Redes em Docker

Professor: Pedro Horchulhack

Agenda

1. Redes Definidas por Software (SDNs) e Modelo de Rede do Docker
2. Tipos de rede em Docker
3. Gerenciamento de portas

Redes Definidas por Software e Modelo de Rede do Docker

Redes Definidas por Software e Modelo de Rede do Docker

- São redes definidas através de softwares, ou seja, virtuais
  - Redes físicas dependem de um hardware específico, de um fabricante específico, para se comunicarem

- São uma alternativa para infraestruturas de rede principalmente para sua abstração
  - Gestão
  - Flexibilidade
  - Extensibilidade

- Também são uma ótima opção para serviços em nuvem

Redes Definidas por Software e Modelo de Rede do Docker

- Contêineres se beneficiam desse conceito por criarem redes virtuais, através de software, para se comunicarem

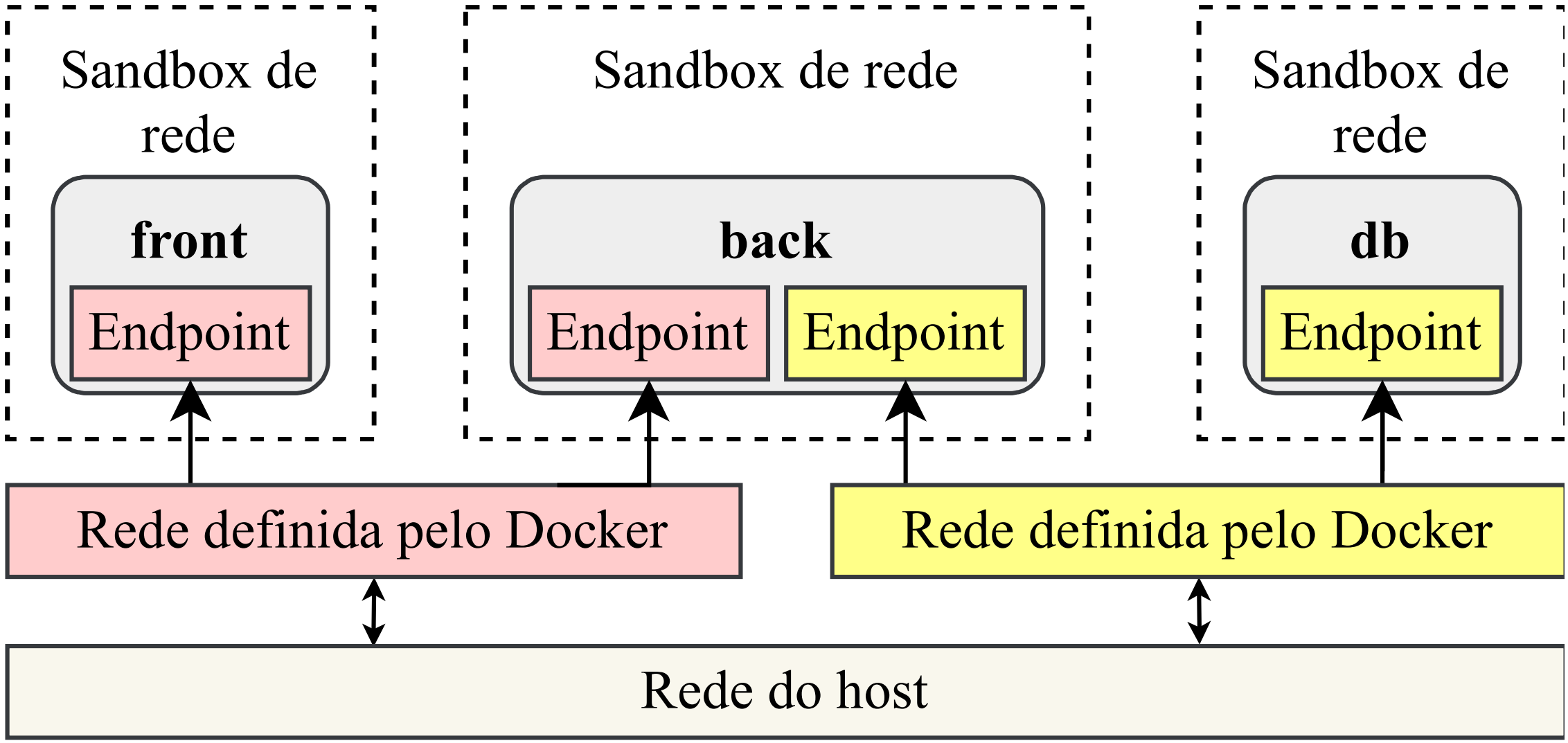
- Permite isolarmos contêineres a nível de rede
  - Podem atuar como firewalls entre contêineres

Redes Definidas por Software e Modelo de Rede do Docker

- O Modelo de Rede de Contêineres (CNM) especifica os requisitos mínimos para implementação de uma rede no Docker

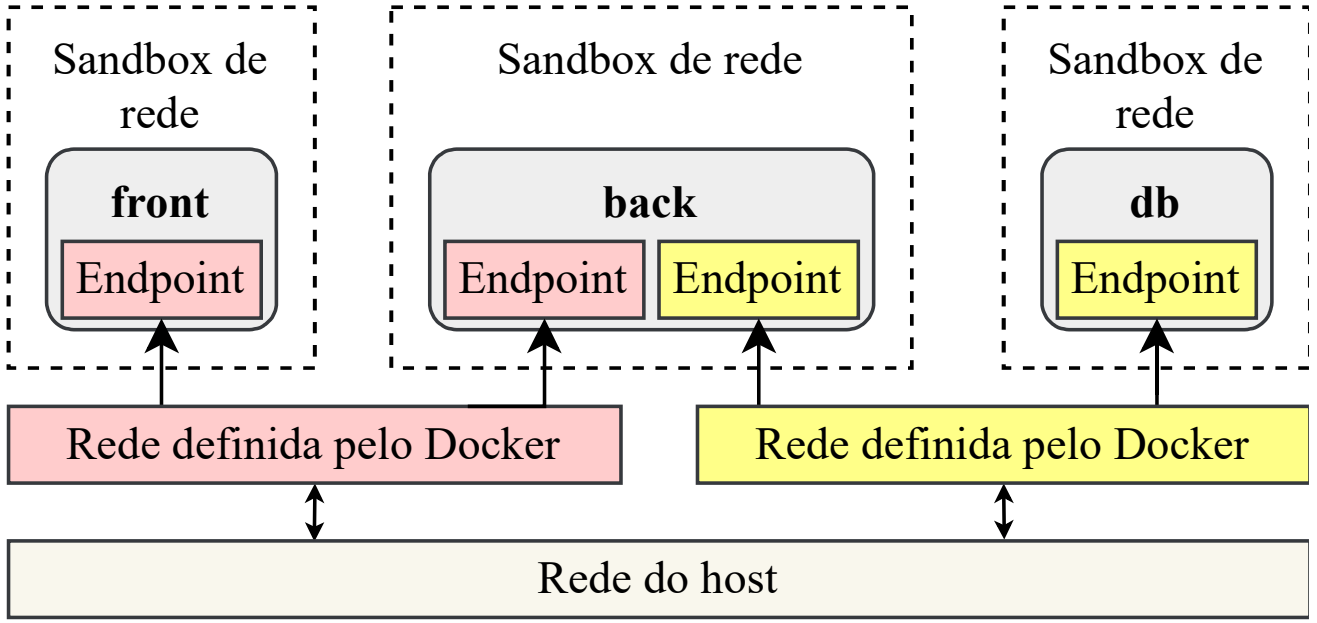
- O CNM consiste em três elementos:
  - **Sandbox de rede**: Contexto que isolará os contêineres do mundo externo
  - **Endpoint**: Conecta a sandbox com a rede
  - **Rede**: Rede implementada no host

Redes Definidas por Software e Modelo de Rede do Docker



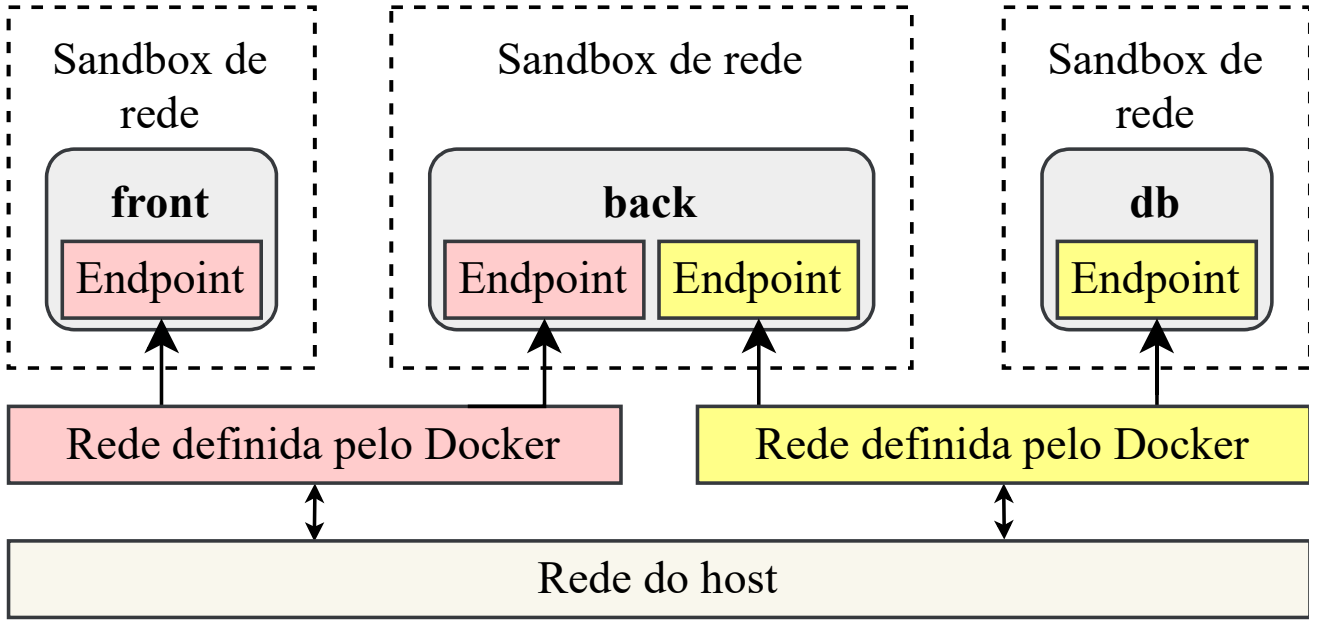
Redes Definidas por Software e Modelo de Rede do Docker

- Cada sandbox irá atuar como um *namespace* de rede
- Cada endpoint será uma interface de rede dentro do contêiner
- Cada rede definida pelo docker será conectada a uma rede no host



Redes Definidas por Software e Modelo de Rede do Docker

- A criação de SDNs é barata e fácil
- Contêineres que estão conectados à mesma rede **conseguem** se comunicar livremente
  - Também é válido para comunicação com o mundo externo



Redes Definidas por Software e Modelo de Rede do Docker

– Algumas implementações do CNM:

Rede	Empresa	Escopo	Descrição
Bridge	Docker	Local	Baseado em redes **bridge** do linux, permite comunicação entre contêineres num mesmo host
Macvlan	Docker	Local	Configura múltiplos endereços de camada-2 em uma única interface física no host
Overlay	Docker	Global	Rede de contêineres multi-nó baseado na tecnologia VXLan
Weave Net	Weaveworks	Global	Redes Docker simples, resilientes e multi-nó
Contiv Network Plugin	Cisco	Global	Rede de contêineres open-source

Tipos de rede

Tipos de rede – Rede Bridge

- Rede que permitirá a conexão entre contêineres no mesmo host ou ao host

- Criará uma interface de rede no sistema chamada **docker0**

```
pedro@secplab00:~$ ifconfig docker0
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:6cff:fe05:f4b0  prefixlen 64  scopeid 0x20<link>
        ether 02:42:6c:05:f4:b0  txqueuelen 0  (Ethernet)
        RX packets 72064  bytes 4407697 (4.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 86830  bytes 289340206 (289.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Tipos de rede – Rede Bridge

- Essa interface também é criada no Docker, agindo como intermediária entre host e contêineres

- Podemos gerenciar redes no Docker através do comando **`docker network`**

```
Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks
```

Tipos de rede – Rede Bridge

- Listando as redes vemos que existe a interface `bridge`, com driver `bridge` e escopo local

```
pedro@secplab00:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
34ff17abe768   bridge    bridge    local
1d3e8d6fd7be   host      host      local
0ecb0c25d3de   none      null      local
```

Tipos de rede – Rede Bridge

```
pedro@secplab00:~$ docker network inspect b...
[
  {
    "Name": "bridge",
    "Id": "34ff17abe76803abf94ff49ef2ccde5e
    "Created": "2024-05-22T16:13:21.8482763
    "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": {},
    "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": {}
  }
```

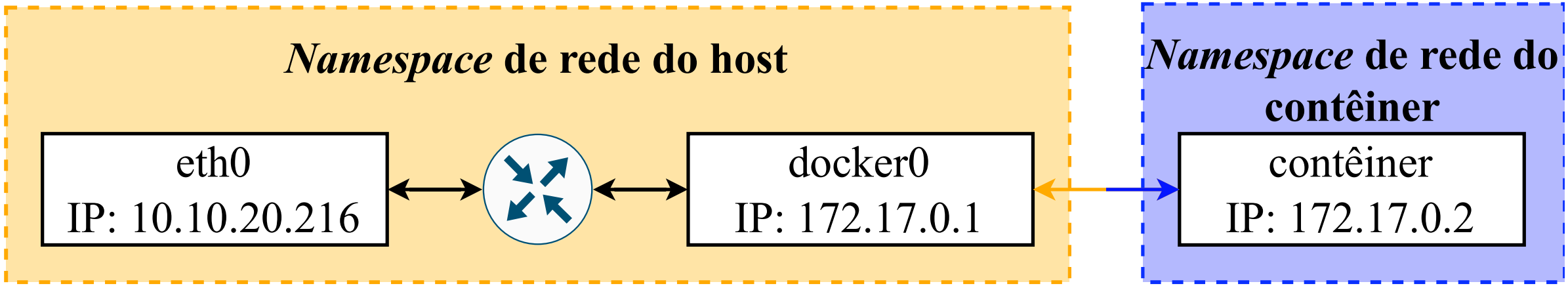
```
pedro@secplab00:~$ ifconfig docker0
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:6cff:fe05:f4b0  prefixlen 64  scopeid 0x20<link>
        ether 02:42:6c:05:f4:b0  txqueuelen 0  (Ethernet)
        RX packets 72064  bytes 4407697 (4.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 86830  bytes 289340206 (289.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- Aqui temos algumas configurações sobre a rede bridge
- A subrede do contêiner é `172.17.0.0/16`, onde:
  - Gateway: `172.17.0.1`
  - Primeiro endereço: `172.17.0.2`
  - Último endereço: `172.17.255.255`
- Na criação de redes Docker é importante se atentar às subredes!
  - Conflitos de IP

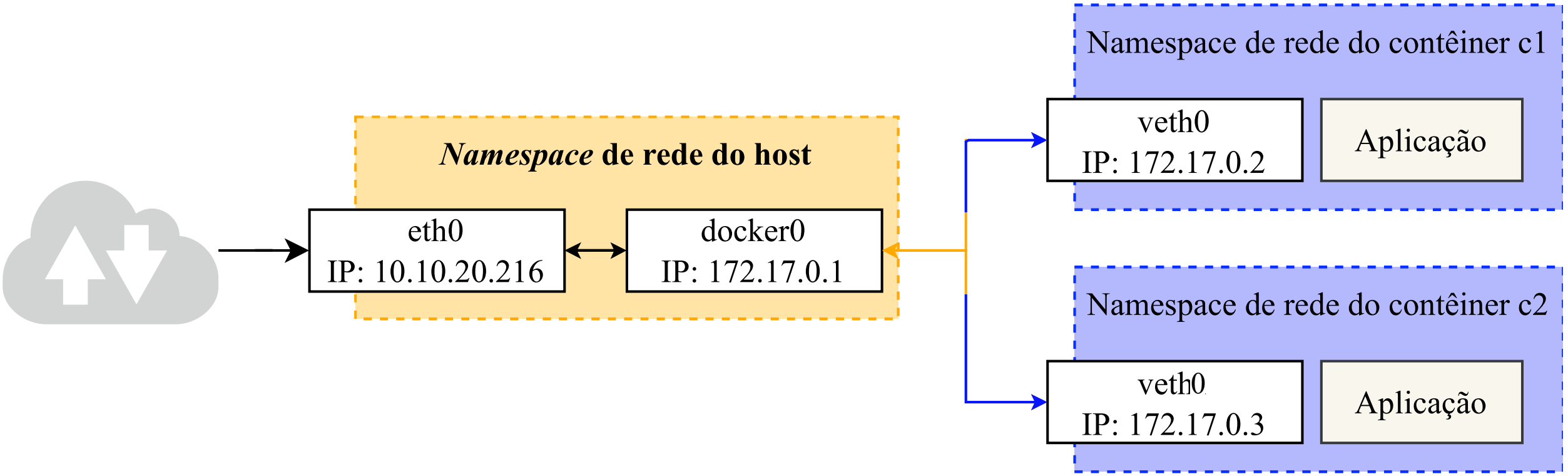
Tipos de rede – Rede Bridge

- Por padrão, toda a conexão de saída (OUTBOUND) do contêiner é permitida, enquanto que as conexões de entrada (INBOUND) são bloqueadas

- Cada contêiner possui uma interface de ethernet virtual (`veth`) conectada à bridge (`docker0`)



Tipos de rede – Rede Bridge



Tipos de rede – Rede Bridge

- Vamos criar uma rede bridge chamada `rede-teste`

```
docker network create --driver bridge rede-teste
docker network inspect rede-teste | grep Subnet
```

```
pedro@secplab00:~$ docker network inspect rede-teste | grep Subnet
                "Subnet": "172.18.0.0/16",
```

- Vamos criar uma rede bridge, com uma subrede diferente, chamada `rede-teste1`

```
docker network create --driver bridge --subnet “10.1.0.0/16” rede-teste1
docker network inspect rede-teste1 | grep Subnet
```

```
pedro@secplab00:~$ docker network inspect rede-teste1 | grep Subnet
                "Subnet": "10.1.0.0/16"
```

Tipos de rede – Rede Bridge

- Para utilizarmos uma rede em um contêiner, temos duas opções:
  - Inicializar especificando uma rede
  - Conectar o contêiner após sua inicialização

- Exemplo de inicialização de um contêiner em uma rede específica:

```
docker run -it --rm --name=c1 --network=rede-teste
alpine:latest /bin/sh
```

- Exemplo de conexão de um contêiner a uma rede:

```
docker network connect rede-teste1 c1
```

Importante lembrar:
Para a conexão de um contêiner a uma rede, **ambos** devem existir

Tipos de rede – Rede Bridge

- Vamos verificar as configurações de rede do contêiner `c1`
  - Dentro do contêiner, execute os comandos:
    - `ip addr show`
    - `ip route`

```
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
37: eth0@if38: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ip route
default via 172.18.0.1 dev eth0
172.18.0.0/16 dev eth0 scope link  src 172.18.0.2
```

Tipos de rede – Rede Bridge

- Agora crie mais um contêiner na rede `rede-teste`, com o nome `c2`
  - Não esqueça de mudar a flag `-it` para `-d`
- Verifique o endereço IP do contêiner criado
- Vamos inspecionar a rede `rede-teste`


```
    "Containers": {
      "60386d5392a1a57e7a42e59c9ce756bcd62f84c1fb53ea8eabfd3786f361018e": {
        "Name": "c1",
        "EndpointID": "3a680bd1d94d8a8b2035a4818787579b045e15a6d640b8ca6780e23a7b4d976e",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
      },
      "e72bb2c63e87d7d9b4371a166405e83e22e46bc410826c0e901445d1cac2ed1a": {
        "Name": "c2",
        "EndpointID": "ec2834d1df70b73a785a71cf6dd1b0110f40159f5685ea7209237ce7061d086a",
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
      }
```

Tipos de rede – **Host** e null

- A rede host é utilizada para depuração ou análise a partir da máquina hospedeira
  - É recomendado que utilize somente nesses cenários**. Jamais em produção!**

- Problemas de se utilizar a rede **host**
  - Segurança
  - Isolamento
  - Conflito de portas

Tipos de rede – **Host** e null

```
pedro@secplab00:~$ docker run -it --rm --network=host alpine sh
/ # ip addr show eno1
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP qlen 1000
    link/ether 78:24:af:79:90:65 brd ff:ff:ff:ff:ff:ff
    inet 10.32.11.36/26 brd 10.32.11.63 scope global dynamic noprefixroute eno1
       valid_lft 32768sec preferred_lft 32768sec
    inet6 fe80::6e61:27f:39d7:5c3b/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
/ # ip route
default via 10.32.11.1 dev eno1  metric 100
10.1.0.0/16 dev br-ff2c6a3d3812 scope link  src 10.1.0.1
10.32.11.0/26 dev eno1 scope link  src 10.32.11.36  metric 100
10.32.11.36 dev sdwan0 scope host  metric 25
10.95.35.246 dev sdwan0 scope host  metric 25
100.95.0.251 dev sdwan0 scope host  metric 25
100.95.0.252 dev sdwan0 scope host  metric 25
100.95.0.253 dev sdwan0 scope host  metric 25
100.95.0.254 dev sdwan0 scope host  metric 25
100.96.0.0/12 dev sdwan0 scope host  metric 25
169.254.0.0/16 dev eno1 scope link  metric 1000
172.17.0.0/16 dev docker0 scope link  src 172.17.0.1
172.18.0.0/16 dev br-43a546d907ac scope link  src 172.18.0.1
```

```
pedro@secplab00:~$ ifconfig eno1
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.32.11.36  netmask 255.255.255.192  broadcast 10.32.11.63
        inet6 fe80::6e61:27f:39d7:5c3b  prefixlen 64  scopeid 0x20<link>
        ether 78:24:af:79:90:65  txqueuelen 1000  (Ethernet)
        RX packets 16892420  bytes 2206717540 (2.2 GB)
        RX errors 43  dropped 81666  overruns 0  frame 28
        TX packets 16588393  bytes 1756822299 (1.7 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0xf7100000-f7120000

pedro@secplab00:~$ ip route
default via 10.32.11.1 dev eno1 proto dhcp metric 100
10.1.0.0/16 dev br-ff2c6a3d3812 proto kernel scope link src 10.1.0.1 linkdown
10.32.11.0/26 dev eno1 proto kernel scope link src 10.32.11.36 metric 100
10.32.11.36 dev sdwan0 proto static scope host metric 25
10.95.35.246 dev sdwan0 proto static scope host metric 25
100.95.0.251 dev sdwan0 proto static scope host metric 25
100.95.0.252 dev sdwan0 proto static scope host metric 25
100.95.0.253 dev sdwan0 proto static scope host metric 25
100.95.0.254 dev sdwan0 proto static scope host metric 25
100.96.0.0/12 dev sdwan0 proto static scope host metric 25
169.254.0.0/16 dev eno1 scope link metric 1000
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
172.18.0.0/16 dev br-43a546d907ac proto kernel scope link src 172.18.0.1
```

Tipos de rede – Host e **null**

- A rede null, como o nome já diz, é uma rede isolada do mundo
  - Utilizada quando contêineres não precisam de acesso à rede
  - Exemplo: Alguma contêiner somente de processamento

```
pedro@secplab00:~$ docker run -it --rm --network=none alpine sh
/ # ip a show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
/ # ip route
/ #
```

Tipos de rede – Host e **null**

- A rede null, como o nome já diz, é uma rede isolada do mundo
  - Utilizada quando contêineres não precisam de acesso à rede
  - Exemplo: Alguma contêiner somente de processamento

```
pedro@secplab00:~$ docker run -it --rm --network=none alpine sh
/ # ip a show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
/ # ip route
/ #
```

Utilizando o mesmo *namespace* de rede

- É possível associarmos um novo contêiner a um *namespace* de rede já existente

- Permite acesso a rede localhost entre contêineres

Utilizando o mesmo *namespace* de rede – Exemplo

```
pedro@secplab00:~$ docker inspect nginx -f "{{json .NetworkSettings.Networks}}" | jq
{
  "rede-teste": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": null,
    "MacAddress": "02:42:ac:12:00:02",
    "NetworkID": "43a546d907acde8567fb6c349652cc565a101c0861655d7c787f05b1be7fe13c",
    "EndpointID": "afc241e7d30a29661cf4c9dd39b80446684470da27c39439ea5cb471aa90a26f",
    "Gateway": "172.18.0.1",
    "IPAddress": "172.18.0.2",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "DriverOpts": null,
    "DNSNames": [
      "nginx",
      "6b787b2a12a0"
    ]
  }
}
```

```
pedro@secplab00:~$ docker run -d --rm --name teste --network=container:nginx alpine /bin/sh -c 'sleep 3600
0ac80fa16601a8508b0e3abb25e24bc9235b530e27e4686d069d348b0aa39f7f
pedro@secplab00:~$ docker inspect teste -f "{{json .NetworkSettings}}" | jq
{
  "Bridge": "",
  "SandboxID": "",
  "SandboxKey": "",
  "Ports": {},
  "HairpinMode": false,
  "LinkLocalIPv6Address": "",
  "LinkLocalIPv6PrefixLen": 0,
  "SecondaryIPAddresses": null,
  "SecondaryIPv6Addresses": null,
  "EndpointID": "",
  "Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "IPAddress": "",
  "IPPrefixLen": 0,
  "IPv6Gateway": "",
  "MacAddress": "",
  "Networks": {}
}
```

Utilizando o mesmo *namespace* de rede – Exemplo

```
pedro@secplab00:~$ docker exec -it teste /bin/sh -c 'wget -qO- localhost'
<!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>
```

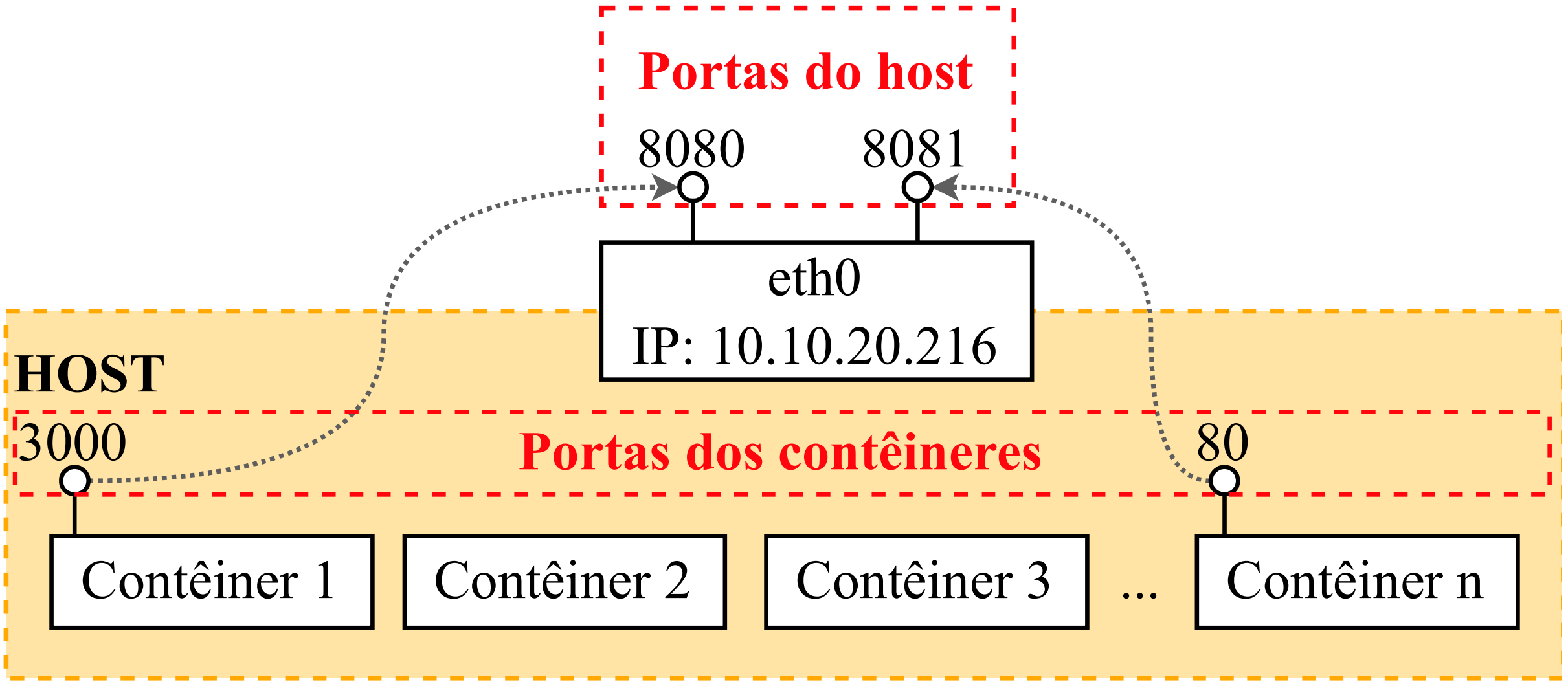
```
pedro@secplab00:~$ docker exec -it teste ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:172.18.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:31 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3775 (3.6 KiB)  TX bytes:0 (0.0 B)
```

Gerenciamento de portas

Gerenciamento de portas

- As redes dos contêineres atuam, também, como um firewall
  - Bloqueiam acesso externo ao contêiner e permitem acesso externo do contêiner
- Precisamos, de alguma forma, dependendo da aplicação, permitir acesso externo
  - Vamos **publicar** portas de um contêiner para que seja acessível do host
- A pilha de rede do contêiner e do host são completamente separadas
  - Fazemos um mapeamento entre uma porta do host para uma porta do contêiner

Gerenciamento de portas



Gerenciamento de portas

- Temos duas estratégias para publicas portas:
  1. Publicar portas específicas do contêiner para específica do host
  2. Publicar portas dos contêineres para uma aleatória, no range **32xxx**, no host

- Exemplos:
  1. `docker run --rm -P -d nginx`
  2. `docker run --rm -p 8080:80 -d nginx`

- Execute o primeiro comando, veja qual foi a porta associada e acesse-a pelo navegador

Referências

Referências

1. Docker. **Volumes | Docker Docs**. Docker Inc, 2024. Disponível em: <https://docs.docker.com/storage/volumes/>. Acesso em 27 de maio de 2024.
2. Docker. **Networking Overview**. Docker Inc, 2024. Disponível em: <https://docs.docker.com/network/> Acesso em: 29 de maio de 2024.
3. Docker. **Docker overview**. Docker Inc., 2024. Disponível em: <https://docs.docker.com/get-started/overview/#docker-architecture>. Acesso em 12 de maio de 2024.
4. Tigera. **Container Networking: What YouShould Know**. Tigera Inc., 2024. Disponível em: <https://www.tigera.io/learn/guides/kubernetes-networking/container-networking/> Acesso em 29 de maio de 2024.

Referências

5. RICE, Liz. **Container security: fundamental technology concepts that protect containerized applications**. First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2020.
6. SCHENKER, Gabriel Nicolas. **The ultimate Docker container book: build, test, ship, and run containers with Docker and Kubernetes**. Third edition. Birmingham, UK: Packt Publishing Ltd., 2023.
7. FARHADY, Hamid; LEE, HyunYong; NAKAO, Akihiro. **Software-Defined Networking: A survey**. Computer Networks, v. 81, p. 79–95, 2015.

Contato: p.horchulhack@pucpr.br