED          SIZE
nginx-allumy-img   latest    6aaf65aa7d28   32 minutes ago   7.8MB
alpine             latest    324bc02ae123   3 weeks ago      7.8MB
nginx              latest    900dca2a61f5   7 weeks ago      188MB
hello-world        latest    d2c94e258dcb   15 months ago    13.3kB
```

***Docker image history***: Exibe o historico de uma imagem do docker, trazendo informações sobre sua criação, alterações, tamanho, camadas que a compõe e outras informações relevantes.

```
PS C:\Users\davi.barros\imagens> docker image history nginx-allumy-img
IMAGE          CREATED             CREATED BY
 SIZE      COMMENT
6aaf65aa7d28   About an hour ago   sh
 131B
<missing>      3 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/sh"]
 0B
<missing>      3 weeks ago         /bin/sh -c #(nop) ADD file:99093095d62d04215…
 7.8MB
```

***Docker image import***: Utilizado para criar uma imagem docker a partir de um arquivo .tar, reaproveitando um sistema de arquivos como base para a nova imagem.

```
PS C:\Users\davi.barros\imagens> cd..
PS C:\Users\davi.barros> docker image import teste-export.tar
open teste-export.tar: The system cannot find the file specified.
PS C:\Users\davi.barros> cd imagens
PS C:\Users\davi.barros\imagens> docker image import teste-export.tar
sha256:ec272e319cc14bf8b968e1c2743482d30cfba40368eb333a69e6d6fa49ff1276
PS C:\Users\davi.barros\imagens> docker image import teste-export.tar ubuntu-impor
tado
sha256:812ceb2bf9a5cb24948436a8bff976e3d7b44ab2e748bdb51f65cce9c0631f69
PS C:\Users\davi.barros\imagens> docker image ls ubuntu-importado
REPOSITORY         TAG       IMAGE ID       CREATED         SIZE
ubuntu-importado   latest    812ceb2bf9a5   7 seconds ago   78.1MB
PS C:\Users\davi.barros\imagens> docker container run -it ubuntu-importado sh
# ls
allumy  boot  etc   lib    media  opt   root  sbin  sys  usr
bin     dev   home  lib64  mnt    proc  run   srv   tmp  var
# exit
PS C:\Users\davi.barros\imagens>
```

Outros comandos úteis para gerenciamento de imagens:

- ***Docker image inspect:*** Usado para obter informaçoes detalhadas sobre a imagem
- ***Docker image prune:*** Limpa imagens não utilizadas
- ***Docker image pull:*** Baixa uma imagem de um registro
- ***Docker image tag:*** Cria um novo nome para uma imagem existente
- ***Docker image push:*** Envia uma magem para um registro

***Comandos utilizados para gerenciar imagens de containers:***

- **build** - Constrói uma imagem a partir de um Dockerfile.
- **history** - Mostra o histórico de uma imagem.
- **import** - Importa o conteúdo de um arquivo tar para criar uma imagem de sistema de arquivos.
- **inspect** - Exibe informações detalhadas sobre uma ou mais imagens.
- **load** - Carrega uma imagem a partir de um arquivo tar ou da entrada padrão (STDIN).
- **ls** - Lista as imagens.
- **prune** - Remove imagens não utilizadas.
- **pull** - Baixa (puxa) uma imagem ou repositório de um registro.
- **push** - Envia (empurra) uma imagem ou repositório para um registro.
- **rm** - Remove uma ou mais imagens.

- **save** - Salva uma ou mais imagens em um arquivo tar (streaming para STDOUT por padrão).
- **tag** - Cria uma tag (etiqueta) TARGET_IMAGE que se refere à SOURCE_IMAGE.

***Comando utilizados para gerenciamento do ambiente de execução dos containers (Docker engine):***

- **Docker system info**: Informaçoes sobre o docker, containers e sobre a máquina hospedeira
- **Docker system prune:** Deleta todos os containers, imagens e redes não utilizadas.
- **Docker image prune:** Remove todas as imagens nao associadas a containers
- **Docker container prune:** Limpa todos os containers não utilizados
- **Docker network prune:** Remove todas as redes não utilizadas