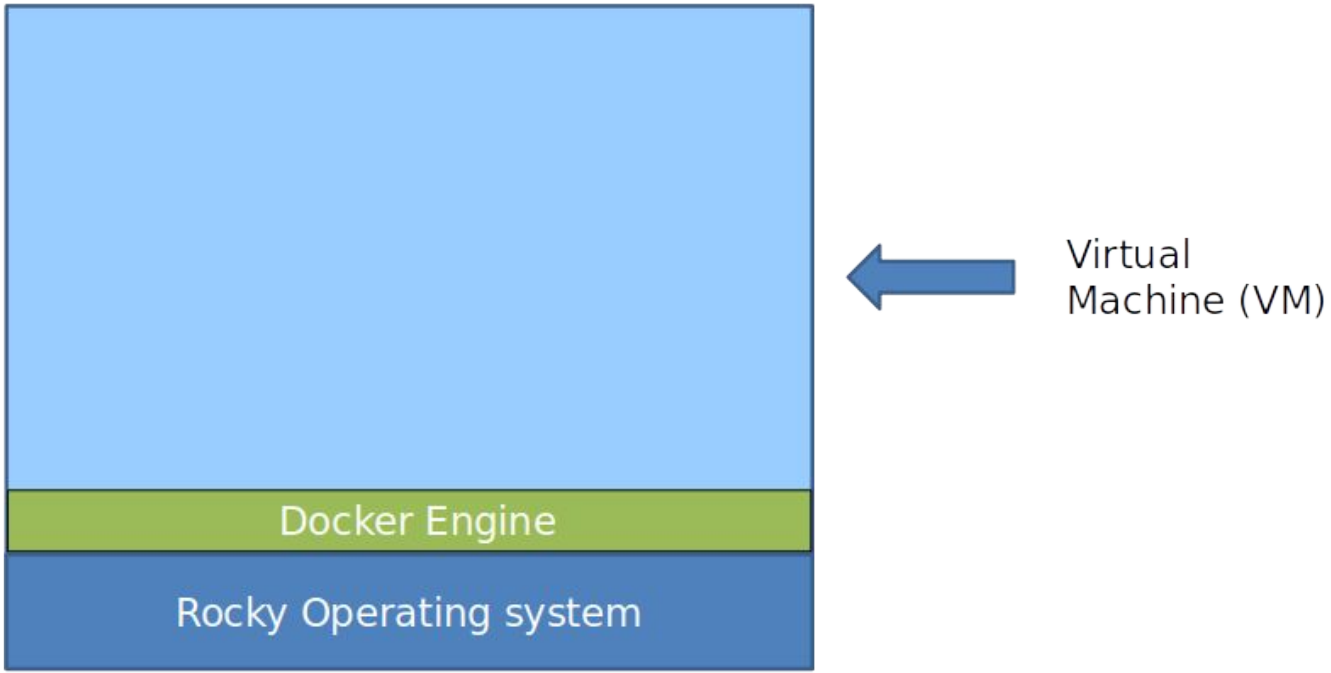
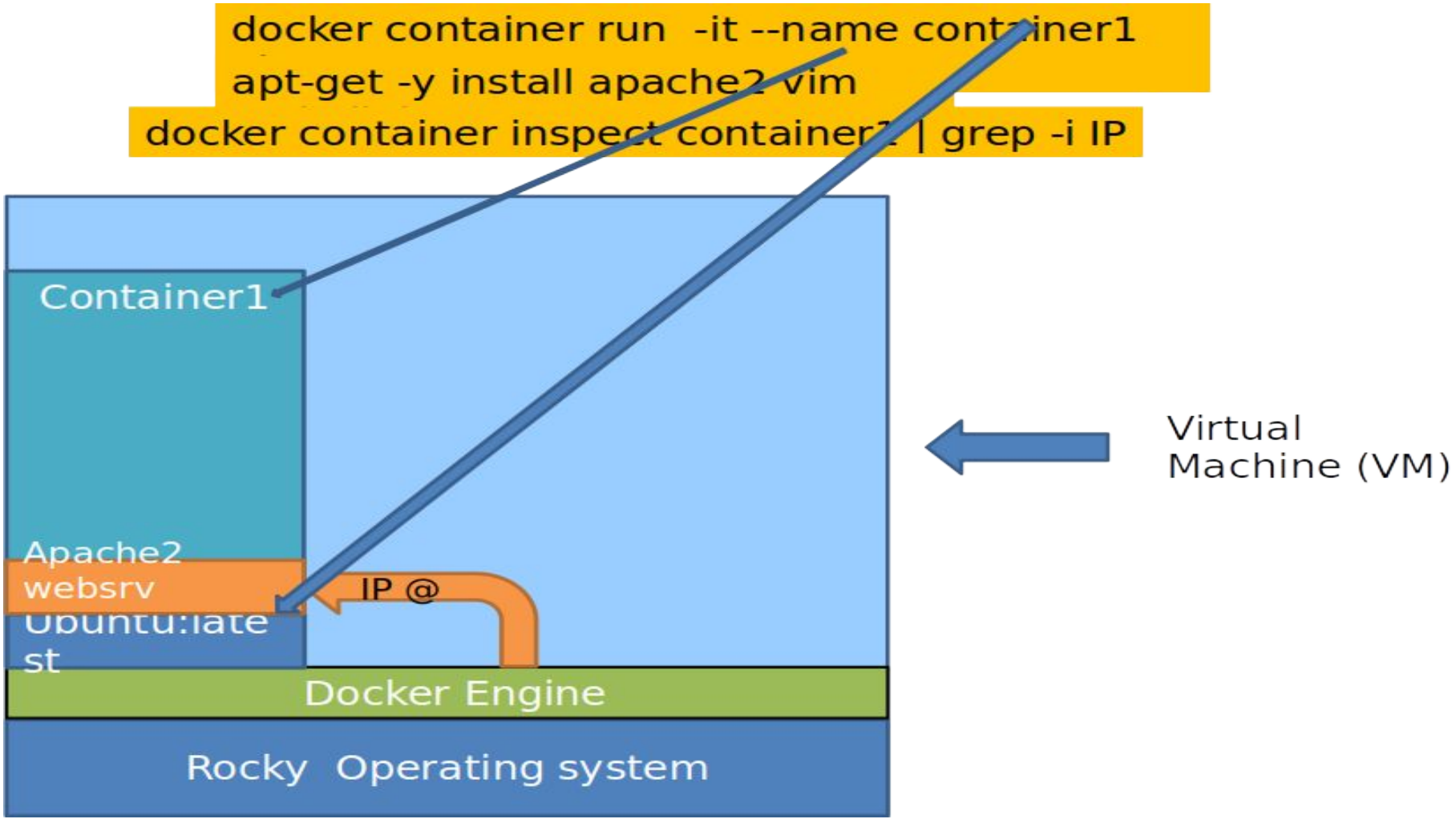
```
[root@docker vagrant ~]# ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 08:00:27:26:66:72 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.19/24 brd 192.168.1.255 scope global dynamic noprefixroute
enp0s3
       valid_lft 76902sec preferred_lft 76902sec
    inet6 fe80::a00:27ff:fe26:6672/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
group default
    link/ether 02:42:7d:87:08:c2 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
[root@docker vagrant ~]#
```

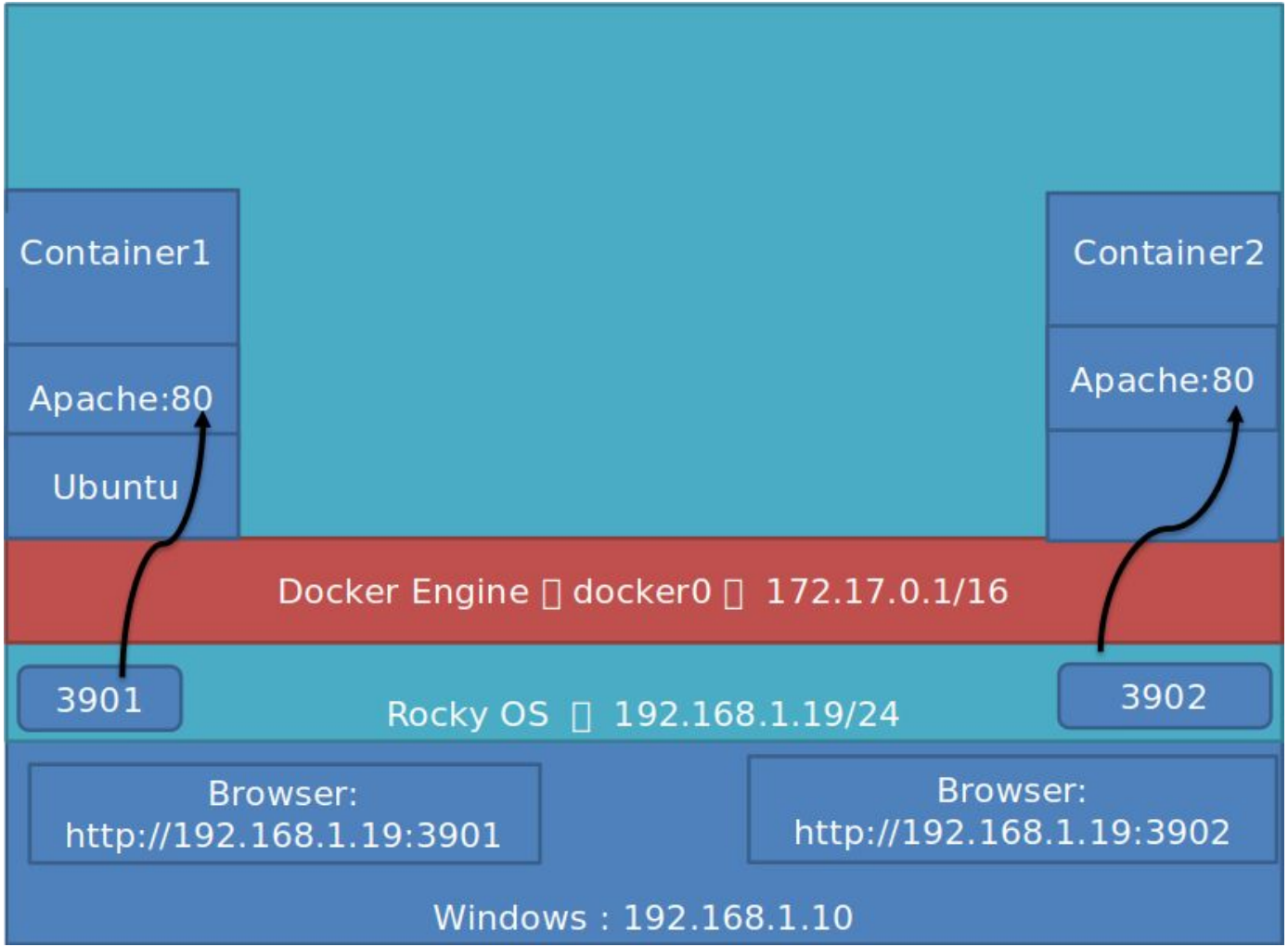
```
yum remove docker  docker-client  docker-client-latest  docker-common  docker-latest docker-latest-logrotate  docker-logrotate docker-engine podman buildah -y
```

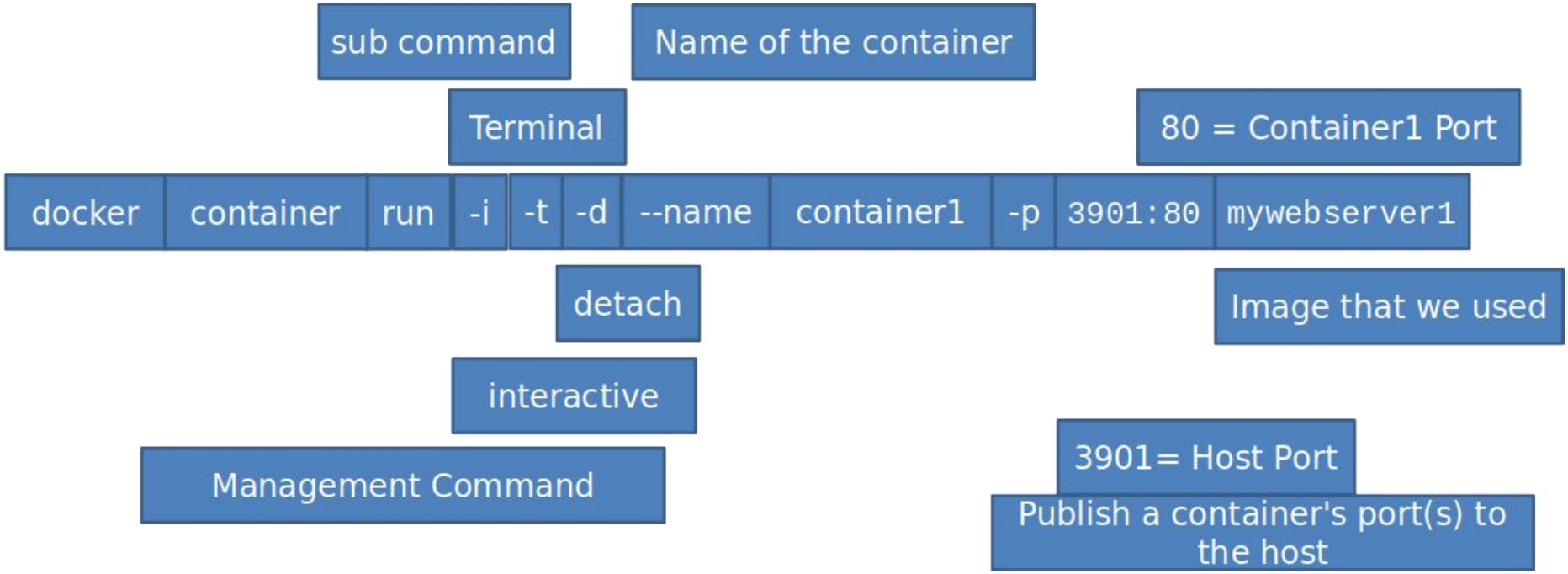



```
yum install docker-ce docker-ce-cli containerd.io -y
```

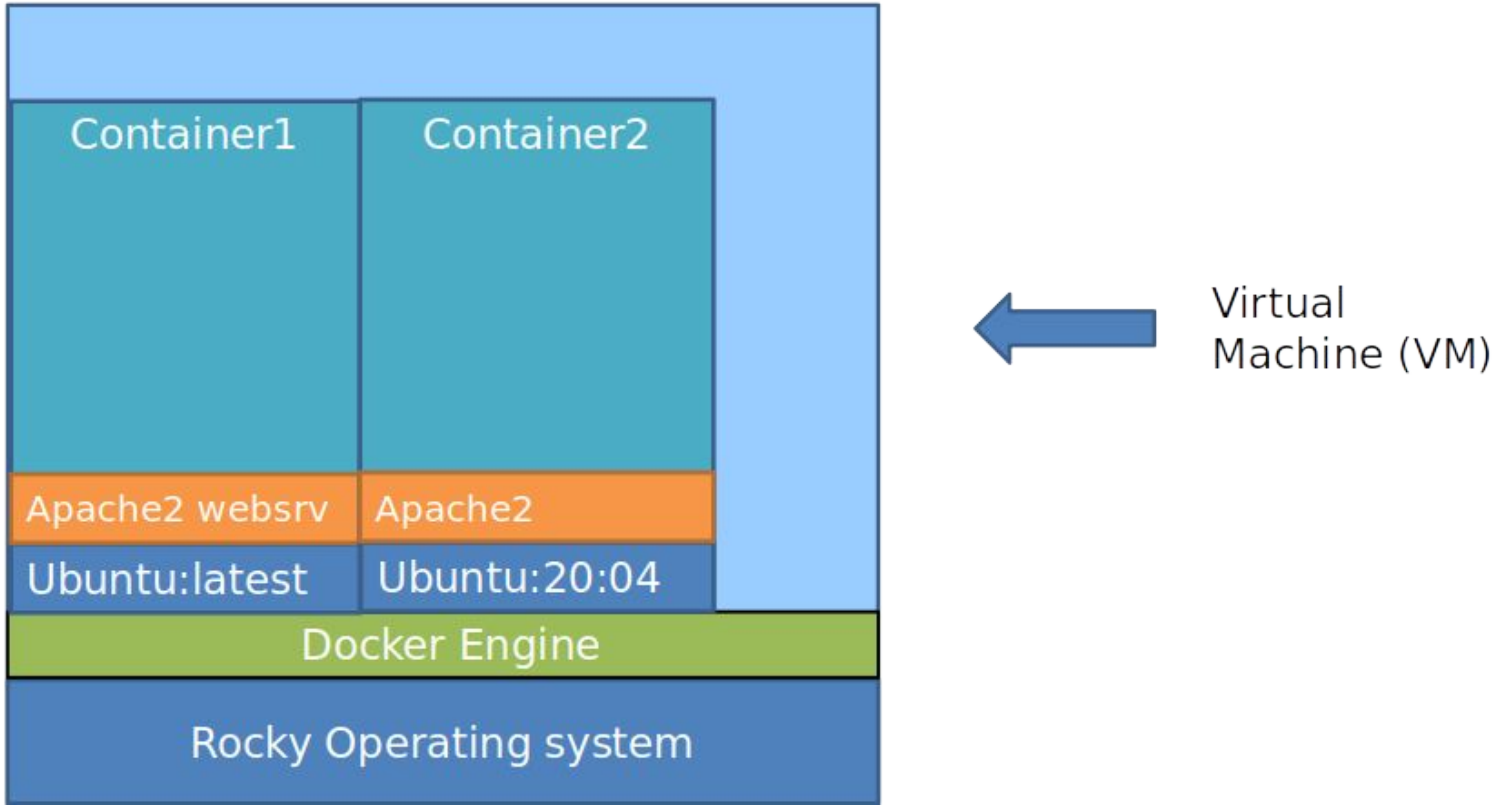

docker container run -it --name container1
apt-get -y install apache2 vim
docker container inspect container1 | grep -i IP
Container1
Apache2
websrv
Ubuntu:late
st
IP @
Docker Engine
Rocky Operating system
Virtual
Machine (VM)

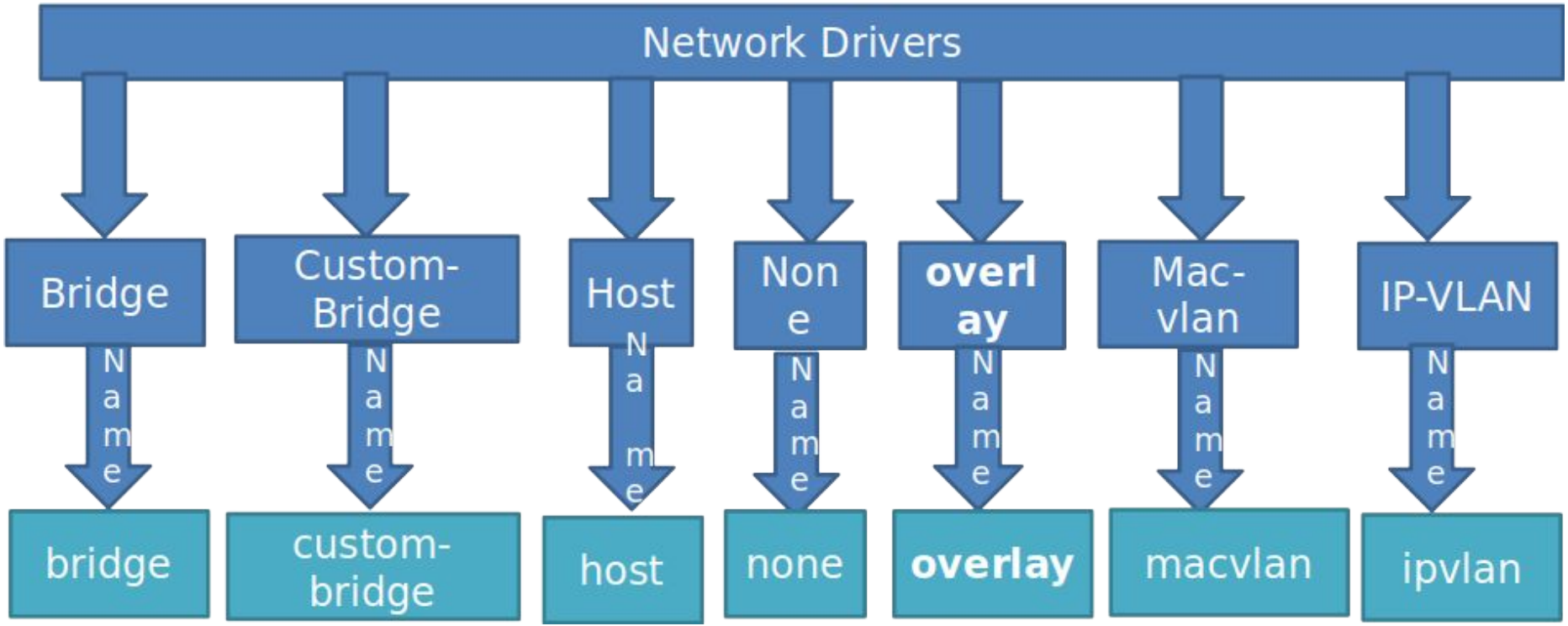
Publish container’s port to host


Container1
Apache:80
Ubuntu
Container2
Apache:80
Docker Engine docker0 172.17.0.1/16
3901
Rocky OS 192.168.1.19/24
3902
Browser:
http://192.168.1.19:3901
Browser:
http://192.168.1.19:3902
Windows : 192.168.1.10

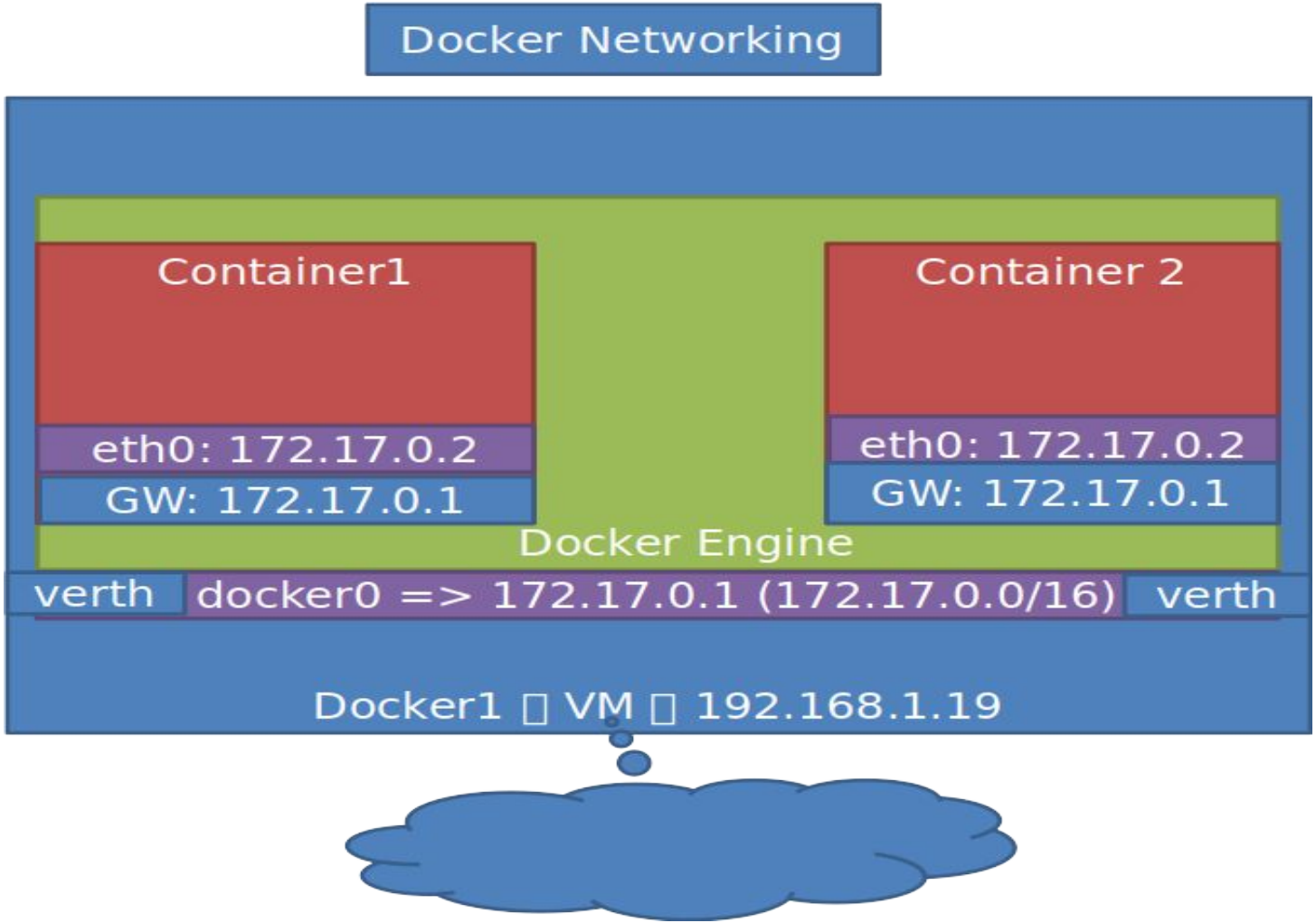

sub command
Name of the container
Terminal
80 = Container1 Port
docker container run -i -t -d --name container1 -p 3901:80 mywebserver1
detach
Image that we used
interactive
3901= Host Port
Management Command
Publish a container's port(s) to the host

docker container run -it --name container2 ubuntu:20:04




Network Drivers
Bridge
Name
bridge
Custom-Bridge
Name
custom-bridge
Host
Name
host
None
Name
none
overlay
Name
overlay
Mac-vlan
Name
macvlan
IP-VLAN
Name
ipvlan

```
[root@docker vagrant]# docker network ls
NETWORK ID          NAME                DRIVER
SCOPE
969f1f77cb4d        bridge              bridge
    local
d80eda927a30        host                host
    local
e8be33cf6aa9        none                null
    local
```


Docker Networking
Container1
eth0: 172.17.0.2
GW: 172.17.0.1
Container 2
eth0: 172.17.0.2
GW: 172.17.0.1
Docker Engine
verth
docker0 => 172.17.0.1 (172.17.0.0/16)
verth
Docker1 □ VM □ 192.168.1.19

Agenda

- What is None driver and where we can use?
- What is host driver and where we can use?
- What is default bridge driver?
- How to Create / Delete the network driver?
- Difference between default & custom bridge driver ?
- What is ipvlan driver and where we can use?
- What is macvlan driver and where we can use?

None Network

- This network will totally disable the networking stack on a container. It means that, this mode will not configure IP address for container and does not have access to external word.
- Option that we can use in the command line to enable this network

  --network none / --net none

Use case of None network

- It has the loopback Ip address, we can execute batch jobs in the container.

LAB None network driver

https://github.com/cfitechops/Docker/blob/main/04-docker.md

Host Network Driver

What is Host network?

Docker Host network is also called as docker network host.

It is a default network driver offered by Docker engine.

One can use host network, If we don’t want to isolate the container’s network from the host machine.

It means, container will share the host’s network stack and all interfaces from the host will be available to the container.

No extra IP will be assigned to a container and no port need to be defined at the time of container creation.

What are the use case of Host networking

Where we need performance improvement, because it does not require network address translation (NAT).

In production, we use seldom this networking driver.

The host networking driver only works on Linux hosts, and is not supported on Docker Desktop for Mac, & Windows, or Docker EE for Windows Server.

LAB Host network driver

https://github.com/cfitechops/Docker/blob/main/05-docker.md

Default bridge driver

docker container run -it -d --name cut1-container 1 -p 3901:80 mywebserver1

docker container run -it -d --name cust1-container2 -p 3902:80 mywebserver1

docker container inspect cust1-container2 | grep -i ipaddress

[root@ ~]# ***docker container run -it -d --name cut2-container 1 -p 3903:80 mywebserver1***

[root@ ~]# **docker container inspect cut2-container1 | grep -i ipaddress**
"SecondaryIPAddresses": null,
"IPAddress": "172.17.0.4",
"IPAddress": "**172.17.0.4**",

[root@ ~]# **docker container exec container1 ping -c 2 172.17.0.4**
PING 172.17.0.4 (172.17.0.4) 56(84) bytes of data.
64 bytes from 172.17.0.4: icmp_seq=1 ttl=64 time=0.070 ms
64 bytes from 172.17.0.4: icmp_seq=2 ttl=64 time=0.191 ms

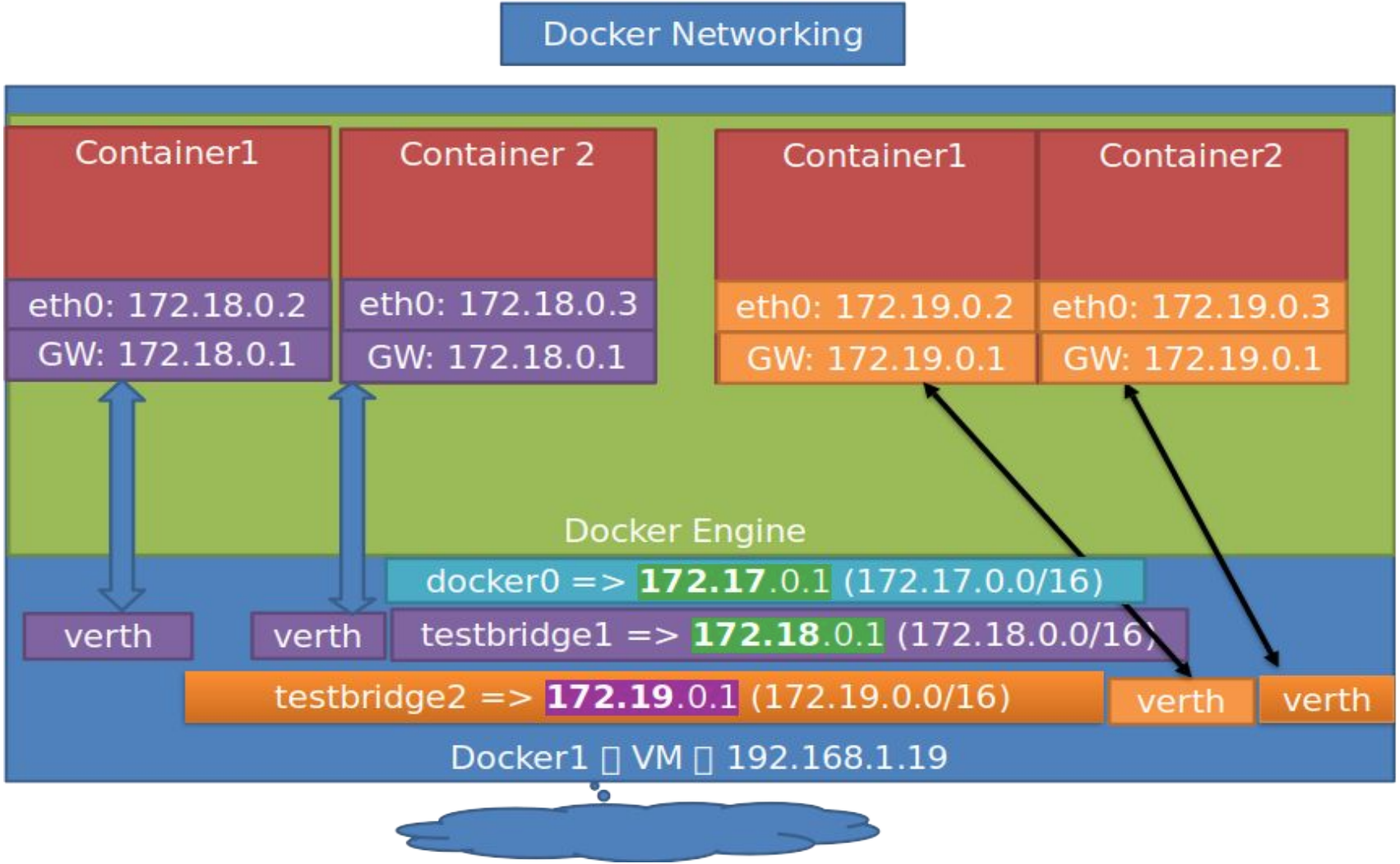
--- 172.17.0.4 ping statistics ---
2 packets transmitted, 2 received, **0% packet loss**, time 999ms
rtt min/avg/max/mdev = 0.070/0.130/0.191/0.061 ms

[root@ ~]# **docker container exec -it cust1-container 1 /bin/bash**
root@27ed6de3a308:/data# **curl http://172.17.0.4**
<p> I'm running this website on Docker Container1 on Ubuntu OS
root@27ed6de3a308:/data#

How to separate the customers' network

- It depends, what exactly your requirement is.
- If you want to use your VM / Host machine IP address with different port numbers to access the containers, then you may use custom bridge network.

We will try to understand different situation when we process with other network drivers.


Docker Networking
Container1
Container 2
Container1
Container2
eth0: 172.18.0.2
GW: 172.18.0.1
eth0: 172.18.0.3
GW: 172.18.0.1
eth0: 172.19.0.2
GW: 172.19.0.1
eth0: 172.19.0.3
GW: 172.19.0.1
Docker Engine
docker0 => 172.17.0.1 (172.17.0.0/16)
verth
verth
testbridge1 => 172.18.0.1 (172.18.0.0/16)
testbridge2 => 172.19.0.1 (172.19.0.0/16)
verth
verth
Docker1 VM 192.168.1.19

How to **Create** the **custom network driver**?

```
docker network create testbridge1
docker network create testbridge2
```

```
[root@ ~]# docker network  inspect testbridge1
[
    {
        "Name": "testbridge1",
        "Id": "a07b927f55ee3d4120b265f2dfb984cae42348070f980bdcbb61d0e21cafced7",
        "Created": "2022-07-16T06:07:08.926119086-04:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
```

```
[root@ ~]# docker network  inspect bridge | grep Subnet
            "Subnet": "172.17.0.0/16",
```

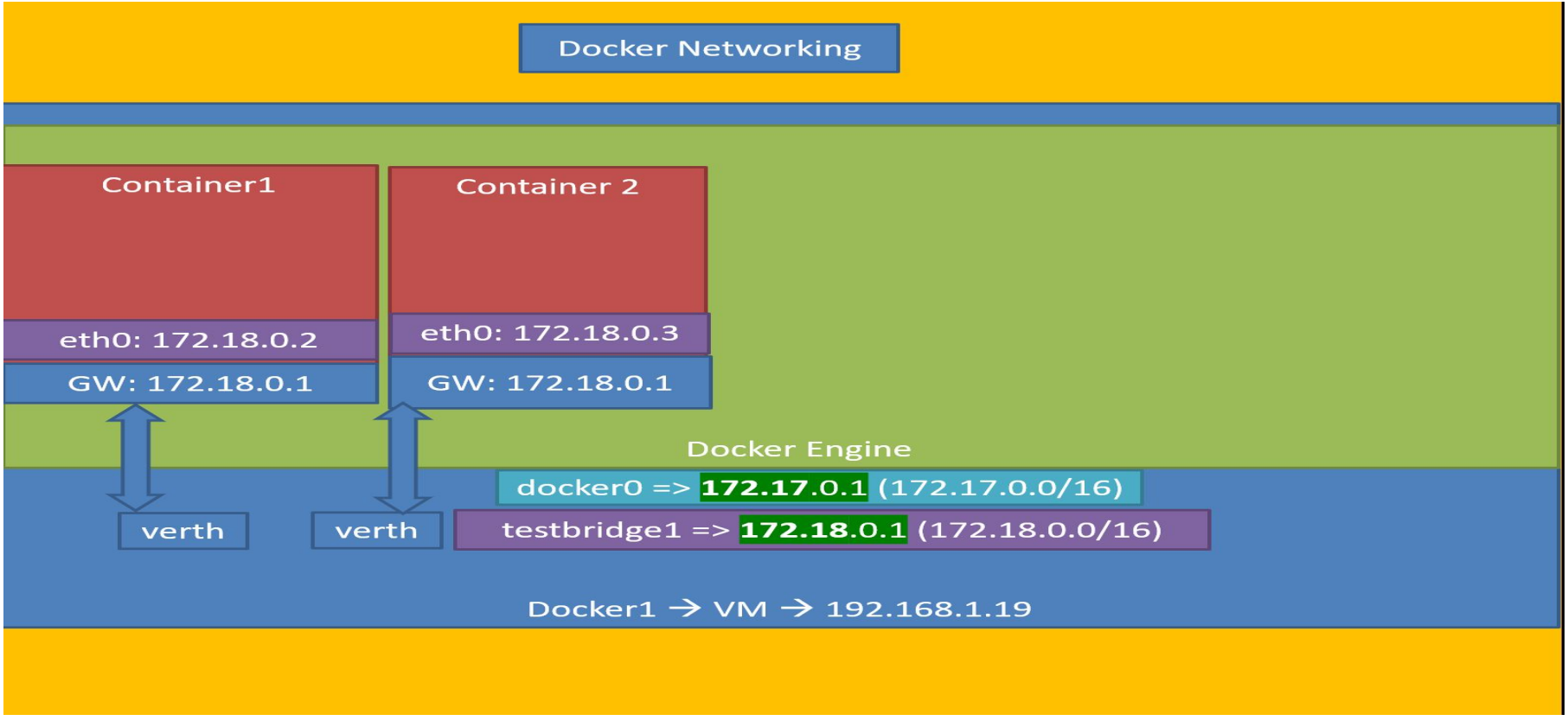
How to attach the container with specific network

docker run -itd **--network=[network-name]** imagename

docker run -itd **--network= testbridge1** mywebserver1

```
docker container run -it -d --network=testbridge1  --name testbridge1-container1  -p 3904:80 mywebserver1

docker container run -it -d --network=testbridge1  --name testbridge1-container2  -p 3905:80 mywebserver1

docker container run -it -d --network=testbridge2 --name testbridge2-container1  -p 3906:80 mywebserver1

docker container run -it -d  --net testbridge2  --cap-add=NAT_ADMIN  --name testbridge2-container2  -p
3907:80 mywebserver1


[root@ ~]# docker container exec testbridge1-container1 ifconfig eth0 | grep Mask
          inet addr:172.18.0.2  Bcast:172.18.255.255  Mask:255.255.0.0

[root@ ~]# docker container exec testbridge2-container1 ifconfig eth0 | grep Mask
          inet addr:172.19.0.2  Bcast:172.19.255.255  Mask:255.255.0.0
```

Ping test within custom bridge network



```
[root@ ~]#  docker container inspect testbridge1-container2 | grep -i IPaddress
        "SecondaryIPAddresses": null,
        "IPAddress": "",
                "IPAddress": "172.18.0.3",

[root@ ~]#  docker container exec testbridge1-container1 ping -c 2 172.18.0.3
PING 172.18.0.3 (172.18.0.3) 56(84) bytes of data.
64 bytes from 172.18.0.3: icmp_seq=1 ttl=64 time=0.142 ms
64 bytes from 172.18.0.3: icmp_seq=2 ttl=64 time=0.063 ms

[root@ ~]#   docker container exec testbridge1-container1 ping -c 2 testbridge1-container2
PING testbridge1-container2 (172.18.0.3) 56(84) bytes of data.
64 bytes from testbridge1-container2.testbridge1 (172.18.0.3): icmp_seq=1 ttl=64 time=0.072 ms
64 bytes from testbridge1-container2.testbridge1 (172.18.0.3): icmp_seq=2 ttl=64 time=0.066 ms
```

Inbuild DNS in container

[root@ ~]# docker container exec **testbridge1-container1** cat /etc/resolv.conf

nameserver **127.0.0.11**

options ndots:0

[root@ ~]#

Ping test between two custom bridge network

[root@ ~]# docker container inspect **testbridge2-container1** | grep -i IPaddress

"SecondaryIPAddresses": null,

"IPAddress": "",

"IPAddress": "**172.19.0.2**",

[root@ ~]# **docker container exec testbridge1-container1 ping -c 2 -w 3 172.19.0.2**

PING 172.19.0.2 (172.19.0.2) 56(84) bytes of data.

--- 172.19.0.2 ping statistics ---

3 packets transmitted, 0 received, **100% packet loss,** time 2048ms

[root@ ~]# **docker container exec testbridge1-container1 ping -c 2 -w 3 testbridge2-container1**

ping: unknown host testbridge2-container2

[root@ ~]#

How to make a connectivity between two different networks?

Syntax: docker network connect [OPTIONS] NETWORK CONTAINER

[root@ ~]# **docker network connect testbridge1 testbridge2-container1**

[root@ ~]# **docker container exec testbridge1-container1 ping -c 2 -w 3 testbridge2-container1**

PING testbridge2-container1 (172.18.0.4) 56(84) bytes of data.

64 bytes from testbridge2-container1.testbridge1 (172.18.0.4): icmp_seq=1 ttl=64 time=0.185 ms

64 bytes from testbridge2-container1.testbridge1 (172.18.0.4): icmp_seq=2 ttl=64 time=0.113 ms

2 packets transmitted, 2 received, **0% packet loss**, time 1001ms

rtt min/avg/max/mdev = 0.113/0.149/0.185/0.036 ms

[root@ ~]# **docker container inspect testbridge2-container1 | grep -i IPaddress**

"SecondaryIPAddresses": null,

"IPAddress": "",

"IPAddress": "172.18.0.4",

"IPAddress": "172.19.0.2",

How to remove the connectivity between two different networks?

Syntax: docker network disconnect [OPTIONS] NETWORK CONTAINER

[root@ ~]# docker network **disconnect** testbridge1 testbridge2-container1

[root@ ~]# docker container exec testbridge1-container1 ping -c 2 -w 3 testbridge2-container1

ping: unknown host testbridge2-container1

[root@ ~]#

Difference between default Bridge & custom Bridge network?

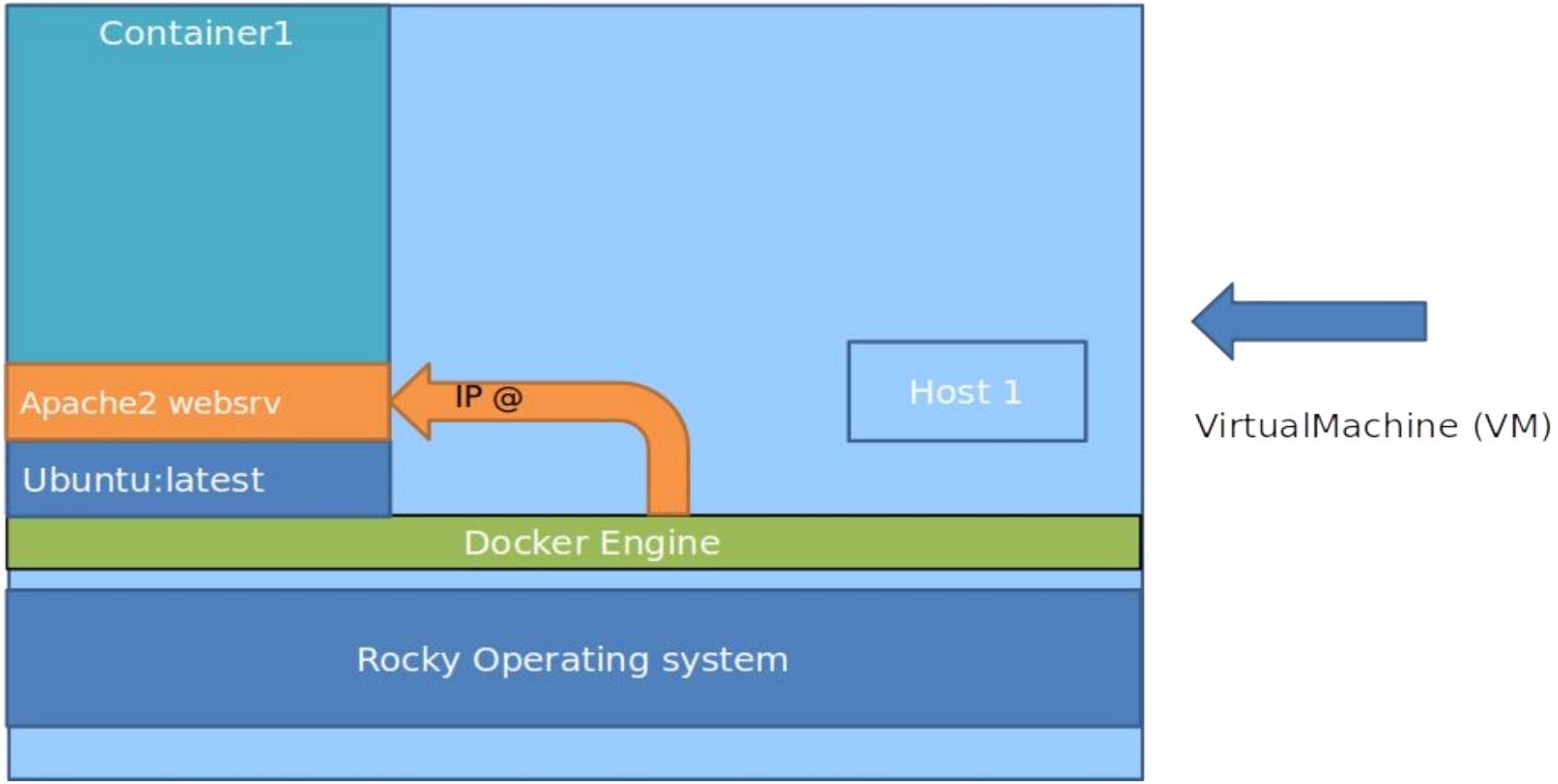
Default Bridge Network

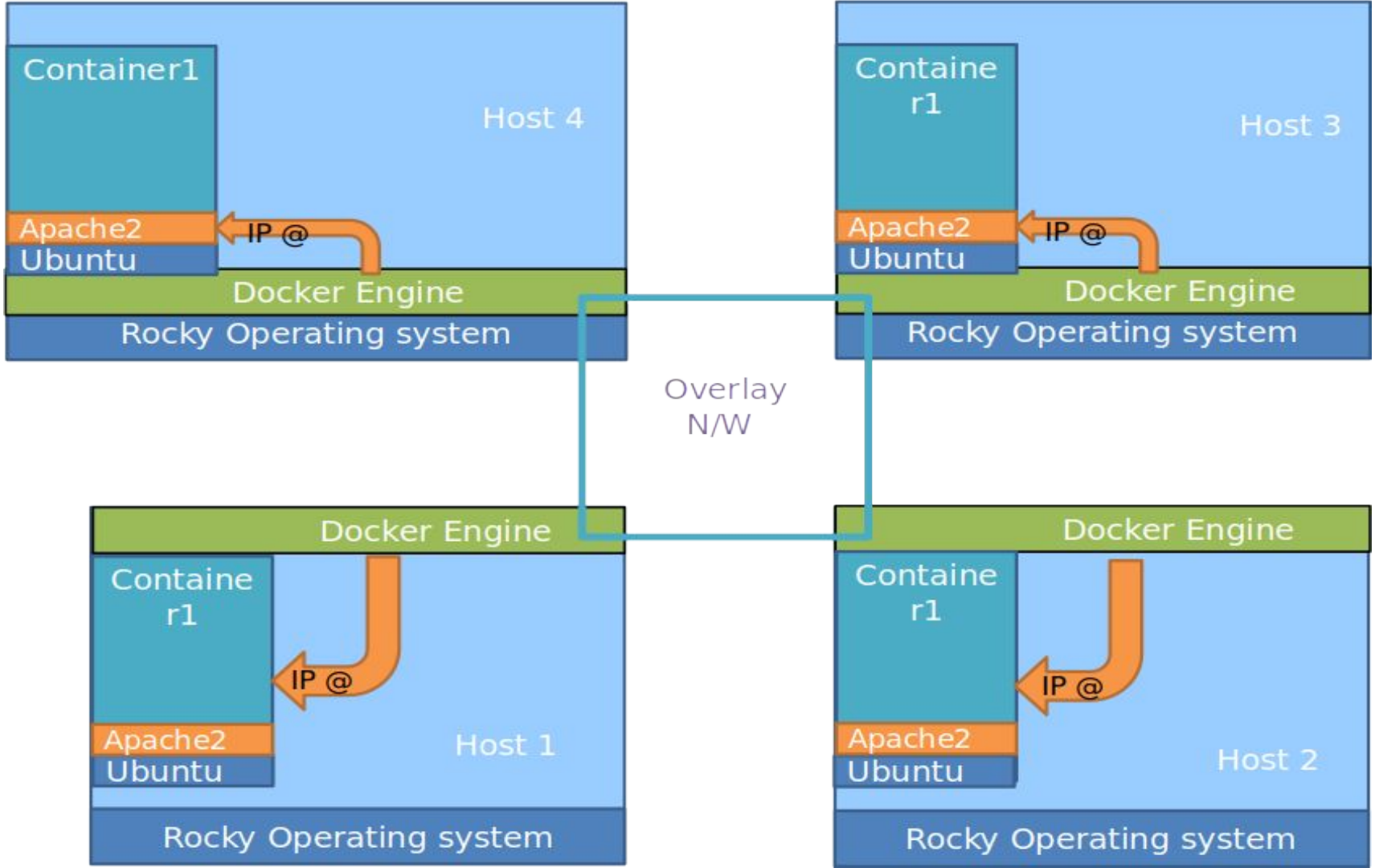
- Traffic allow between all containers.
- Can ping the container by IP @ only. Not by container name.
- There is no inbuilt DNS.

Custom Bridge Network

- Traffic allows only its attached containers.
- Can ping either container name or container IP.
- In built DNS enabled by default.

Overlay networks




Container1
Host 4
Apache2
Ubuntu
IP @
Docker Engine
Rocky Operating system
Containe
r1
Host 3
Apache2
Ubuntu
IP @
Docker Engine
Rocky Operating system
Overlay
N/W
Docker Engine
Containe
r1
IP @
Apache2
Ubuntu
Host 1
Rocky Operating system
Docker Engine
Containe
r1
IP @
Apache2
Ubuntu
Host 2
Rocky Operating system

If we have multiple Docker daemon host and want to add all of them in a single network driver, we may use Overlay Network.

The overlay network driver creates a distributed network among multiple Docker daemon hosts.

This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate securely when encryption is enabled.

Docker transparently handles routing of each packet to and from the correct Docker daemon host and the correct destination container.

Initialize the Swarm service on master node.

docker swarm init

On another host, how to join the swarm cluster?

docker swarm join

Once it done, you can create the Overlay network driver on master node.

How to create Overlay network ?

docker network create -d overlay my-overlay

Lab Overlay Network Driver

IPVLAN

- The IPvlan driver gives users full control over both IPv4 and IPv6 addressing.
- It means that user has control of layer 2 VLAN tagging and even IPvlan L3 routing for users interested in underlay network integration.
- This is the reason, IPVLAN is a true network virtualization technique.
- For other network drivers, we use subinterfaces called VERTH. In IPVLAN, we attach the container directly to the Docker host interface, such as eth0 and so on.

Lab IP VLAN Network Driver

MACVLAN

- Some applications which monitor the network traffic requires direct connection. Which means should have a unique MAC address.

- In this type of situation, you can use the macvlan network driver to assign a MAC address to each container’s virtual network interface, making it appear to be a physical network interface directly connected to the physical network.

- All the functionality of IPVLAN is adopted by MACVLAN. In Macvlan, all the containers will have dedicated MAC address for outside world.

- Thus, practical of MACVLAN is not necessary.

LAB Bridge network driver

https://github.com/cfitechops/Docker/blob/main/06-docker.md