

# Acessando "de dentro" e "de fora"

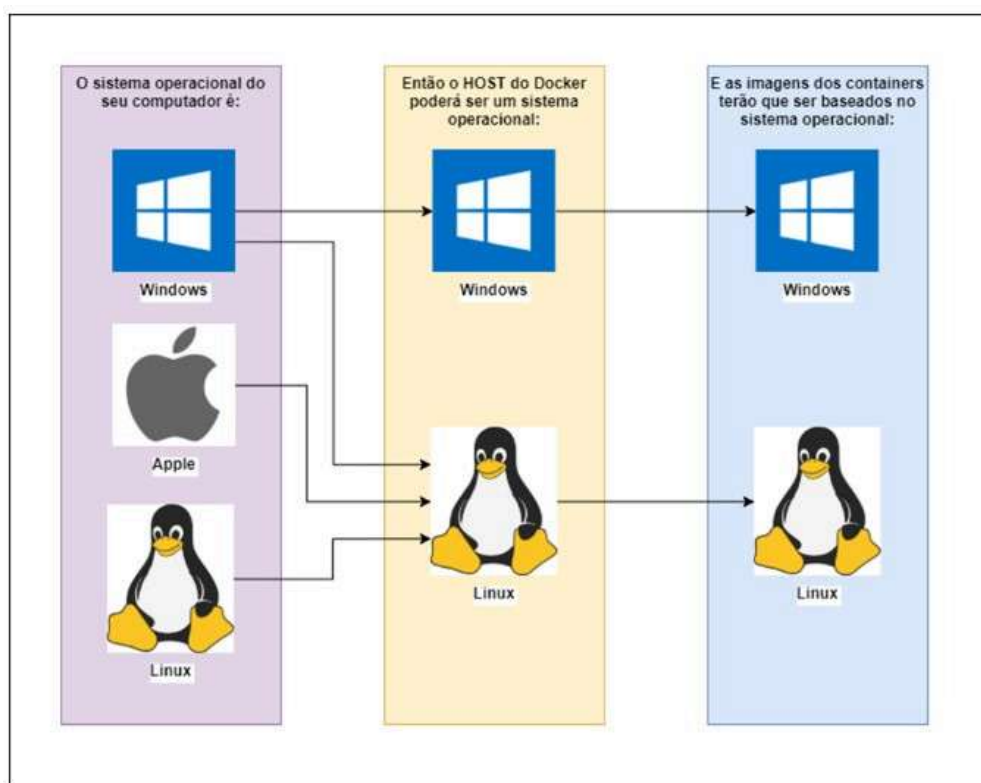
**C**omo vimos anteriormente, há um conceito em relação ao “host”, isto é, a base que sustenta tudo em relação ao funcionamento dos containers e os containers em si que “rodam em cima” do “host”. Os containers usam CPU, memória e disco do “host” como se pegassem um pedacinho emprestado o suficiente para que o container se sustente.

Falando um pouco de comunicação, isso também se aplica à interface de redes, ou seja, se um container precisar “sair para a internet”, o container usa no final do dia a placa de rede do “host” para fazer essa conectividade, tudo gerenciado pelo Docker.

Podemos então pensar assim: todo container, independente da imagem base que a serve (um Ubuntu Linux para rodar um servidor web como o Nginx, ou uma imagem base com Java para rodar aplicações Java, que inclui a Java Virtual Machine, entre outras imagens base), é necessário separar como isso funciona de fato: o “host” não necessariamente é o sistema operacional do seu computador!

Se já percebeu aqui, foram indicados alguns exemplos de uso do Docker no Windows. Estamos usando o WSL (Windows Subsystem for Linux) em cima do Windows, isso possibilita usarmos imagens baseados em Linux para subir os containers, bem “debaixo do nariz” do Windows!

Não vamos esquecer um outro ator importante nisso tudo quando se fala em sistema operacional: o Mac da Apple. O Mac não é nem Linux, nem Windows, mas basicamente quando subimos o Docker nele, também usamos um Linux “por trás da cortina” para subir nossos containers, há algumas diferenças na abordagem do Mac em relação ao Windows, vamos detalhar um pouco:



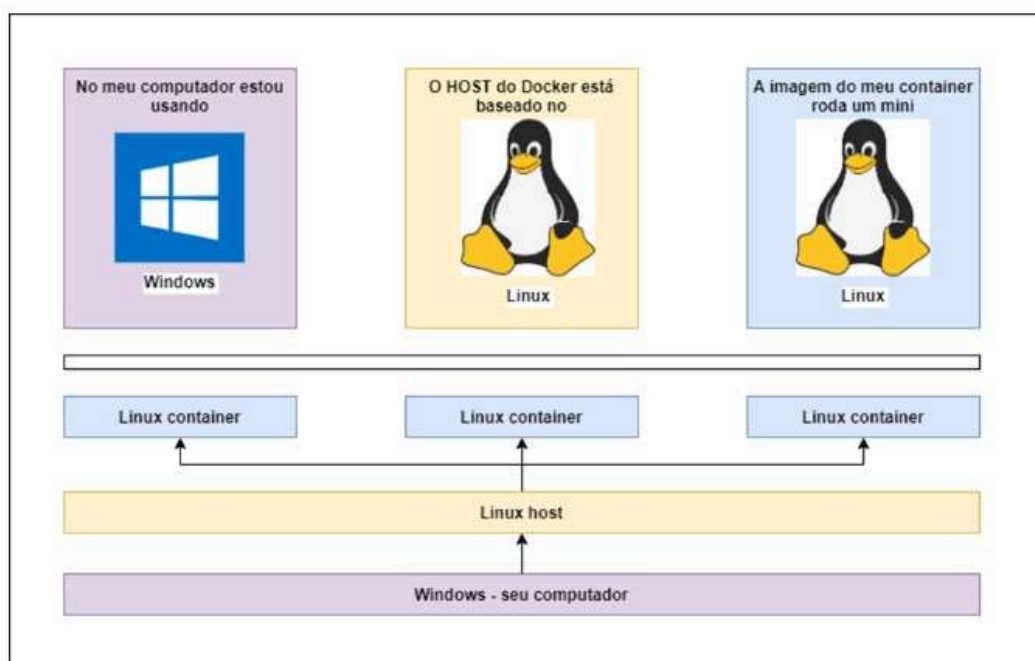
Veja que no caso do Windows, pode haver sim um “host” do Windows (geralmente utilizando uma ferramenta de virtualização como o Hyper-V), só que as imagens dos containers terão que ser obrigatoriamente Windows. Isso faz com que estes containers só rodem também no Docker cujo host seja baseado em Windows, o que limita bastante as soluções, mas pode ser que hajam sim alguns requisitos para entrarmos nesta possibilidade.

No caso do MacOS da Apple, você será obrigado a ter um “host” baseado em Linux, então não há esse problema de poder portar a solução para qualquer sistema operacional, o único requisito é que tenha um “host” Linux.

Das 3 opções (Windows, MacOS da Apple e as várias distribuições do Linux), todas podem ter “host” baseado em sistema operacional Linux, por consequência as imagens terão que ser baseadas em Linux também, só que a grande vantagem é que você consegue portar a solução para qualquer sistema operacional, desde que o Docker use como “host” o Linux. Por esta razão é preferível ter e construir

imagens baseadas em Linux, você conseguirá utilizar praticamente em qualquer lugar.

Ok, agora que sabemos isso, vamos adentrar um pouco mais em relação ao sistema operacional. Revisitando um tema importante sobre a base das imagens que construímos, vimos que é possível construirmos uma imagem com a base em Ubuntu, por exemplo. Isso quer dizer que é como se o container tivesse um “mini sistema operacional”, vou exemplificar com um desenho ilustrativo para entendermos melhor:



Então para o Docker, o “host” Linux vai gerenciando vários mini computadores, que são os containers em Linux também, uma das coisas que este gerenciamento traz é a comunicação entre o ambiente do Windows que está rodando o Docker e os containers Linux que estão debaixo do “host”.

Mas como conseguir chamar à partir do nosso Windows um container “X” por exemplo e como saber chamar um outro container “Y”, se está tudo dentro do mesmo “host”?

A resposta é: através do bind de porta.

Eita, mas o que é bind de porta? Que treco técnico!

Vamos lá:

Imagine o lugar onde você mora, tem um monte de casas, e um carteiro precisa chegar até as casas certas para entregar as cartas endereçadas a cada pessoa. Vamos dizer que hajam 3 casas numa mesma rua, o carteiro sabe a casa certa pelo número de cada casa, que não se repetem, cada casa tem seu número individual. Assim funcionam com as portas! O “bind” é pegar o container que representaria uma “casa” e dar um “número” (ou seja, a porta) a ela!

Quem for mais técnico deve estar me xingando agora, pois não é tão simples assim, mas por hora é a maneira mais simples de fazer uma analogia para que todos entendam!

É que na prática, você pode atribuir mais de uma porta a um mesmo container, então na prática é com se uma mesma casa pudesse ter mais de um número, mesmo assim o importante é que nesse “bind”, não existem números repetidos! Uma “casa” pode até ter uns 5 números atribuídos a ela (todos diferentes entre si), mas as demais “casas” nunca poderão ter nenhum desses 5 números!

Vamos ver isso na prática, abaixo está um comando para criar um container baseado em Linux com o Nginx, que é um servidor web. O meu Docker aqui usa o WSL (Windows Subsystem for Linux) como “host” do ambiente Docker, por isso eu consigo executar containers baseados em Linux:

```
docker run --name NginxSemBindDePorta -d nginx
```

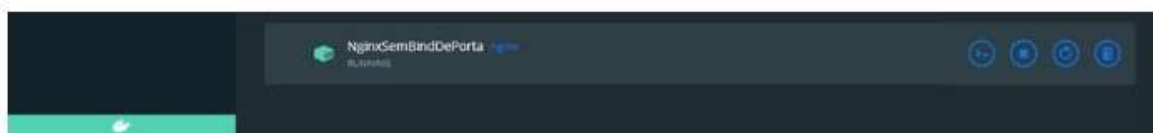
```
Selecionar Prompt de Comando
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\event>docker run --name NginxSemBindDePorta -d nginx
23edae0041f295b5011cd9906f24ab5025d891e3312c17b7060c884380f7a53

C:\Users\event>docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                    NAMES
23edae0041f2   nginx     "/docker-entrypoint. ..." 10 seconds ago Up 3 seconds   80/tcp                NginxSemBindDePorta
30b5fc7df0ba   pythonapi:latest   "uvicorn app.main:ap ..." 2 days ago    Up 2 hours    0.0.0.0:8000->8000/tcp   MeuPythonAPI
04daa76caaf   springbootapi:latest   "java -jar /usr/loca ..." 2 days ago    Up 2 hours    0.0.0.0:8080->8080/tcp   MeuSpringBootAPI
386565c98dd7   mysql:latest        "docker-entrypoint.s ..." 7 days ago    Up 2 hours    0.0.0.0:3306->3306/tcp, 33060/tcp   MeuMySQL
0dad8ee4cd75   oracle/database:18.4.0-xe   "/bin/sh -c 'exec $O ..." 7 days ago    Up 2 hours (healthy) 0.0.0.0:1521->1521/tcp, 0.0.0.0:5500->5500/tcp   MeuOracle

C:\Users\event>
```

Veja que o container foi criado com uma porta, neste caso é a porta 80. Isso seria como se eu estivesse dentro da minha casa e soubesse que o meu número de porta é 80, mas eu não coloquei esse número fora da minha casa, então o carteiro nunca conseguiria chegar na minha casa.



Vamos pegar uma ajudinha no Docker Desktop já que estamos no Windows, se você olharem bem para o nosso container “NginxSemBindDePorta”, tem 4 botões à direita, mas nenhuma delas é para acessar o servidor web do Nginx neste container, isso acontece porque não fizemos o “bind de porta”, ou seja, o container não está acessível pelo Windows.

Agora vamos criar outro container com Nginx também, só que com um bind de porta:

```
docker run --name NginxComBindDePorta -p 9280:80 -d nginx
```

```
Selecionar Prompt de Comando
C:\Users\levert>docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
23edae8841f2   nginx         "/docker-entrypoint..." 10 seconds ago Up 3 seconds   80/tcp                  NgInxSew
b1ndDePorta   pythonapi:latest "uvicorn app.main:ap..." 2 days ago    Up 2 hours    0.0.0.0:8000->8000/tcp   MeuPytho
18b5fc7dfbba   API          "java -jar /usr/loca..." 2 days ago    Up 2 hours    0.0.0.0:8000->8000/tcp   MeuSprin
8BootAPI
266565c98dd7   mysql:latest   "docker-entrypoint.s..." 7 days ago    Up 2 hours    0.0.0.0:3306->3306/tcp; 33060/tcp   MeuMySQL
ddad88e4cd75   oracle/database:18.4.0-xe "/bin/sh -c 'exec $@..." 7 days ago    Up 2 hours (healthy) 0.0.0.0:1521->1521/tcp, 0.0.0.0:5500->5500/tcp   MeuOrac1
e

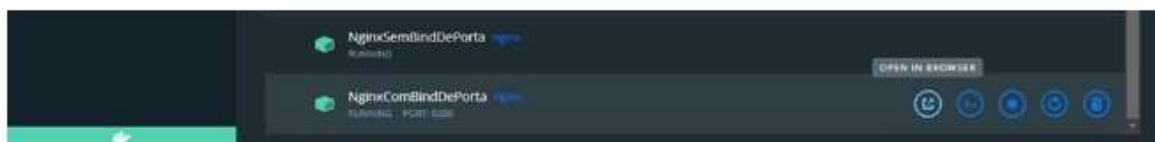
C:\Users\levert>docker run --name NgInxComBindDePorta -p 9280:80 -d nginx
ba7a553d6d8c3d79e42121d032c1043931250d1d1f7a04334774a7e10c8344

C:\Users\levert>docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
ba7a553d6d8c   nginx         "/docker-entrypoint..." 8 seconds ago  Up 2 seconds   0.0.0.0:9280->80/tcp     NgInxCom
23edae8841f2   nginx         "/docker-entrypoint..." 6 minutes ago  Up 6 minutes    80/tcp                  NgInxSew
b1ndDePorta   pythonapi:latest "uvicorn app.main:ap..." 2 days ago    Up 2 hours    0.0.0.0:8000->8000/tcp   MeuPython
18b5fc7dfbba   API          "java -jar /usr/loca..." 2 days ago    Up 2 hours    0.0.0.0:8000->8000/tcp   MeuSpring
8BootAPI
266565c98dd7   mysql:latest   "docker-entrypoint.s..." 7 days ago    Up 2 hours    0.0.0.0:3306->3306/tcp; 33060/tcp   MeuMySQL
ddad88e4cd75   oracle/database:18.4.0-xe "/bin/sh -c 'exec $@..." 7 days ago    Up 2 hours (healthy) 0.0.0.0:1521->1521/tcp, 0.0.0.0:5500->5500/tcp   MeuOracle

C:\Users\levert>
```

Veja que agora temos 2 coisas: o 80/tcp que já vem com o Nginx que é mais ou menos como eu estou em casa e sei o número da minha casa, já o 0.0.0.0:9280 é o tal bind de porta, significa que é como se eu colocasse um número na porta da minha casa para o carteiro encontrá-la.

Aqui vem dois detalhes interessantes: no bind de porta, o número para o “carteiro” encontrar a casa (no caso 9280) não precisa ser exatamente igual ao número interno do container (que é 80), não tem problema pois é feito um tipo de “depara”, agora olhemos o Docker Desktop para ver a diferença:



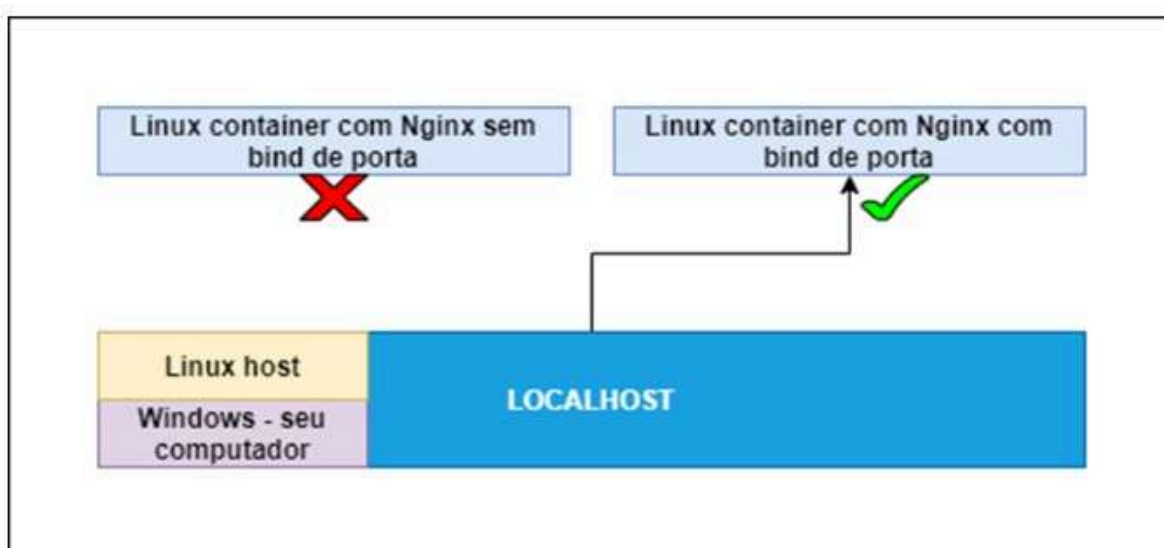
Veja que agora apareceu um quinto botão, passando o mouse por cima dele fica escrito “Open in Browser” que significa “abrir no navegador”. Ao clicar neste botão o Google Chrome (que aqui é meu navegador padrão) abre e sou direcionado para o seguinte endereço: localhost:9280.



“O carteiro conseguiu chegar na casa!”, mas peraê, o que é o tal do localhost e porque não ficou 0.0.0.0:9280?

Vamos lá: pegando um pouco de conhecimento de redes mas não adentrando muito (vai um curso só disso para explicar tudo), quando indicamos 0.0.0.0 significa pode vir requisição de qualquer lugar. O qualquer lugar inclui o localhost.

O localhost por sua vez é um apelido que damos ao nosso computador local, o que tem o Windows (ou o Linux, ou o MacOS se for o seu caso), é um apelido comumente utilizado em sistemas operacionais para indicar que estamos aqui mesmo, neste local do host. Ilustrando isso:

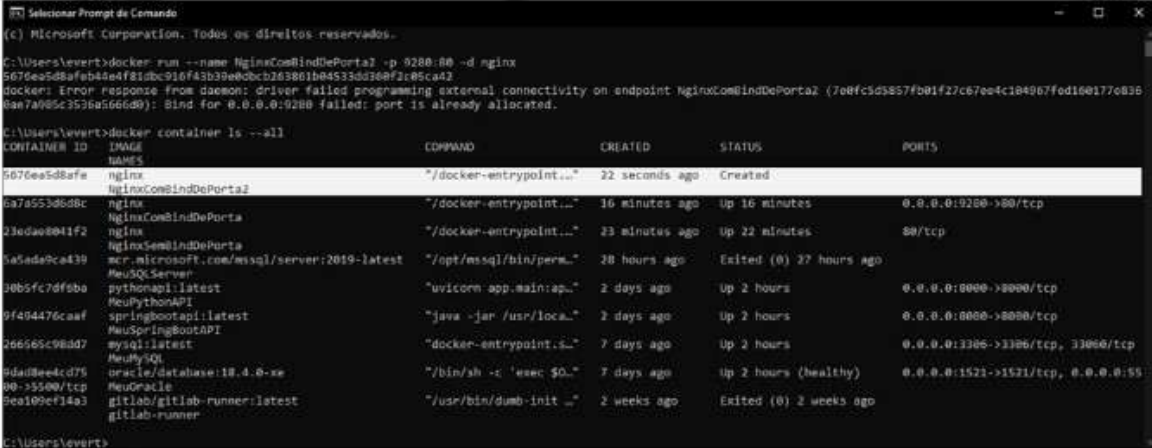


O localhost no final do dia é o teu computador, rodando o Windows como sistema operacional principal e tendo dentro dele o Docker rodando com o “host” em Linux. Ele consegue chegar no container com o bind de porta pois esse bind permite a

entrada através de qualquer lugar (0.0.0.0, ou seja, incluindo seu localhost) traduzindo a porta 9280 para a porta 80 interna do container.

Agora vamos tentar subir mais um container com Nginx, repetindo a porta 9280 pra ver o que acontece.

```
docker run --name NginxComBindDePorta2 -p 9280:80 -d nginx
```



```
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\event>docker run --name NginxComBindDePorta2 -p 9280:80 -d nginx
5676ee5d8afeb44ef81dbc916f43b39e0bcb263861b04533d368f2c09ca42
docker: Error response from daemon: driver failed programming external connectivity on endpoint NginxComBindDePorta2 (70efcd3857fb01f27c67ee4c104067f0d10e1770836
baa7a985c3526a566d0): Bind for 0.0.0.0:9280 failed: port is already allocated.

C:\Users\event>docker container ls --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
5676ee5d8afe	nginx	"/docker-entrypoint..."	22 seconds ago	Created	
ea7a553b0db0c	nginx	"/docker-entrypoint..."	16 minutes ago	Up 16 minutes	0.0.0.0:9280->80/tcp
23edae0041f2	nginx	"/docker-entrypoint..."	23 minutes ago	Up 22 minutes	80/tcp
6a5ada9ca439	mcr.microsoft.com/mssql/server:2019-latest	"/opt/mssql/bin/per..."	28 hours ago	Exited (0) 27 hours ago	
30b5fc7df0ba	pythonapi:latest	"uvicorn app.main:ap..."	2 days ago	Up 2 hours	0.0.0.0:8000->8000/tcp
9f494476caaf	springbootapi:latest	"java -jar /usr/loca..."	2 days ago	Up 2 hours	0.0.0.0:8000->8000/tcp
26656c98ad7	mysql:latest	"docker-entrypoint.s..."	7 days ago	Up 2 hours	0.0.0.0:3306->3306/tcp, 33060/tcp
6dad8e4cd75	oracledatabase:18.4.0-xe	"/bin/sh -c 'exec \$@"	7 days ago	Up 2 hours (healthy)	0.0.0.0:1521->1521/tcp, 0.0.0.0:55
00-55500/tcp	NeuOracle				
6ea109ef14a3	gitlab/gitlab-runner:latest	"/usr/bin/dumb-init ..."	2 weeks ago	Exited (0) 2 weeks ago	
	gitlab-runner				

```
C:\Users\event>
```

Veja que deu erro, ele até criou o container, mas não conseguiu subir, a coluna “STATUS” está como “Created”, se tivesse subido esta coluna estaria como “Up”. Isso porque já tem um container ativo usando a porta 9280, então vimos na prática que não tem como subir 2 containers e deixá-los ativos usando a mesma porta. Eu até poderia parar o container que está usando a porta 9280 e subir essa, mas se eu tentar subir as 2 com a mesma porta vai dar pau!

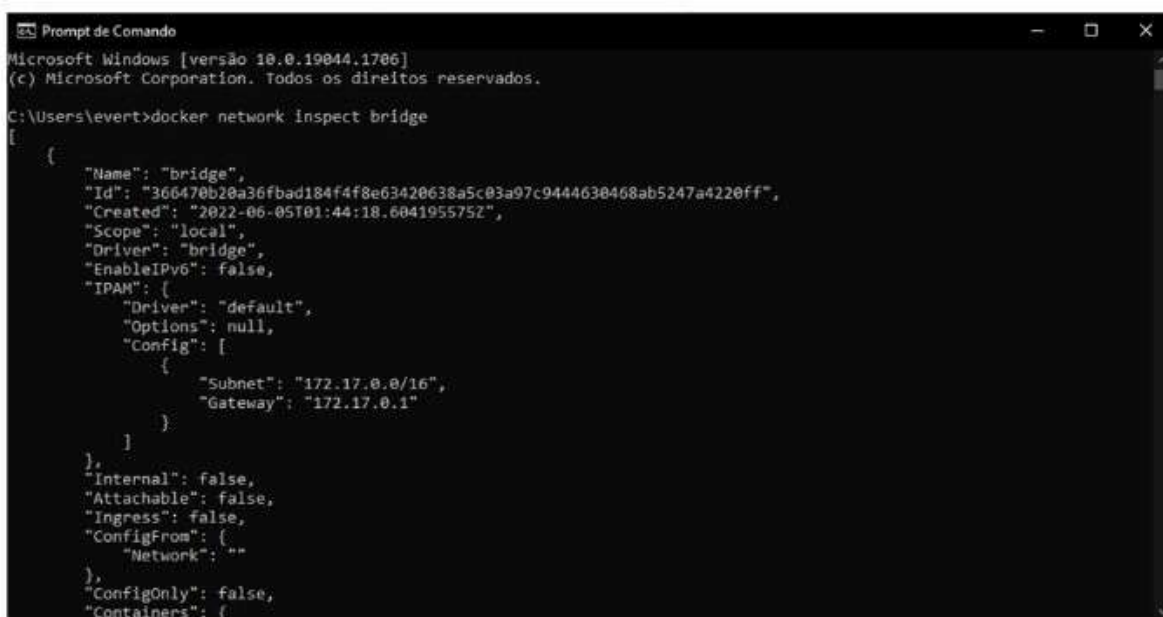
Então a regra, para ficar mais claro é: se tiver bind de porta, uma mesma porta não pode ser usada por 2 ou mais containers ativos ao mesmo tempo.

Não quero deixar esse assunto muito complexo, mas ainda há mais uma coisa para olharmos: será que os containers conseguem falar com eles mesmos? Uma coisa é fazer a conexão partindo do host com o bind de porta, mas será que um container fala com o outro? Vamos descobrir:



Primeiramente, uma coisa que não comentei é que ao criar esses containers eu não indiquei a “sub-rede” à qual ambos os containers pertencem. O padrão é pertencerem à uma “sub-rede” chamada “bridge”, que existe por padrão no Docker, se você não diz nada então o container “cai” nessa “sub-rede”, vamos olhar essa “sub-rede” então com o seguinte comando:

docker network inspect bridge



```
Prompt de Comando
Microsoft Windows [versão 10.0.19044.1706]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\event>docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "366470b20a36fbad184f4f8e63420638a5c03a97c9444630468ab5247a4220ff",
    "Created": "2022-06-05T01:44:18.604195575Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
```



```
    "23edae8041f295b5011cd99b6f2dab5025d891e3312c17b7060c8884308f7a53": {
      "Name": "NginxSemBindDePorta",
      "EndpointID": "091665b1f399bfc429bbbaac5c2bdacc595816cfc6dc713dccb03bb2c0c2e7db",
      "MacAddress": "02:42:ac:11:00:04",
      "IPv4Address": "172.17.0.4/16",
      "IPv6Address": ""
    },
    "6a7a553d6d8c3d79e42121d032c16439312b0d1d1df17a64334774a7e10c8344": {
      "Name": "NginxComBindDePorta",
      "EndpointID": "3ba21f01e0616fa2db5185f1a7c5cf4711fd3c4eb7c610e25ace34ab1fffd3ea",
      "MacAddress": "02:42:ac:11:00:05",
      "IPv4Address": "172.17.0.5/16",
      "IPv6Address": ""
    }
  },
  "Options": {
    "com.docker.network.bridge.default_bridge": "true",
    "com.docker.network.bridge.enable_icc": "true",
    "com.docker.network.bridge.enable_ip_masquerade": "true",
    "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
    "com.docker.network.bridge.name": "docker0",
    "com.docker.network.driver.mtu": "1500"
  },
  "Labels": {}
}
```

Vemos que os 2 containers que criamos, “NginxSemBindDePorta” e o “NginxComBindDePorta” estão nessa “sub-rede”, vamos usar o comando para entrar em uma delas para chamar a outra, repare no endereço IP de ambas, sendo respectivamente 172.17.0.4 e 172.17.0.5:

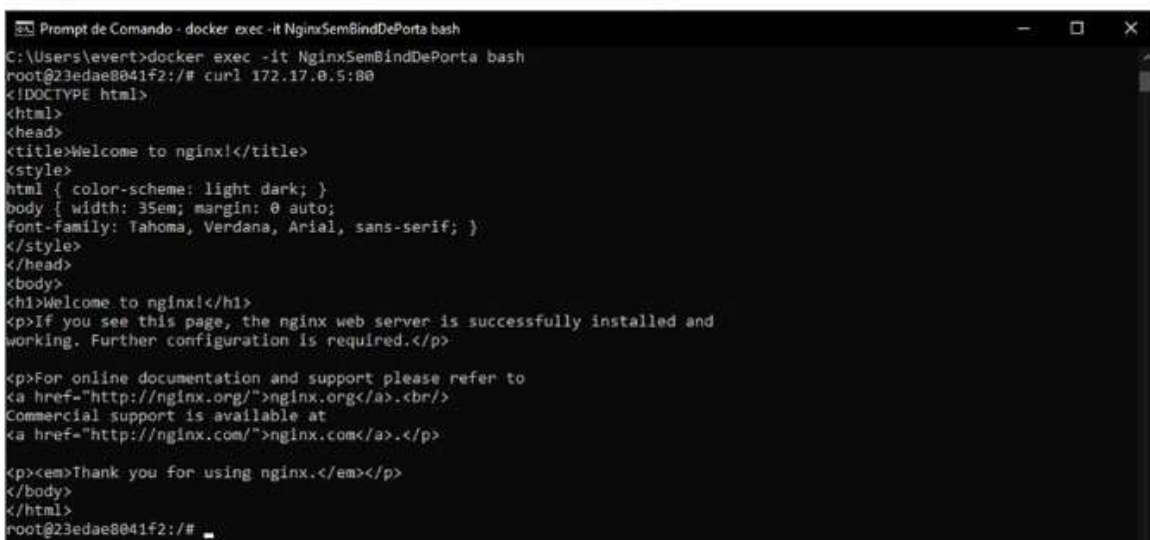
```
docker exec -it NginxSemBindDePorta bash
```

Isso fará com que “entremos” no container “NginxSemBindDePorta” usando um terminal “bash”, o comando “docker exec” roda comandos em um container ativo e o parâmetro “-it” é para rodar no modo interativo, ou seja, usar o terminal no próprio Prompt de Comando do Windows.

Logo que entrar, vamos tentar chamar o Nginx do outro container, o “NginxComBindDePorta”. Lembram que era para lembrar do endereço IP? Agora vai ser muito útil, vamos usar um comando chamado “curl” para fazer uma chamada como se fosse um Google Chrome chamando o site, o comando é esse:

```
curl 172.17.0.5:80
```

Aqui eu tenho que indicar o IP do outro container para dizer aonde eu quero ir e o número da porta que é a porta padrão do Nginx dentro do container. O resultado é este:



```
Prompt de Comando - docker exec -it NginxSemBindDePorta bash
C:\Users\event>docker exec -it NginxSemBindDePorta bash
root@23edae8041f2:/# curl 172.17.0.5:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@23edae8041f2:/#
```

É claro que como é um terminal, não tem como ver o site com aquele visual mais bonito, mas o que podemos ver aqui é que fomos respondidos. O que retornou foi o código HTML do servidor web, provavelmente do arquivo index.html que vem por padrão com a imagem do Nginx.

Agora vamos sair deste container e entrar no outro e fazer a mesma coisa, só que invertendo a chamada, primeiro eu saio do container com o seguinte comando:

```
exit
```

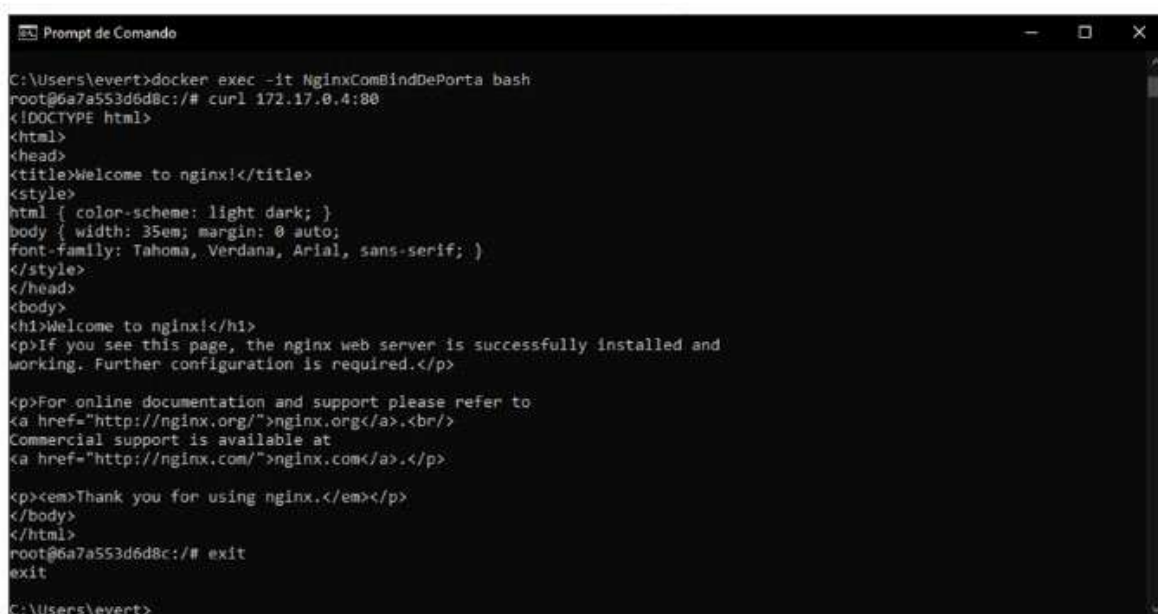
Depois eu entro no outro container com o seguinte comando:

```
docker exec -it NginxComBindDePorta bash
```

E de dentro deste container eu faço a chamada com o “curl” para o container “NginxSemBindDePorta”:

```
curl 172.17.0.4:80
```

O resultado da chamada ao container que não tem o bind:

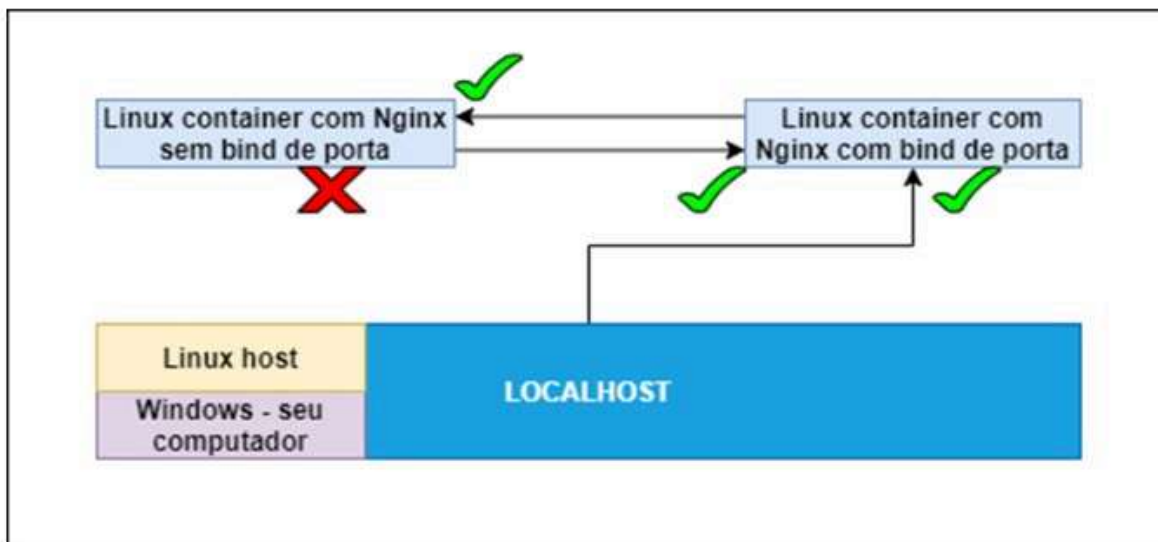


```
Prompt de Comando
C:\Users\event>docker exec -it NginxComBindDePorta bash
root@6a7a553d6d8c:/# curl 172.17.0.4:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@6a7a553d6d8c:/# exit
exit
C:\Users\event>
```

Deu certo também!!! Vou ilustrar como toda essa comunicação funcionou:



Isso só foi possível pois ambos os containers estão na mesma “sub-rede”, então vamos fazer um último teste, criando um container também com Nginx só que em outra “sub-rede” que eu tenho aqui, chamado “MinhaRede”:

`docker run --name NginxComBindDePorta3 --network MinhaRede -p 9380:80 -d nginx`

```

C:\Users\event>docker run --name NginxComBindDePorta3 --network MinhaRede -p 9380:80 -d nginx
6395cab0ad5419ab09916840e01437610d6cb70e103607a73913939bc8e7ef

C:\Users\event>docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS
NAMES
6395cab0ad5    nginx     "/docker-entrypoint..." 17 seconds ago Up 10 seconds 0.0.0.0:9380->80/tcp
NginxComBindDePorta3
6a7a553d8dc    nginx     "/docker-entrypoint..." 38 minutes ago Up 38 minutes 0.0.0.0:9280->80/tcp
NginxComBindDePorta
23edae8041f2   nginx     "/docker-entrypoint..." 45 minutes ago Up 45 minutes 80/tcp
NginxSemBindDePorta
38b5fc7df6ba   pythonapi:latest "uvicorn app.main:ap..." 2 days ago    Up 3 hours    0.0.0.0:8000->8000/tcp
MeuPythonAPI
9f494476caaf   springbootapi:latest "java -jar /usr/loc..." 2 days ago    Up 3 hours    0.0.0.0:8000->8000/tcp
MeuSpringBootAPI
206565c8dd07   mysql:latest "docker-entrypoint.s..." 7 days ago    Up 3 hours    0.0.0.0:3306->3306/tcp, 33060/tcp
MeuMySQL
9dad8eeacd75   oracle/database:18.4.0-xe "/bin/sh -c 'exec $O..." 7 days ago    Up 3 hours (healthy) 0.0.0.0:1521->1521/tcp, 0.0.0.0:5500->5500/tcp
MeuOracle
  
```

```
Prompt de Comando
C:\Users\levert>docker network inspect MinhaRede
[
  {
    "Name": "MinhaRede",
    "Id": "3c2529ba8bef3b0c75f730100d32f8db0b84b7f02df68b6b3251dd8bc4db6e6",
    "Created": "2022-05-29T19:00:04.020744494Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "26e565c9dd7f9e81840d380b9f9c097f27d6a73f8b7a9c3938ca64c370648a": {
        "Name": "NewMySQL",
        "EndpointID": "e161c9cd588789e7ef83caf1a5efb6c93e14d84f064013c5f47ab2dbfce707",

```

```
Selecionar Prompt de Comando
1.
  "38b5fc7df6beadc8b014fb8ebb22623b52f7e13a44aa2765fccb2288bed63a13": {
    "Name": "NewPythonAPI",
    "EndpointID": "b24d3d5b4f7bf36e435f69db519e836235cc7b22bc658082292bb8c65a07f66e",
    "MacAddress": "02:42:ac:12:00:05",
    "IPv4Address": "172.18.0.5/16",
    "IPv6Address": ""
  },
  "9dad8eedcd75fd957eadea7ba379d85654df4f7aa4c89d879e5adb2de9336a2": {
    "Name": "NewOracle",
    "EndpointID": "3dd76fac3978032b5da058f5b2ced3ad6c1de991a15ebf1486597fcb45d360b",
    "MacAddress": "02:42:ac:12:00:02",
    "IPv4Address": "172.18.0.2/16",
    "IPv6Address": ""
  },
  "9f494476cafa060130e1f4cd9dd090fe81ee82f1e9318cf44c7ca0f7bd98726": {
    "Name": "NewSpringBootAPI",
    "EndpointID": "4d8b8e40f158aa9da3b26503d0cdbe067839a0e5916496eb56d5423da6f0bc7",
    "MacAddress": "02:42:ac:12:00:04",
    "IPv4Address": "172.18.0.4/16",
    "IPv6Address": ""
  },
  "e995c0bd9ad5419ab09916840e001417610d6cb70e103607a73013929bc8e7ef": {
    "Name": "NginxComBindDePorta3",
    "EndpointID": "7f03ec0c7202f9edf97f1ece01c2eb1344da54ac40b10abcaa70bf8df0ed10f",
    "MacAddress": "02:42:ac:12:00:06",
    "IPv4Address": "172.18.0.6/16",
    "IPv6Address": ""
  },

```

Entrando no container (veja o IP diferente, não começa mais com 172.17, agora começa com 172.18, isso é porque as “sub-redes” são diferentes!):

docker exec -it NginxComBindDePorta3 bash

E vamos tentar chamar primeiro aquele container NginxComBindDePorta primeiro, já que ele tem bind de porta e aceita requisição de qualquer lugar (0.0.0.0):

curl 172.17.0.5:9280

Usei a porta 9280 pois é a que está disponível “pra fora”, mas também poderia tentar na porta 80:

curl 172.17.0.5:80

Será que funcionou? Vamos ver:

```
Prompt de Comando - docker exec -it NginxComBindDePorta3 bash
root@e395cabd0ad5:/# curl 172.17.0.5:9280
curl: (28) Failed to connect to 172.17.0.5 port 9280: Connection timed out
root@e395cabd0ad5:/# curl 172.17.0.5:80
curl: (28) Failed to connect to 172.17.0.5 port 80: Connection timed out
root@e395cabd0ad5:/#
```

Não funcionou porque as “sub-redes” dentro do ambiente do Docker estão separadas e não “se enxergam”, por isso não adianta chamar de dentro do “NginxComBindDePorta3” que tem o IP começando com “172.18” o outro container chamado “NginxComBindDePorta” que tem o IP começando com “172.17”. Na real dentro do “172.18” é como se não existisse nenhum IP “172.17”, por isso eles não se enxergam!

E se usarmos localhost, o que acontece?

curl localhost:80

```
Prompt de Comando - docker exec -it NginxComBindDePorta3 bash
root@e395cabd0ad5:/# curl 172.17.0.5:9280
curl: (28) Failed to connect to 172.17.0.5 port 9280: Connection timed out
root@e395cabd0ad5:/# curl 172.17.0.5:80
curl: (28) Failed to connect to 172.17.0.5 port 80: Connection timed out
root@e395cabd0ad5:/# curl localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

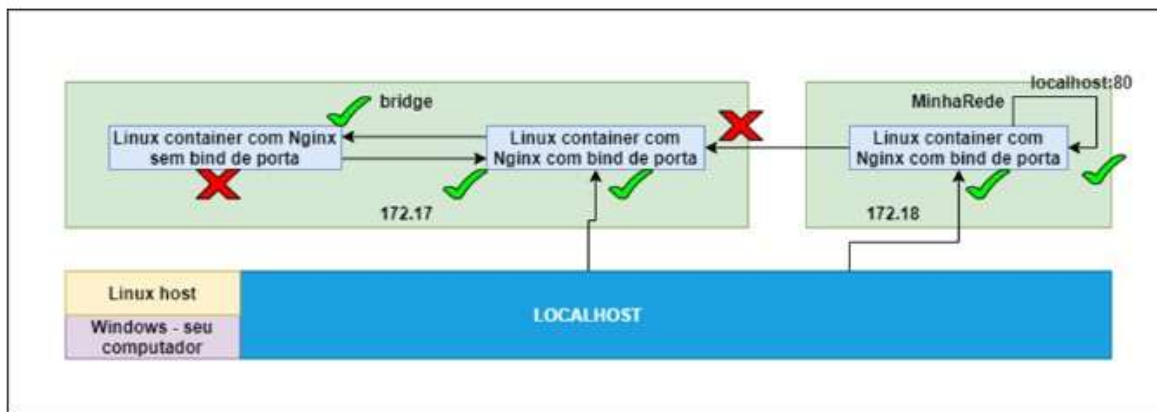
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@e395cabd0ad5:/#
```

Aqui tem pegadinha: quando chamamos localhost de dentro do

“NginxComBindDePorta3”, o “localhost” não é nosso Windows, é o próprio

“NginxComBindDePorta3”! Lembra que eu disse que localhost existe em qualquer sistema operacional? Isso incluir nosso container que tem um “mini” sistema operacional próprio! Então chamar “localhost:80” é chamar a si próprio! A imagem à seguir ilustra isso:



Nesta aula gigante vimos como funciona esse conceito de sistema operacional local, “host” e os “mini” sistemas operacionais dos containers e como eles “se conversam”.

### Atividade Extra

Para se aprofundar no assunto desta aula leia o capítulo 10 da bibliografia de referência: VITALINO, J. F. N.; CASTRO, M. A. N. Descomplicando o docker. 2.ed. Brasport: 2018

Como bibliografia complementar com maior detalhamento sobre imagens e camadas, leia o capítulo 11 da seguinte referência: POULTON, Nigel. Docker Deep

Dive: Zero to Docker in a single book (English Edition). May 2020 ed. Nigel Poulton:  
2020

### Referência Bibliográfica

- VITALINO, J. F. N.; CASTRO, M. A. N. Descomplicando o docker. 2.ed. Brasport: 2018.

**Ir para exercício**