

docker

Docker For Beginners



Sonu Madhesiya



Table of Contents

1. Introduction
 - 1.1. What is Docker?
 - 1.2. Why Use Docker?
 - 1.3. Docker vs. Virtual Machines
2. Getting Started
 - 2.1. Installing Docker
 - 2.2. Starting the Docker Service
3. Docker Basics
 - 3.1. Containers vs. Virtual Machines
 - 3.2. Running Your First Container
4. Docker Commands
 - 4.1. Docker Images
 - 4.2. Docker Containers
 - 4.3. Docker Volumes
 - 4.4. Docker Networking
 - 4.5. Docker Compose
5. Docker Volumes
6. Docker Networking
7. Pushing Docker Images to Docker Hub
8. Dockerfiles
 - 8.1. Sample Dockerfile Examples (Ubuntu, Apache2, Nginx, Node.js, React.js)
9. Basic Questions & Answers

Docker

To run the docker, you have to start the docker service by the below command:
sudo systemctl start docker

*Docker runs on Port 80. So, if you couldn't run docker on your system. Then definitely, there is some other service that will be running on Port 80 such as Nginx or apache2.
So, first stop that service which is running on Port 80 then, start the docker service.*

Theory

- Docker is a tool for running applications in an isolated environment.
- Like to VM

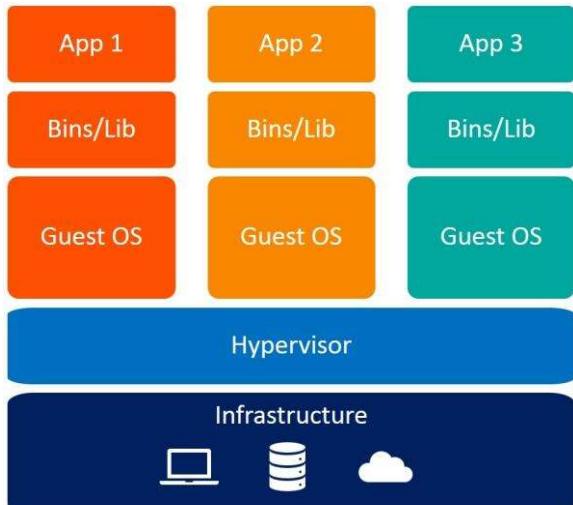
Containers

Containers are an abstraction at the app layer that package code and dependencies together. Multiple containers can run on the same machine and share the same OS kernel with other containers, each running an isolated process in userspace.

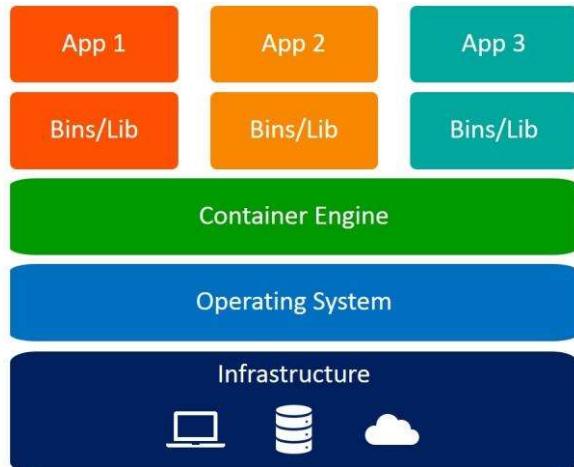


Virtual Machines

Virtual machines are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an OS, the app, necessary binaries and libraries taking up tens of GBs. VMs can also be slow to boot.



Virtual Machines



Containers

To Install and configure the Docker on Ubuntu 22.04, follow the below link:

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04>

Docker Commands



S.NO	Command	Use Case
Docker Images		
1.	docker pull <image_name>	To pull the docker image from the docker hub.
2.	docker images	To list all images excluding hidden intermediate images.
3.	docker images -a	To list only docker images id excluding hidden intermediate images.
4.	docker rmi <image_name>	To remove the specific docker image.
5.	docker history <tags:version>	To see the image layer of docker.
6.	docker image inspect <image_id>	To check all the information related to the image
7.	docker image prune	To delete all the images from the local server
8.	docker image history <image_id>	To see all the history related to the image
9.	docker image save <image_id> < <desired_image_name>.tar	To save the image and all the other information like history,etc excluding mounted volumes data.
10.	docker image load <desire_image_name>.tar	To get the image and all the other information too.
11.	docker image import <desired_image_name>.tar <new_desired_image_name>	To import the saved image from the container as an image.
12.	docker image push	To push the image to registry such as Dockerhub.

Docker Container

13.	<code>docker run -p 8080:80 <image_name>:<tag></code>	To run the container at the specific port number with the image tag.
14.	<code>docker run -d -p <image_name>:<tag></code>	To run the container at the specific port number with the image tag with detach mode.
15.	<code>docker ps</code>	To list all the running containers.
16.	<code>docker ps -a</code>	To list both all the running containers and stopped containers.
17.	<code>docker stop <container_id></code>	To stop the specific container.
18.	<code>docker start <container_id></code>	To start the specific container.
19.	<code>docker rm <container_id></code>	To remove the specific container.
20.	<code>docker rm \$(docker ps -aq)</code>	To remove all the containers. But can't remove running containers.
21.	<code>docker rm -f \$(docker ps -aq)</code>	To remove all the running containers and stopped containers.
22.	<code>docker run --name <container_id> -d -p 80:80 nginx:latest</code>	To give the name to the container.
23.	<code>docker exec -it <container_id> bash</code>	To go into the container.
24.	<code>docker container prune -f</code>	To remove all the stopped containers.
25.	<code>docker inspect <container_id></code>	To see all the details of the specific container.
26.	<code>docker logs <container_id></code>	To see all the logs of the specific container.

27.	docker logs -f <container_id>	To follow the logs or live logs.
28.	Ctrl+p+q	Whenever you exit from the container, the container stops and exits. So, to get rid of this, press these buttons when you are in the container.
29.	docker container top <container_id>	To list all the processes running inside the specific container.
30.	docker container stats	To see all the CPU utilization, Memory usage, Inputs, Outputs about every container.
31.	docker rename <container_id> <new_container_name>	To change the container name.
32.	docker container restart <container_id>	To restart the container
33.	docker container rename <container_id> <new_name>	To rename the container.
34.	docker container attach <container_id>	To attach to the container again after detaching it by -d argument.
35.	docker kill <container_id>	To kill the container.
36.	docker container wait <container_id>	Container will wait to kill or stop itself. Whenever the container will stop or kill then, it will notify the terminal by "0" status.
37.	docker container pause <container_id>	To pause the container.
38.	docker container unpause <container_id>	To unpause the container.
39.	docker container port <container_id>	To know the port mapping of the container like

		80/tcp -> 0.0.0.0:45 80/tcp -> :::45
40.	docker container create <container_name> <command>	To create a container only, It will not start the container. If you want to start the container then, use docker container start <container_name> command after this.
41.	docker container diff <container_id>	To observe the change in the container such as files creation or deletion, etc from the previous version. Note: to check/follow the file creations or deletion use the watch command: watch 'docker container diff <container_id>'
42.	docker container cp <source_file(from local)> <container_id:/directory>	To copy the files from the local to the container's file system. Such as docker container cp index.html 99316f94a56f:/usr/local
43.	docker container export <container_id> -o <desired_image_name>.tar	To save the running/stopped container by importing it as a docker image. If you want to import this image then, you have to use docker image export command.
44.	docker run --name <container_name> --volumes-from <old_container_name> -d -p 8081:80 nginx:latest	To share the container volume
45.	docker rm \$(docker stop \$(docker ps -aq))	The final output of this command is to remove the containers directly (Time Saving).
46.	docker container run -it --network <your_network_name> <image_name>	It will run the container and attach the network with the container as well.

Volumes

47.	docker volume create	To create the docker volume
48.	docker volume inspect	Get a detailed information of the docker volumes

49.	<code>docker volume ls</code>	To list all the docker volumes
50.	<code>docker container commit --author "Author Name" -m "Commit Description" <container_id> <desired_image_name></code>	To create the image from the running container.
51.	<code>docker run --name <container_name> -v \$(pwd) :/usr/share/nginx/html/:ro -d -p 8080:80 nginx:latest</code>	To link the file and send the file to the container volume.
52.	<code>docker run -d --name 1st -v voce:/var/lib/mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=true mysql</code>	It will create the docker volume too and run the container as well. So, you don't have to write two commands separately instead of doing both work in one individual command.
53.	<p><code>docker run -it -v /home/amanpathak/Documents/:/tmp/new/folder/ ubuntu:latest bash</code></p> <p style="text-align: center;"><i>OR</i></p> <p><code>docker run -it --mount type=bind,source=\$(pwd),target=/temp/upper/lower/ ubuntu bash</code></p>	<p>It will bind mount your folder into the docker container.</p> <p>If the destination folder is not present then, it will auto create the folders that are not created and If you update any file or folder on the local machine or host machine then, the same changes will happen on the docker container such as modify the file content.</p> <p>It will do the same thing as above command, but there are some changes that you can notice here.</p>

Networking

54.	<code>docker network inspect <network_id></code>	To get the all details regarding the docker network driver
55.	<code>docker network create -d <network_type> <your_network_name></code>	It will create the network with network driver type as well.
56.	<code>docker container run -it --network <your_network_name> <image_name></code>	It will run the container and attach the network with the container as well.

57.	docker container run -it --network=none <image_name>	It will run the container with a null docker network driver.
58.	docker network connect <network_name> <container_name>	To connect the container with th docker network
59.	docker network disconnect <network_name> <container_name>	To disconnect the container with th docker network
60.	docker network prune	To remove unused docker network
61.	docker rm <network_name>	To remove the specific docker network
62.	docker network ls	To list all the docker networks

Docker Compose

63.	docker compose build	Build or rebuild services
64.	docker compose convert	Converts the compose file to platform's canonical format
65.	docker compose cp	Copy files/folders between a service container and the local filesystem
66.	docker compose create	Creates containers for a service.
67.	docker compose down	Stop and remove containers, networks
68.	docker compose events	Receive real time events from containers.
69.	docker compose exec	Execute a command in a running container.
70.	docker compose images	List images used by the created containers
71.	docker compose kill	Force stop service containers.
72.	docker compose logs	View output from containers
73.	docker compose ls	List running compose projects
74.	docker compose pause	Pause services
75.	docker compose port	Print the public port for a port binding.
76.	docker compose ps	List containers

77.	docker compose pull	Pull service images
78.	docker compose push	Push service images
79.	docker compose restart	Restart containers
80.	docker compose rm	Removes stopped service containers
81.	docker compose run	Run a one-off command on a service.
82.	docker compose start	Start services
83.	docker compose stop	Stop services
84..	docker compose top	Display the running processes
85..	docker compose unpause	Unpause services
86.	docker compose up	Create and start containers
87.	docker compose version	Show the Docker Compose version information
88.	docker search <image_name>	Allows you to search for Docker images available on Docker Hub or other container registries.
89.	docker system prune	This command cleans up unused data such as containers, networks, and volumes. Useful for freeing up disk space.
90.	docker network inspect <network_name>	Provides detailed information about a specific Docker network, including its configuration and containers connected to it.
91.	docker-compose up -d	Starts containers defined in a Docker Compose file in detached mode, running them in the background.
92.	docker-compose ps -q	Lists only the container IDs of containers defined in a Docker Compose file.
93.	docker-compose stop	Stops containers defined in a Docker Compose file without removing them.
94.	docker-compose restart	Restarts containers defined in a Docker Compose file.

95.	docker-compose logs -f	Displays and follows the logs of all containers defined in a Docker Compose file in real-time.
96.	docker-compose exec <service_name> <command>	Executes a command in a specific service container defined in a Docker Compose file.
96.	docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' <container_id>	Retrieves the IP address of a running container.
97.	docker image prune -a	Removes all unused images, not just dangling ones.
98.	docker-compose down -v	Stops and removes containers defined in a Docker Compose file, including associated volumes.
99.	docker-compose logs <service_name>	Displays the logs of a specific service container defined in a Docker Compose file.
100.	docker-compose build <service_name>	Rebuilds a specific service defined in a Docker Compose file.
101.	docker-compose scale <service_name>=<number_of_instances>	Scales the number of containers for a specific service in a Docker Compose file.

Things you need to know about

Docker Volumes:

By default all files created inside a container are stored on a writable container layer

The data doesn't persist when that container is no longer running

A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.

Docker has two options for containers to store files in the host machine

so that the files are persisted even after the container stops

VOLUMES and **BIND MOUNTS**

Volumes are stored in a part of the host filesystem which is managed by Docker

Non-Docker processes should not modify this part of the filesystem

Bind Mounts may be stored anywhere on the host system

Non-Docker processes on the Docker host or a Docker container can modify them at any time

In Bind Mounts, the file or directory is referenced by its full path on the host machine.

Volumes are the best way to persist data in Docker

volumes are managed by Docker and are isolated from the core functionality of the host machine

A given volume can be mounted into multiple containers simultaneously.

When no running container is using a volume, the volume is still available to Docker and is not removed automatically. You can remove unused volumes using docker volume prune.

When you mount a volume, it may be named or anonymous.

Anonymous volumes are not given an explicit name when they are first mounted into a container

Volumes also support the use of volume drivers, which allow you to store your data on remote hosts or cloud providers, among other possibilities.

For the Docker Volume Demo, Go through the video

[Docker Volume Demo of MySQL](#)

Docker Networking

The default network driver in the docker network is bridge. But if you want the network driver as per your want, then you can create it and then attach it with the docker container.

There are multiple types of docker network drivers such as bridge, host, macvlan, ipvlan, overlay, none.

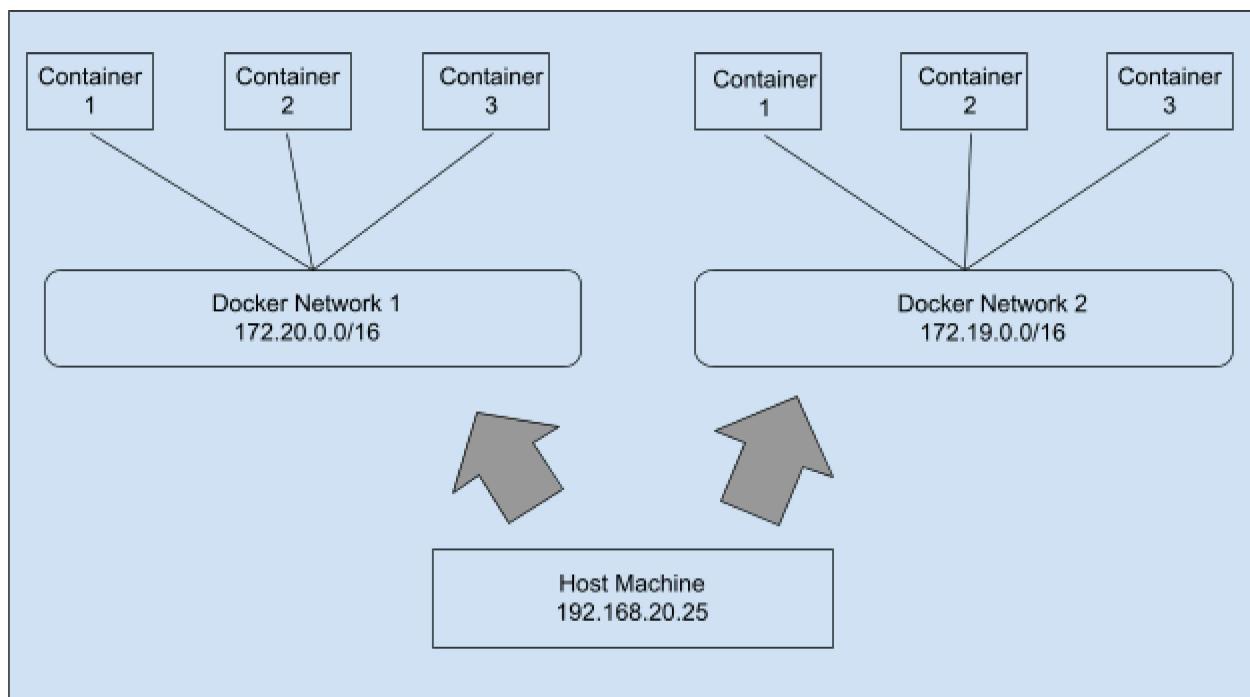
Docker takes care of the networking aspects so that the containers can communicate with other containers and also with the Docker Host. If you do an ifconfig on the Docker Host, you will see the Docker Ethernet adapter. This adapter is created when Docker is installed on the Docker Host.

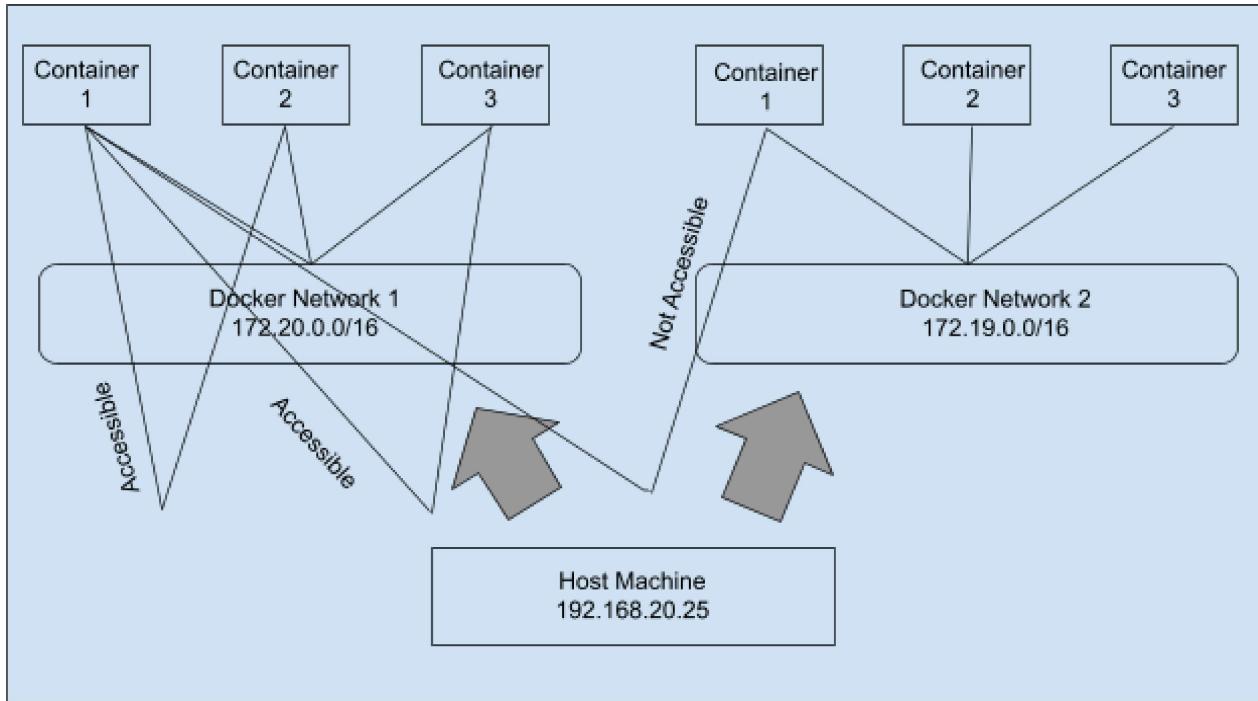
Docker Networking (Bridge Network)

`enp0s10: | en| | --> ethernet v | p0| --> bus number (0) v s10 --> slot number (10)`

This is a bridge between the Docker Host and the Linux Host.

Whenever you run a container, it by default attaches with a bridge network driver but if you want to attach with your desired network driver then, you can do it by the command, in which you have to specify the network name (assuming that network is already created by you).

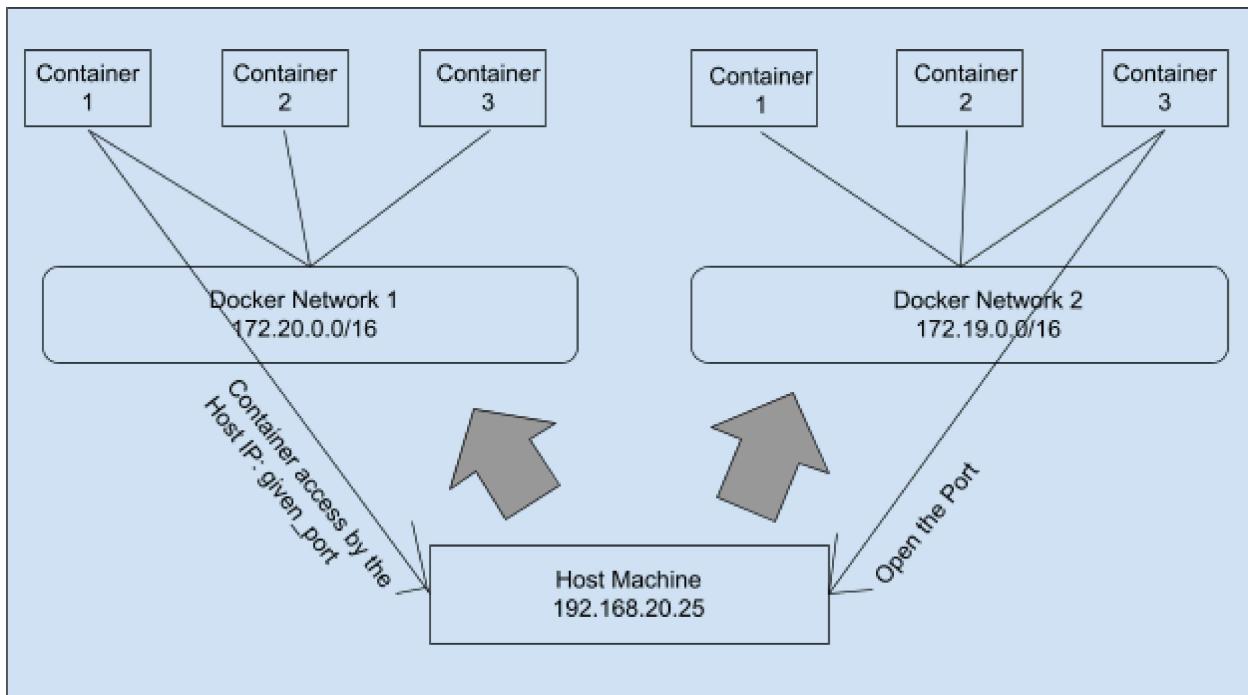




Docker Container 1 under the Docker network 1 can access or communicate with Docker Container 2 and Docker Container 3. Also Every Docker Container can communicate with any other Docker Container, if that Container is under the same Docker network.

But, any docker container of docker network 1 can't communicate with any docker container of docker network 2 because the network is not the same.

So, to access or communicate with other docker network's containers you can do it with the help of port mapping.



-> Create two network for Docker containers

```
root@aman-pathak:/home/amanpathak# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
03df63b2be07   bridge    bridge      local
e84e86cd8a7f   host      host       local
97e8dd2e9ca6   none     null      local
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak# docker network create -d bridge network1
97e8dd2e9ca6ca66577bb13b73d423c14b6cdde363ed3712d95f399a2412b03f562
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
03df63b2be07   bridge    bridge      local
e84e86cd8a7f   host      host       local
97e8dd2e9ca6   network1  bridge      local
97e8dd2e9ca6   network2  bridge      local
3b2f15468018   none     null      local
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak# docker network create -d bridge network2
17e67de39205e14b6c5273963d2853968865141345d368b7f4e45fc4d46ab0
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
03df63b2be07   bridge    bridge      local
e84e86cd8a7f   host      host       local
97e8dd2e9ca6   network1  bridge      local
17e67de39205e  network2  bridge      local
3b2f15468018   none     null      local
root@aman-pathak:/home/amanpathak#
```

-> Create two Docker Containers, First for ubuntu and other one for nginx(with port open by -P argument)

```
root@aman-pathak:/home/amanpathak# docker container run -itd --network network1 ubuntu
d53a47c6a2d66eb86eb24570f16a1a5eb3ba95a5edsec3e9da436bf6e092719
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
d53a47c6a2d6   ubuntu             "bash"              4 seconds ago     Up 3 seconds          0.0.0.0:22->22/tcp   gracious_ganguly
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak# docker container run -itd -P --network network2 nginx
611fe923dc51d0698cf8950cd931024bee86c0e3d68580697d56ade970f777f2
root@aman-pathak:/home/amanpathak#
root@aman-pathak:/home/amanpathak# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
611fe923dc51   nginx             "/docker-entrypoint..." 4 seconds ago     Up 3 seconds          0.0.0.0:49157->80/tcp   thirsty_black
d53a47c6a2d6   ubuntu             "bash"              17 seconds ago    Up 16 seconds         0.0.0.0:80->80/tcp   gracious_ganguly
root@aman-pathak:/home/amanpathak#
```

-> details of nginx docker container, Remember the IP

```
{"MacAddress": "",  
 "Networks": {  
     "network1": {  
         "IPAMConfig": null,  
         "Links": null,  
         "Aliases": [  
             "d53a47c6a2d6"  
         ],  
         "NetworkID": "97e8dd2e9ca66577bb13b73d423c14b6cdde363ed3712d95f399a2412b03f562",  
         "EndpointID": "2a2541639e4c1398e833bde43379a9361f385db5fd130eb543098bc2340ab4c6",  
         "Gateway": "172.25.0.1",  
         "IPAddress": "172.25.0.2",  
         "IPPrefixLen": 16,  
         "IPv6Gateway": "",  
         "GlobalIPv6Address": "",  
         "GlobalIPv6PrefixLen": 0,  
         "MacAddress": "02:42:ac:19:00:02",  
         "DriverOpts": null  
     }  
 },  
 }
```

-> details of ubuntu docker container, Remember the IP

```

    "MacAddress": "",
    "Networks": {
        "network2": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": [
                "611fe923dc51"
            ],
            "NetworkID": "17e67de39205e14b6c8c5273963d2853968865141345d368b7f4e45fc4d6ab0",
            "EndpointID": "c78630565242597332423b912dae8eed98935c71591c86e863b62d9a6bfe275e",
            "Gateway": "172.26.0.1",
            "IPAddress": "172.26.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:1a:00:02",
            "DriverOpts": null
        }
    }
}

```

-> Remember the IP of the host machine

```
wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.0.225 netmask 255.255.254.0 broadcast 192.168.1.255
inet6 fe80::d2bb:dc7:cled:e178 prefixlen 64 scopeid 0x20<link>
ether 4c:d5:77:79:02:df txqueuelen 1000 (Ethernet)
RX packets 440190 bytes 479953176 (479.9 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 184926 bytes 74578398 (74.5 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

-> Check that Port is open or Not.



-> enter in the ubuntu container and use wget to check that the container will communicate with another network's container or not.

But as you can see, the wget is not installed then install it first.

```

root@aman-pathak:/home/amanpathak# docker container exec -it gracious_ganguly bash
root@53a47c6a2d6:/#
root@53a47c6a2d6:/#
root@53a47c6a2d6:/#
root@53a47c6a2d6:/#
root@53a47c6a2d6:/# wget 192.168.0.225
sh: wget: command not found
root@53a47c6a2d6:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu jammy InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [4644 B]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [344 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [322 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [132 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [342 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy-haarkorts InRelease [99.8 kB]

root@53a47c6a2d6:/# apt-get install wget
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:

```

-> Now, enter the command wget:<host_machine_ip>

```
root@d53a47c6a2d6:/# wget 192.168.0.225
--2022-08-26 12:44:59--  http://192.168.0.225/
Connecting to 192.168.0.225:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18934 (18K) [text/html]
Saving to: 'index.html'

index.html          100%[=====] 18.49K --.-KB/s   in 0s

2022-08-26 12:44:59 (128 MB/s) - 'index.html' saved [18934/18934]
root@d53a47c6a2d6:/#
```

You can see here, that the different docker network container is accessible and can communicate with the different docker network container.

You can also ping or access the docker container within the same network by the hostname and container name as well.

- I. Create a network
- II. Create two container, both must be running in detach mode
- III. Go in the first container and copy the hostname of it by the hostname command.
- IV. Go in the second container and run the command ping <hostname_container1>

The Output will look like this:

```
root@f8c13edd84dc:/# ping 8e05f02c4b07
PING 8e05f02c4b07 (172.27.0.2) 56(84) bytes of data.
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=1 ttl=64 time=0.228 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=3 ttl=64 time=0.151 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=4 ttl=64 time=0.157 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=5 ttl=64 time=0.156 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=6 ttl=64 time=0.152 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=7 ttl=64 time=0.151 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=8 ttl=64 time=0.153 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=9 ttl=64 time=0.150 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=10 ttl=64 time=0.151 ms
64 bytes from 8e05f02c4b07.one (172.27.0.2): icmp_seq=11 ttl=64 time=0.152 ms
```

a host type network driver works as a host.

When you launch a container using a host network driver, then the network will not be in an isolated environment, but the rest of things apart from the network driver will be in an isolated environment.

When you want to run the docker container with no network driver then, it is a null network driver.

We cannot attach any network to a container that has none network already attached and we cannot attach any network if the container is having a host network attach(host cannot be connected to any other network).

Dockerhub Push commands

This command will build the docker image

docker build --tag ubuntu-mine:latest .

This command will give the tag to the docker image

docker tag ubuntu-mine:latest avian19/custom-ubuntu:latest

This command will push the docker image to Dockerhub/ECR/ACR

docker push avian19/custom-ubuntu:latest

Dockerfiles

The use Of ENTRYPPOINT

```
root@aman-pathak:/home/amanpathak/Images# cat Dockerfile
FROM ubuntu:16.04
RUN apt-get update && mkdir -p /new/true/
COPY test.sh /new/true/
ENTRYPOINT ["/new/true/test.sh"]
```

```
root@aman-pathak:/home/amanpathak/Images# cat test.sh
#!/bin/bash
echo "Hey! This is $1"
```

```
root@aman-pathak:/home/amanpathak/Images# docker build --tag ubuntu-new .
```

```
root@aman-pathak:/home/amanpathak/Images# docker run -it ubuntu-new
Hey! This is
root@aman-pathak:/home/amanpathak/Images# docker run -it ubuntu-new Aman Pathak
Hey! This is Aman
root@aman-pathak:/home/amanpathak/Images# docker run -it ubuntu-new Aman
Hey! This is Aman
```

Ubuntu and Some other Configurations

```
FROM ubuntu:latest
RUN apt-get update -y
RUN apt-get install nginx -y
RUN apt-get install -y apt-utils
RUN service nginx start
RUN groupadd -r user && useradd -r -g user user
RUN mkdir youtube
RUN chown user:user youtube
RUN chown user:user youtube && touch test123.txt
```

```
WORKDIR app/  
  
RUN echo "This is a text file" > abc.txt  
  
CMD echo "This is For Sure"  
  
USER user  
  
EXPOSE 80
```

Apache2/httpd

```
FROM httpd:latest  
  
COPY . /usr/local/apache2/htdocs/  
  
EXPOSE 80
```

Nginx

```
FROM nginx:1.23.0-alpine  
  
COPY . /usr/share/nginx/html/  
  
EXPOSE 80
```

Nodejs

```
FROM node:16-alpine  
  
WORKDIR app/  
  
ADD ./package*.json ./  
  
RUN npm install  
  
ADD ..  
  
CMD node index.js
```

```
EXPOSE 3000
```

Reactjs

```
FROM node:16-alpine
```

```
WORKDIR app/
```

```
ADD package*.json ./
```

```
RUN npm install --silent
```

```
RUN npm install react-script@3.4.1 -g --silent
```

```
CMD [“npm”, “start”]
```

```
EXPOSE 3000
```