Docker

A. Docker Overview
   1. Docker Image
   2. Docker Container
   3. Docker network
   4. Docker volume
   5. Docker Register
B. Docker file
C. Docker compose
   - Microservice with docker file /docker compose

D. Docker swarm (do youself – turorial is mentioned on reference link)
E. Reference

Documented by : Anil Gupta

*A Docker Overview

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

The Docker platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

- Your developers write code locally and share their work with their colleagues using Docker containers.
- They use Docker to push their applications into a test environment and execute automated and manual tests.
- When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
- When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Responsive deployment and scaling

Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments. Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

Running more workloads on the same hardware

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

Docker architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

The Docker daemon

The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (`docker`) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to `dockerd`, which carries them out. The `docker` command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the `docker pull` or `docker run` commands, the required images are pulled from your configured registry. When you use the `docker push` command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

*1.DOCKER IMAGES

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the `ubuntu` image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.
You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

sno	**[IMAGE]** Command	description
1	Docker image ls / Docker images →list all docker images	

```
anil@DESKTOP-0J0O329:/$ docker image ls
REPOSITORY                          TAG       IMAGE ID       CREATED         SIZE
dockerhub825313/ubuntu_21_04_tree_git   latest    8b45f347f70e   10 hours ago    225MB
image_ubuntu_21.04_tree_git         latest    8b45f347f70e   10 hours ago    225MB
image_crt_with_running_container    latest    adbfdd488311   17 hours ago    72.9MB
ubuntu_imported_image               latest    8014e4d46765   19 hours ago    201MB
```

2	Docker image ls –format '{{.ID}} , {{.Repository}}' →output in comma separated format	

```
anil@DESKTOP-0J0O329:/$ docker image ls --format '{{.ID}},{{.Repository}}'
8b45f347f70e,dockerhub825313/ubuntu_21_04_tree_git
8b45f347f70e,image_ubuntu_21.04_tree_git
adbfdd488311,image_crt_with_running_container
8014e4d46765,ubuntu_imported_image
```

3	Docker image history [image id]	

```
anil@DESKTOP-0J0O329:/$ docker image history 8b45f347f70e
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
8b45f347f70e   10 hours ago   /bin/bash                                       145MB     this is test commit
1fc773f9e714   6 weeks ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      6 weeks ago    /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>      6 weeks ago    /bin/sh -c [ -z "$(apt-get indextargets)" ]     0B
<missing>      6 weeks ago    /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>      6 weeks ago    /bin/sh -c #(nop) ADD file:e40576843421cb419…   80.7MB
```

4	Docker image rm -f [images id/names] → remove image –f (if any container associated with this image force to remove)	

```
anil@DESKTOP-0J0O329:/$ docker image rm -f 8b45f347f70e 8b45f347f70e adbfdd488311
Untagged: dockerhub825313/ubuntu_21_04_tree_git:latest
Untagged: dockerhub825313/ubuntu_21_04_tree_git@sha256:9b223e7986374d93b1ca2dd865f970d9fe21e1b7200ad7ed2b6d4f11744d44ce
Untagged: image_ubuntu_21.04_tree_git:latest
Deleted: sha256:8b45f347f70e10b23c65c83675476eb9415208dfb5298e71a7e5cc110a2760f6
Deleted: sha256:d310304b266c35d3879c9e530777b3e6774545402c78d4c44f012689f19ceb9b
Untagged: image_crt_with_running_container:latest
Deleted: sha256:adbfdd4883119a258027a1969fd06979d11e625fe28319717229f8f96cad46d9
```

5	Docker image inspect [image name/image id]	

```
anil@DESKTOP-0J0O329:/$ docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
ubuntu_imported_image   latest    8014e4d46765   19 hours ago    201MB
nginx                   latest    35c43ace9216   2 weeks ago     133MB
ubuntu                  21.04     1fc773f9e714   6 weeks ago     80.7MB
ubuntu                  latest    f63181f19b2f   6 weeks ago     72.9MB
hello-world             latest    bf756fb1ae65   14 months ago   13.3kB
anil@DESKTOP-0J0O329:/$ docker image inspect 1fc773f9e714
[
    {
        "Id": "sha256:1fc773f9e71401f36640ac9f4951e45ca65291d2cbde59e3fb01e34faf586840",
        "RepoTags": [
            "ubuntu:21.04"
        ],
        "RepoDigests": [
```

6	Docker image prune →**[it will remove all unused images , use carefully]**	

7	**Save image into tar file** Docker image save [image] >[image name].tar → [first login as a super/root user]	

```
anil@DESKTOP-0J0O329:/$ sudo -s
root@DESKTOP-0J0O329:/# sudo docker image save nginx > nginx.tar
root@DESKTOP-0J0O329:/# ls -lh
```

```
drwx------   2 root root  16K Apr 10  2019 lost+found
drwxr-xr-x   2 root root 4.0K Feb 20 05:18 media
drwxr-xr-x   5 root root 4.0K Mar  1 17:48 mnt
-rw-r--r--   1 root root 131M Mar  4 12:18 nginx.tar
drwxr-xr-x   2 root root 4.0K Feb 20 05:18 opt
dr-xr-xr-x 139 root root    0 Mar  3 15:51 proc
```

8	**Load tar file image**
	Docker image load < [image tar file]
	`root@DESKTOP-0J0O329:/# docker image load < nginx.tar`
	`Loaded image: nginx:latest`

9	**Docker image save VS docker container export** **Save → it will contains all info of an image like tag, versions,layers etc** (docker image save ubuntu:16.04) -> only 16.04 version of tar file will be saved in one tar file (docker image save ubunt) -> it will save all versions of ubuntu images in one tar file **Export →it will create image tar from running container (it will contains only basic info)**
10	**Docker image load Vs docker image import** Load -> it will contain all info of an image Import -> it will contain only basic info of an image

*2. DOCKER CONTAINERS

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

1. Container

	Command (moduler)	Desc
1	docker image ls	List of docker images
2	docker container ls	List running container
3	docker container ls -a	List all container running/stoped
4	docker container run [image name] or docker container run ubuntu [image name]=/ubuntu/linux/nginx etc Or Docker container run ubuntu sleep 30	Create and run container of [image type] os , it will stop after creation of container Create and run container for 30 second
5	docker container rm [container-id]	Remove container
6	Docker container rm [c-id1 c-id2]	Remove multiple container ,id separated by space
7	docker container start [container - id]	Start container
8	docker container stop [container-id]	Stop container
9	Docker container run –d [image name] [command] Docker container run –d ubuntu /bin/bash	Detach container (run container in back-ground)
10	Docker container –it [image] [command] Ex: -it(interactive terminal) Docker container run –it ubuntu /bin/bash	Go inside a container ,and work like a separate os, you can install software as well, to exit from container cell =>type Exit , now you will be in main cell, after typing exit container will close.
11	Docker container run 0–it ubuntu /bin/bash Now press CTRL+pq	To keep container running and exit from container cell Press ctrl+pq
12	docker container ls –aq	List all docker container id
13	docker container rm $(docker container ls –aq)	Delete all docker container
14	docker container inspect [running id] ex: (run container in background) docker container run -d nginx 297ba92abf….. Docker container inspect 297ba92abf…..	Inspect all info about running container ,port,ip….. We can check container server is running or not in browser ■ Add ipadress to browser
15	Docker container logs [container id]	It will show logs of running container
16	docker container top 297ba92abf44	Show what are the processes running of the container , Container is not a vm , it only runs the processes on top of Host OS.OS treat it as a separate process.
17	Docker container stats	It will show all running containers state,cpu uses,memories..
18	docker container run -d -p 3600:80 --name test_web nginx docker container run –d –P –name Test_web nginx ---------- note: -P will assign dynamic port(any available port) ex: netstat –nltp (to check running port in local ubuntu)	

(PORT mapping)We can redirect local request to , any container server
-p 3600:80 => map local 3600 port with containers 80 port
--name => any specific name of container.
=> hit browser http://172.18.78.151:3600/

172.18.78.151=>local ubuntu system ip and 3600 port

```
anil@DESKTOP-0J0O329:/$ docker container run -it -P nginx bash
```

-P will attach dynamic host port with container port

```
anil@DESKTOP-0J0O329:/$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                    NAMES
927d71227ccd   nginx     "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    0.0.0.0:49155->80/tcp    determi
```

Ex: open browse and type localhost/ ipaddress :port

192.168.134.81:49156 **192.168.134.81=localhost (host machine ip)**

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

localhost:49156

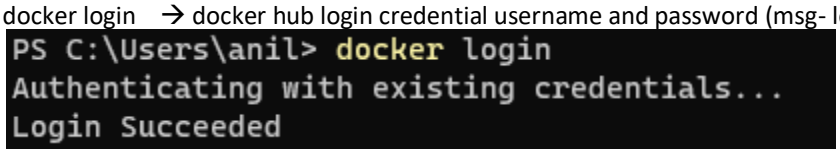
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

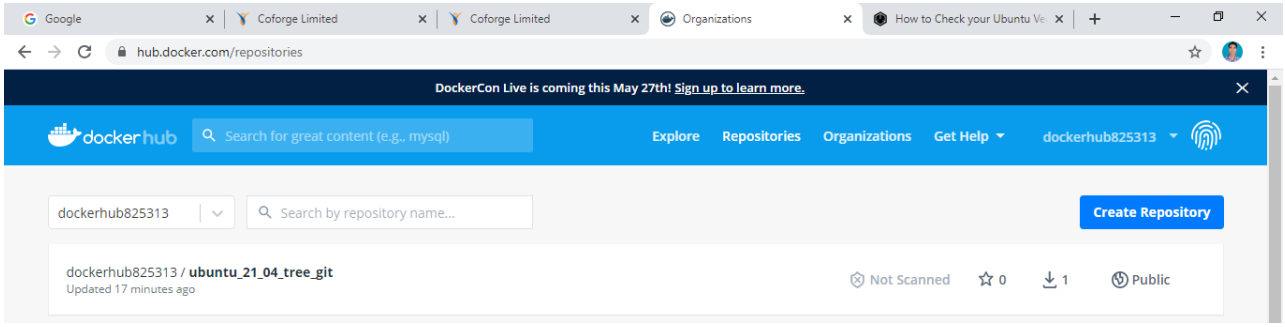
19	Docker container exec –it [container-id] /bin/bash	Excecute /bin/bash cell into container / or get cell access of current container, Now you can install softwares into containers
20	docker container rename [container –id] [name]	Rename container
21	Docker container restart [container-id]	Restart container
22	docker container kill [container- id]	Force stop container , or kill
23	docker container wait [container-id]	Waitting state , for stop container , if container will stop then it will leave docker cell with int (0) return msg.
24	Docker container pause [container-id]	State of container will change form running to pausesd, Now we can not able to access from out side of browser.
25	Docker container unpause [container -id]	Unpause container
26	Docker container prune –f	Prune will delete , all container which is not in use. Only running will remains (-f will not show warnings)
27	Docker container port [container –id/name]	Show container port mapping with parent OS.
28	Docker container create [image name]	It will create pull image and create container . It will not start, during creation.

29	docker container diff [container-id] watch 'docker container diff [container id]'	It will show the details of created, updated and deleted files and folders Watch is linux command which is used to watch container activity in every 2.0s
30	**Export container** • docker container run -it ubuntu /bin/bash #[→create container and inter in container cell] • sudo apt-get update #[→ update container softwares] • apt-get install tree git –y #[→ install tree and git in container] • ctrl+pq #[→ exit cell with running container] • docker container ls • **docker container export [container –id] >ubuntu_g_t.tar** • ls –lh → list files , and it will show create tar file	
31	**Import container (.tar) and run in docker** • Docker image ls • **Docker image import ubuntu_g_t.tar [image name]** → Docker image import ubuntu_g_t.tar **imported_ubuntu_image** • Docker image ls → now it will contain imported_ubuntu_image • Docker container run –it imported_ubuntu_image /bin/bash → run imported ubuntu_image and give cell • Git –version → check git is there or not	
32	**Creating an image from running container (and stop and remove running container , start container with created image and test)** • docker container run -i -t ubuntu /bin/bash →create and run container give cell to container • cd /tmp/ → go to tmp directory • touch 1 2 3 4 5 6 → create files on tmp • ls –lh → list all files of tmp • ctrl+pq → exit cell with running container • docker container ls →list available running container • docker container commit --author "Anil Gupta" -m "this is test commit" c7a981d8784d image_crt_with_running_container → create image from running container • docker container ls → list running container • docker container rm -f c7a981d8784d → stop and remove running container • docker container ls →list running container (check removed or not) • docker container run -it image_crt_with_running_container /bin/bash → run container with created img • cd tmp/ → go to temp directory • ls –lh → check files are there or not	
33	**Pull and push docker images from dockerhub (default image path docker.io)** • Search images in dockerhub (example **docker.io/**ubuntu 21.04 v) • **docker pull ubuntu:21.04** • docker run –it ubuntu:21.04 /bin/bash → run docker container with ubuntu_image_21.04 v • apt-get update • apt-get install tree git –y • ctrl+pq • docker container commit --author "Anil Gupta" -m "this is test commit" c7a981d8784d image_ubuntu_21.04_tree_git • docker image ls • docker image tag [image_name] [dockeruser]/[image_name] ex: docker image tag image_ubuntu_21.04_tree_git dockerhub825313/ubuntu_21_04_tree_git • docker login → docker hub login credential username and password (msg- login succeeded) •  • docker container ls	

```
PS C:\Users\anil> docker image ls
REPOSITORY                              TAG       IMAGE ID       CREATED         SIZE
dockerhub825313/ubuntu_21_04_tree_git   latest    8b45f347f70e   7 minutes ago   225MB
image_ubuntu_21.04_tree_git             latest    8b45f347f70e   7 minutes ago   225MB
image_crt_with_running_container        latest    adbfdd488311   7 hours ago     72.9MB
ubuntu_imported_image                   latest    8014e4d46765   9 hours ago     201MB
nginx                                   latest    35c43ace9216   2 weeks ago     133MB
ubuntu                                  21.04     1fc773f9e714   5 weeks ago     80.7MB
ubuntu                                  latest    f63181f19b2f   5 weeks ago     72.9MB
hello-world                             latest    bf756fb1ae65   14 months ago   13.3kB
```

- **docker push dockerhub825313/ubuntu_21_04_tree_git** →start pushing to dockerhub

```
PS C:\Users\anil> docker push dockerhub825313/ubuntu_21_04_tree_git
Using default tag: latest
The push refers to repository [docker.io/dockerhub825313/ubuntu_21_04_tree_git]
673b54e36c6b: Pushed
aa8011ef7554: Mounted from library/ubuntu
befc9ff11fa7: Mounted from library/ubuntu
c1a5fe2e50c7: Mounted from library/ubuntu
latest: digest: sha256:9b223e7986374d93b1ca2dd865f970d9fe21e1b7200ad7ed2b6d4f11744d44ce size: 1155
```

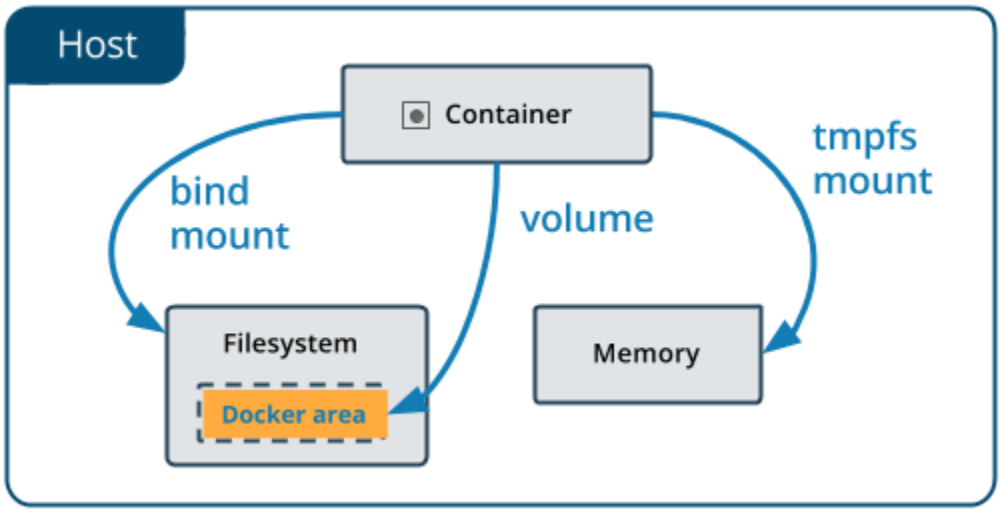


*3. Docker Volume

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
- New volumes can have their content pre-populated by a container.
- Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windows hosts.

In addition, volumes are often a better choice than persisting data in a container's writable layer, because a volume does not increase the size of the containers using it, and the volume's contents exist outside the lifecycle of a given container.



Docker (Volume)

1. Docker volume create volumeTest;
2. Docker volume ls;
3. Docker volume rm [volume name]
4. Docker volume prune → [it will remove all unused volume → if volume is associated with container then if need to remove that container before remove]
5. **docker volume** rm $(**docker volume** ls -q) →remove all volumes
6. **docker volume inspect [volume name]**

Dockerfile (**persist data using 1. Volume/ 2. bind mount / 3.tmpfs mount**)
https://docs.docker.com/storage/

1. Docker volume ls

```
root@DESKTOP-0J0O329:/# docker volume ls
DRIVER    VOLUME NAME
```

2. docker pull mysql

```
root@DESKTOP-0J0O329:/# docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
45b42c59be33: Already exists
b4f790bd91da: Pull complete
```

3. docker image inspect mysql (for mysql image docker will create volumes inside local var/lib/mysql after running container)

```
"Volumes": {
    "/var/lib/mysql": {}
},
```

4. Docker container run –d –name mysql –e MYSQL_ALLOW_EMPTY_PASSWORD=true mysql

```
root@DESKTOP-0J0O329:/# docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=true mysql
174fb4faeaee7edd17d0ca3656107b27f007cc8e9f4a67e04c307ad6b4ef1e28
```

5. Docker volume ls (every time new volume will be created for new container instance)

```
root@DESKTOP-0J0O329:/# docker volume ls
DRIVER    VOLUME NAME
local     c8e6c8442157b28bbb57d6e67577180bfe0801a149f1ba8b20770cdfd494115d
```

6. Docker volume inspect [volume name]

```
root@DESKTOP-0J0O329:/# docker volume inspect c8e6c8442157b28bbb57d6e67577180bfe0801a149f1ba8b20770cdfd494115d
[
    {
        "CreatedAt": "2021-03-06T09:34:48Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/c8e6c8442157b28bbb57d6e67577180bfe0801a149f1ba8b20770cdfd494115d/_data",
        "Name": "c8e6c8442157b28bbb57d6e67577180bfe0801a149f1ba8b20770cdfd494115d",
        "Options": null,
        "Scope": "local"
    }
]
```

7. Docker container exec –it [container id] bash (**go inside running container**)

```
root@DESKTOP-0J0O329:/# docker container exec  -it 174fb4faeaee bash
root@174fb4faeaee:/#
```

8. go inside mysql

```
root@174fb4faeaee:/# mysql
```

9. show databases ;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

10. create database Test1 (and show databases)

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| Test1              |
| Test2              |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.00 sec)
```

11. exit and delete container

```
root@DESKTOP-0J0O329:/# docker container rm -f 174fb4faeaee
174fb4faeaee
```

12. Docker container run –d –name mysql –e MYSQL_ALLOW_EMPTY_PASSWORD=ture mysql
    (create new container)

```
root@DESKTOP-0J0O329:/# docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=true mysql
ca691ee910817af6b296c3916a966e684d819d96de394ef3c40bcccf1b97dd42
```

13. Docker volume ls (both instance has separate volume and new one will start with fresh)

```
root@DESKTOP-0J0O329:/# docker volume ls
DRIVER    VOLUME NAME
local     2d47791ab58933a23e294ef8825db08b798478595c9716355f73d317ebe8edfe
local     c8e6c8442157b28bbb57d6e67577180bfe0801a149f1ba8b20770cdfd494115d
```

14. Docker exec –it [container id] bash

```
root@DESKTOP-0J0O329:/# docker exec -it ca691ee91081 bash
root@ca691ee91081:/#
```

15. mysql databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

16. docker container inspect [container –id] ->[inside mount, we will find volume name associated with container]

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "2d47791ab58933a23e294ef8825db08b798478595c9716355f73d317ebe8edfe",
        "Source": "/var/lib/docker/volumes/2d47791ab58933a23e294ef8825db08b798478595c9716355f73d317ebe8edfe/_data",
        "Destination": "/var/lib/mysql",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

17. **now** create new container and **attach previous volume** with image , and tell container to stop creating new volume but use existing with previous existing data.

    Docker container run –itd –v [volume1]:/var/lib/mysql mysql (if volume will not exist than it will create new volume with name volume1)

    docker container run -itd –v [volume name]:/var/lib/mysql mysql

```
root@DESKTOP-0J0O329:/# docker container run -itd -v c8e6c8442157b28bbb57d6e67577180bfe0801a149f1ba8b20770cdfd494115d:/var/lib/mysql mysql
86ed96afe9c4df2cc429feaa7f6184daf006874d10611ece66dc4901605183e3
```

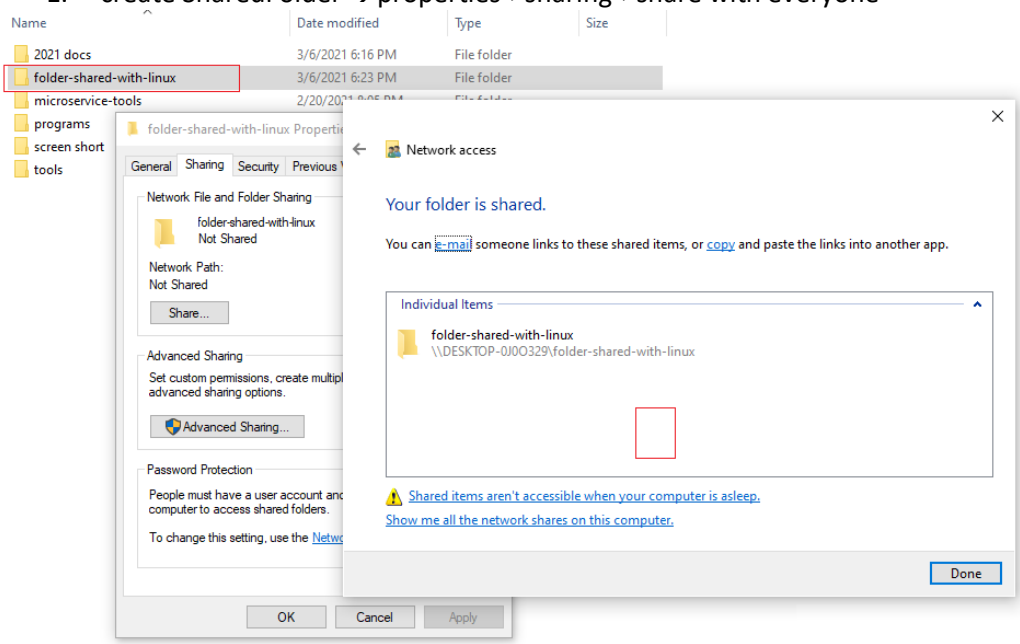
18. go inside container and check mysql databases
    docker exec –it [container id] bash (type mysql)
19. show databases; (we can see associate volumes contains previous database records)

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| Test1              |
| Test2              |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.01 sec)
```

Bind mount (we can mount , host machine file and folder to container)

Share window file with linux

1. create SharedFolder →properties->sharing->share with everyone



2. sudo apt install cifs-utils

```
root@DESKTOP-0J0O329:/# sudo apt install cifs-utils
```

3. Sudo mkdir /share →create forlder in linux

```
root@DESKTOP-0J0O329:/# sudo mkdir /share
```

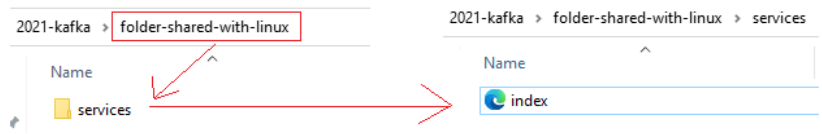
4. Check ip-address of window (wls)

```
Ethernet adapter vEthernet (Default Switch):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::c4f2:c8c9:b9c2:970f%16
   IPv4 Address. . . . . . . . . . . : 192.168.134.81
   Subnet Mask . . . . . . . . . . . : 255.255.255.240
   Default Gateway . . . . . . . . . :

Ethernet adapter vEthernet (WSL):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::b1b4:62e7:4076:7eb0%50
   IPv4 Address. . . . . . . . . . . : 192.168.25.113
   Subnet Mask . . . . . . . . . . . : 255.255.255.240
   Default Gateway . . . . . . . . . :
```

5. sudo mount.cifs //192.168.25.113/folder-shared-with-linux /share -o user=anil

```
root@DESKTOP-0J00329:/# sudo mount.cifs //192.168.25.113/folder-shared-with-linux /share -o user=anil
```

6. check




```
anil@DESKTOP-0J00329:/$ ls
Dockerfile  boot  etc   init  lib32  libx32      media  nginx.tar  proc  run   share  srv  test  usr
bin         dev   home  lib   lib64  lost+found  mnt    opt        root  sbin  snap   sys  tmp   var
anil@DESKTOP-0J00329:/$ cd share
anil@DESKTOP-0J00329:/share$ ls
services
```

2. create container with bind mount (shared folder as volume)

1. **docker container run -it -v /share:/tmp/test/ ubuntu:21.04 bash**

[it will create test folder inside temp and mount share folder on test]

```
root@DESKTOP-0J00329:/# docker container run -it -v /share:/tmp/test/ ubuntu:21.04 bash
root@c17a1d36d269:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

2. check (cd tmp/test)

```
root@c17a1d36d269:/tmp# cd test
root@c17a1d36d269:/tmp/test# ls
services
root@c17a1d36d269:/tmp/test# cd services
root@c17a1d36d269:/tmp/test/services# ls
index.html
```

*4 Docker Network

One of the reasons Docker containers and services are so powerful is that you can connect them together, or connect them to non-Docker workloads. Docker containers and services do not even need to be aware that they are deployed on Docker, or whether their peers are also Docker workloads or not. Whether your Docker hosts run Linux, Windows, or a mix of the two, you can use Docker to manage them in a platform-agnostic way.

This topic defines some basic Docker networking concepts and prepares you to design and deploy your applications to take full advantage of these capabilities.

Network drivers

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

- `bridge`: The default network driver. If you don't specify a driver, this is the type of network you are creating. **Bridge networks are usually used when your applications run in standalone containers that need to communicate.** See bridge networks.
- `host`: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly. See use the host network.
- `overlay`: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers. See overlay networks.
- `macvlan`: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the `macvlan` driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack. See Macvlan networks.
- `none`: For this container, disable all networking. Usually used in conjunction with a custom network driver. `none` is not available for swarm services. See disable container networking.
- Network plugins: You can install and use third-party network plugins with Docker. These plugins are available from Docker Hub or from third-party vendors. See the vendor's documentation for installing and using a given network plugin.

Network driver summary

- **User-defined bridge networks** are best when you need multiple containers to communicate on the same Docker host.
- **Host networks** are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.
- **Overlay networks** are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
- **Macvlan networks** are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.
- **Third-party network plugins** allow you to integrate Docker with specialized network stacks.

Docker Network

1. **default docker network**

```
anil@DESKTOP-0J00329:/$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
58bc84e25c7c   bridge    bridge    local
7c034408dd34   host      host      local
a54a8830a732   none      null      local
```

when container start , bridge network will attach by-default

2. network commamd

```
anil@DESKTOP-0J00329:/$ docker network --help

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
```

3. **if there will be no container running , network will be free. (check with inspect)**
   docker network inpect [network id]

```
anil@DESKTOP-0J00329:/$  docker network inspect 58bc84e25c7c
[
    {
        "Name": "bridge",
 "Containers": {},
```

4. **start at-least one container and inspect netwok bridge**
   docker container run –it nginx →ctrl+pq→docker container ls

```
anil@DESKTOP-0J00329:/$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED             STATUS              PORTS     NAMES
93625b2fb32d   nginx     "/docker-entrypoint.…"   About a minute ago  Up About a minute   80/tcp    lucid_rosalind
```

Docker network inspect [network id]

```
anil@DESKTOP-0J00329:/$ docker network inspect bridge
[
    {
        "Name": "bridge",
"Containers": {
    "93625b2fb32dc1f53e1144097d2483bc99566d86076a7c27546d61bd37ad602d": {
        "Name": "lucid_rosalind",
        "EndpointID": "86f8532a9643d0202ae5207c1baffabf43fedff8b7efed046c69a4af10e648ee",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
    }
},
```

5. **check ifconfig , virtual network list**

```
anil@DESKTOP-0J00329:/$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.25.122  netmask 255.255.255.240  broadcast 192.168.25.127
        inet6 fe80::215:5dff:fe5a:b3d7  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:5a:b3:d7  txqueuelen 1000  (Ethernet)
        RX packets 45386  bytes 25838643 (25.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8564  bytes 785507 (785.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

6. **docker network create –d bridge test (create "test"network of driver type bridge)**

-d bridge= driver type bridge

```
anil@DESKTOP-0J00329:/$ docker network create -d bridge test
e355b9efce2236d7465d6ce085d65a5eea6669da2289516ef1ff6c09d6453d57
```

```
anil@DESKTOP-0J00329:/$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
58bc84e25c7c   bridge    bridge    local
7c034408dd34   host      host      local
a54a8830a732   none      null      local
e355b9efce22   test      bridge    local
```

7. attach network with container

docker container run –it –-network test ubuntu:21.04 bash

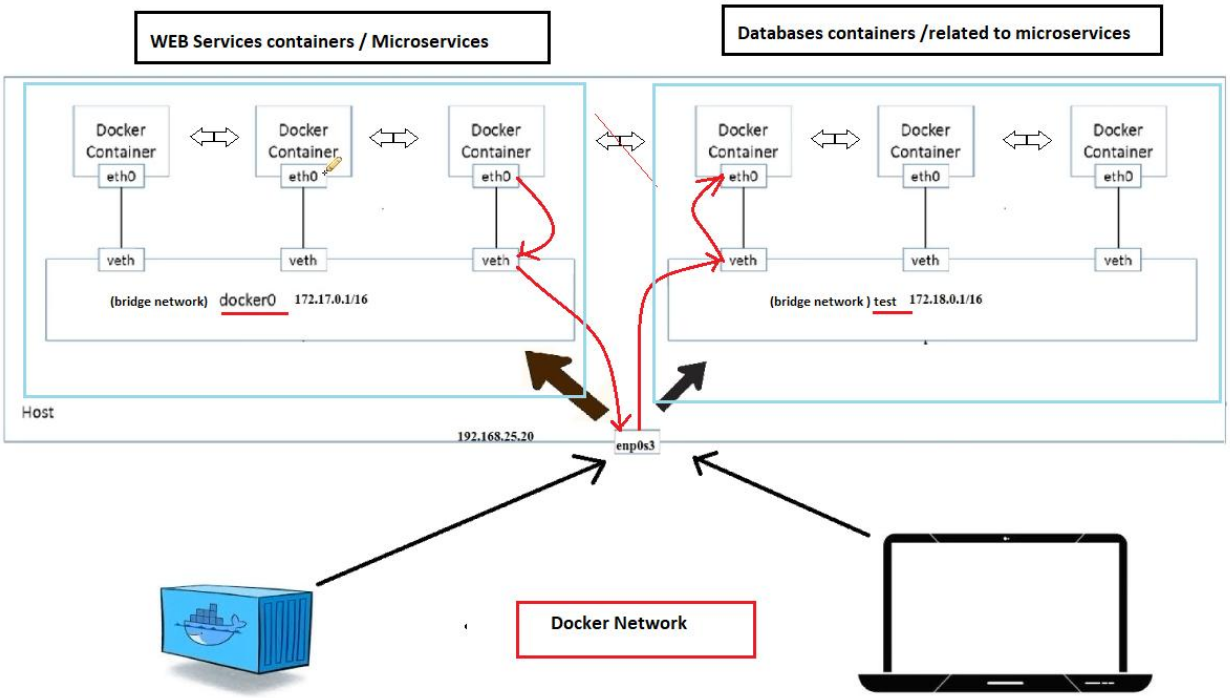
```
anil@DESKTOP-0J00329:/$ docker container run -it --network test ubuntu:21.04 bash
```

8. docker container inspect [container id]

```
anil@DESKTOP-0J00329:/$ docker container inspect 0876e391f83b
```


```
"Networks": {
    "test": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "0876e391f83b"
        ],
        "NetworkID": "e355b9efce2236d7465d6ce085d65a5eea6669da2289516ef1ff6c09d6453d57",
        "EndpointID": "10f6ffbac57cf810d6a79725738f24c6e18be1be0c24c742f56ac70b2ae4b9d5",
        "Gateway": "172.18.0.1",
        "IPAddress": "172.18.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:12:00:02",
        "DriverOpts": null
    }
}
```

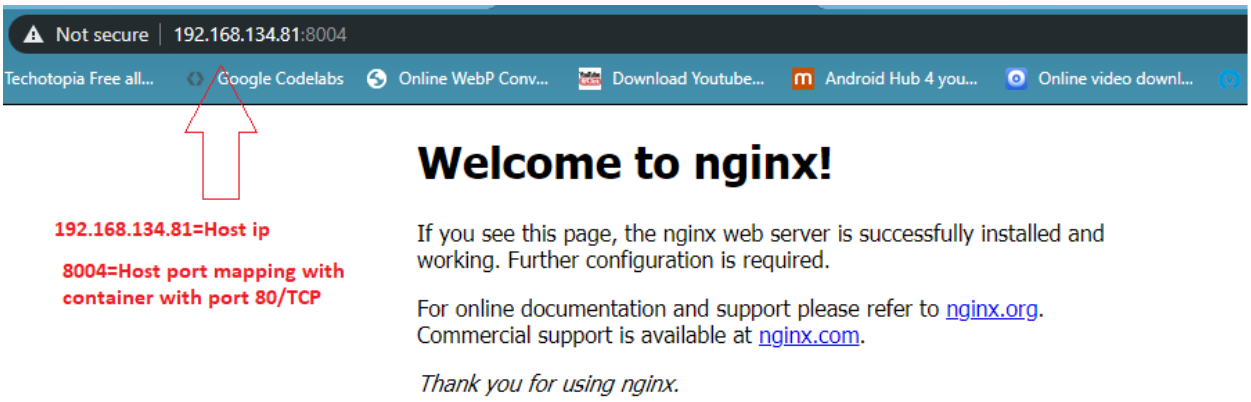
Docker network demo



1. create docker container with default network (bridge)

```
docker container run  -d -p 8001:80 nginx
docker container run  -d -p 8002:80 nginx

docker container run --network=test -d -p 8003:80 nginx
docker container run --network=test -d -p 8004:80 nginx
```

```
anil@DESKTOP-0J00329:/$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                    NAMES
42d833c6e072   nginx     "/docker-entrypoint.…"   5 seconds ago    Up 4 seconds    0.0.0.0:8004->80/tcp     romantic_carver
ef4bb98a2f47   nginx     "/docker-entrypoint.…"   5 minutes ago    Up 5 minutes    0.0.0.0:8003->80/tcp     compassionate_meitner
e525064aeec9   nginx     "/docker-entrypoint.…"   9 minutes ago    Up 9 minutes    0.0.0.0:8002->80/tcp     brave_fermi
ff4f1a6f5e57   nginx     "/docker-entrypoint.…"   12 minutes ago   Up 12 minutes   0.0.0.0:8001->80/tcp     sweet_lumiere
```



2. install necessary software for ip(net-tools),ping(iputils-tool),getfile(wget) each container

   - **docker container exec –it [container-id] /bin/bash**

- **apt-get update**
- **apt-get install -y wget net-tools iputils-ping inside**

3. inspect bridge info (for network type and associated containers)

```
anil@DESKTOP-0J0O329:/$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
55ccc5c1c702   bridge    bridge    local
7c034408dd34   host      host      local
a54a8830a732   none      null      local
e355b9efce22   test      bridge    local
```

b> docker network inspect **55ccc5c1c702**


```
"Name": "bridge",
"Id": "55ccc5c1c702ceaa0f0434f55e22efe792373c0696256ab72d0c20a972f3942e",
"Created": "2021-03-09T16:04:07.302228Z",
"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": {
    "e525064aeec929fc21214d66eb132e858cb642fbc0e26ef8c5c2c289e16024b8": {
        "Name": "brave_fermi",
        "EndpointID": "327ef2e19f4e149e2eb21026628650478bb23a5b11ddb3bbf9f2c23e77e454aa",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
    },
    "ff4f1a6f5e572b6366ecff16c69db45b65b0446c5dec239212e1cbcd63f819e0": {
        "Name": "sweet_lumiere",
        "EndpointID": "da5ac041db9c11a6ed039a105dd6e2d8a7ee3e303eea927445821b805bb2adbd",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "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"
},
```


c)

```
anil@DESKTOP-0J00329:/$ docker network inspect e355b9efce22
[
    {
        "Name": "test",
        "Id": "e355b9efce2236d7465d6ce085d65a5eea6669da2289516ef1ff6c09d6453d57",
        "Created": "2021-03-08T09:51:16.6123714Z",
        "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": {
            "42d833c6e0724ca27f44da19df2a335e26eacddd9a189040b5407c5218a06037": {
                "Name": "romantic_carver",
                "EndpointID": "bdabeef570d25acef0eb4e73e0022d2855cc97aed20b7cbc44abd9a7e851017f",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "ef4bb98a2f47a78afaf15727f93394cda7398553187e9eac8cbf384c1ad8a8db": {
                "Name": "compassionate_meitner",
                "EndpointID": "b852ce547ec2b0db88d051611b9ca548c2475c990febdb27ac0ee184f223e748",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

4. check container ipAddress and ping (same network)

test (bridge network)

172.18.0.0

```
root@42d833c6e072:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.18.0.3  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:ac:12:00:03  txqueuelen 0  (Ethernet)
        RX packets 7154  bytes 10287514 (9.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4331  bytes 238268 (232.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

container 1

```
root@ef4bb98a2f47:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.18.0.2  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:ac:12:00:02  txqueuelen 0  (Ethernet)
        RX packets 7193  bytes 10288265 (9.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4074  bytes 224010 (218.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
root@ef4bb98a2f47:/# ping 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=0.454 ms
64 bytes from 172.18.0.3: icmp_seq=2 ttl=64 time=0.173 ms
64 bytes from 172.18.0.3: icmp_seq=3 ttl=64 time=0.174 ms
64 bytes from 172.18.0.3: icmp_seq=4 ttl=64 time=0.177 ms
^C
--- 172.18.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 98ms
rtt min/avg/max/mdev = 0.173/0.244/0.454/0.121 ms
```

container 2

Connection successful

5. check wheather it will able to ping (cross network)

a) check other network port (docker 0)

bridge network (docker) 172.17.0.0

```
root@e525064aeec9:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
        RX packets 7254  bytes 10292638 (9.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4192  bytes 232732 (227.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 14  bytes 1288 (1.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14  bytes 1288 (1.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

B) try to ping docker0 (network container with test network container)

connection fails

```
root@e525064aeec9:/# ping 172.18.0.3
PING 172.18.0.3 (172.18.0.3) 56(84) bytes of data.
^C
--- 172.18.0.3 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 150ms

root@e525064aeec9:/# wget 172.18.0.3
--2021-03-10 08:20:55--  http://172.18.0.3/
Connecting to 172.18.0.3:80... ^C
```

6. connection of different network via host IpAddress
   wget 192.168.134.81:8004

```
root@e525064aeec9:/# wget 192.168.134.81:8004
--2021-03-10 10:25:01--  http://192.168.134.81:8004/
Connecting to 192.168.134.81:8004... connected.
HTTP request sent, awaiting response... 200 OK
Length: 612 [text/html]
Saving to: 'index.html.1'

index.html.1                        100%[===================================================================>]

2021-03-10 10:25:01 (28.3 MB/s) - 'index.html.1' saved [612/612]
```

Docker network (hostname mapping with network)

- if we will create container with default network , we have to use ipAddress to ping container in same network.
- If we want to enable hostname as IpAddress we need to map network aliases with container id to do so, create our own network (like test) , bydefault custome network aliases will be maped with hostname.

1. Create custom network
   Docker network create test
2. Run docker container in test network (two container)

   docker container run --network=test -it ubuntu:14.04 bash
   docker container run --network=test -it ubuntu:14.04 bash

3. Ping from one container to other container via hostname (you can able to ping successfully)

```
root@8c850d8ef23d:/# ping 19b04ddb6ea6
PING 19b04ddb6ea6 (172.18.0.2) 56(84) bytes of data.
64 bytes from 19b04ddb6ea6.test (172.18.0.2): icmp_seq=1 ttl=64 time=0.599 ms
64 bytes from 19b04ddb6ea6.test (172.18.0.2): icmp_seq=2 ttl=64 time=0.162 ms
64 bytes from 19b04ddb6ea6.test (172.18.0.2): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from 19b04ddb6ea6.test (172.18.0.2): icmp_seq=4 ttl=64 time=0.178 ms
64 bytes from 19b04ddb6ea6.test (172.18.0.2): icmp_seq=5 ttl=64 time=0.172 ms
64 bytes from 19b04ddb6ea6.test (172.18.0.2): icmp_seq=6 ttl=64 time=0.197 ms
```

4. to check difference inspect containers

Default network (bridge)


```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "55ccc5c1c702ceaa0f0434f55e22efe792373c0696256ab72d0c20a972f3942e",
        "EndpointID": "76f100af9475676c4d846bdd82775d7766270efa448637127c153c44e27ea6ed",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.3",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:03",
        "DriverOpts": null
    }
}
```


custome network (test) , it will map container id with network Aliases


```
"Networks": {
    "test": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "19b04ddb6ea6"
        ],
        "NetworkID": "e355b9efce2236d7465d6ce085d65a5eea6669da2289516ef1ff6c09d6453d57",
        "EndpointID": "6d3e04e6a5a27629517fea57fec4baed45460e43f515bab345090819fb101b2c",
        "Gateway": "172.18.0.1",
        "IPAddress": "172.18.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:12:00:02",
        "DriverOpts": null
    }
}
```


Docker network Host

***. If we will run a container with host network then container will treat like a host system. you can access host container via their ipAddress directly (because both container and host will work on same ipaddress) bydefault iPAdress will point 80/tcp port.**

1. **Docker container run --network=host -itd nginx**
2. **Docker container exec –it [container id] /bin/bash**
3. **Apt-get update and apt-get install –y net-tools**
4. **Ifconfig**

continer host machine

```
root@docker-desktop:/# ifconfig
br-e355b9efce22: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        inet6 fe80::42:d5ff:fe37:330a  prefixlen 64  scopeid 0x20<link>
        ether 02:42:d5:37:33:0a  txqueuelen 0  (Ethernet)
        RX packets 8512  bytes 353446 (345.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14423  bytes 20582076 (19.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

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::42:44ff:fed3:b988  prefixlen 64  scopeid 0x20<link>
        ether 02:42:44:d3:b9:88  txqueuelen 0  (Ethernet)
        RX packets 4236  bytes 177580 (173.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7283  bytes 10295986 (9.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.65.3  netmask 255.255.255.0  broadcast 192.168.65.15
        inet6 fe80::50:ff:fe00:1  prefixlen 64  scopeid 0x20<link>
        ether 02:50:00:00:00:01  txqueuelen 1000  (Ethernet)
        RX packets 84988  bytes 122711414 (117.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 38221  bytes 2103031 (2.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 779  bytes 78587 (76.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 779  bytes 78587 (76.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vpnkit0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.65.4  netmask 255.255.255.255  broadcast 192.168.65.255
        inet6 fe80::90f5:f3ff:fee3:91d4  prefixlen 64  scopeid 0x20<link>
        ether 92:f5:f3:e3:91:d4  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 23  bytes 1786 (1.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ubuntu host machine

```
anil@DESKTOP-0J0O329:/$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.25.114  netmask 255.255.255.240  broadcast 192.168.25.127
        inet6 fe80::215:5dff:fe5a:bf08  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:5a:bf:08  txqueuelen 1000  (Ethernet)
        RX packets 26914  bytes 2772287 (2.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 48  bytes 4358 (4.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 638  bytes 74895 (74.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 638  bytes 74895 (74.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

window host machine

```
Ethernet adapter vEthernet (Default Switch):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::c4f2:c8c9:b9c2:970f%16
   IPv4 Address. . . . . . . . . . . : 192.168.134.81
   Subnet Mask . . . . . . . . . . . : 255.255.255.240
   Default Gateway . . . . . . . . . :

Ethernet adapter vEthernet (WSL):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::b1b4:62e7:4076:7eb0%50
   IPv4 Address. . . . . . . . . . . : 192.168.25.113
   Subnet Mask . . . . . . . . . . . : 255.255.255.240
   Default Gateway . . . . . . . . . :
```

***Above all machine have different port , no any host port matching (it is not working on my sys)**
***show I can not able to directly access container host via window host port (http://192.168.134.81/)**

5. **Docker container inspect [container id]**


```
"Networks": {
    "host": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "7c034408dd349898b403a169c73e51c0c75bfc80830dd1fd4fdea164535f527c",
        "EndpointID": "1467a96039aab817a523d3228514ce9dd3c989590a34d10781623e896b1637cd",
        "Gateway": "",
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "",
        "DriverOpts": null
    }
}
```


Host container will not contain any IPaddress/gatway

Docker network none (null)

*. By-default container has bridge network associated with container , if we don't want to attach any network with container , we use **network type none**

1. Docker container run –network=none -it ubuntu:14.04 bash
2. Ifconfig

```
anil@DESKTOP-0J0O329:/$ docker container run --network=none -it ubuntu:14.04 bash
root@318330177a1b:/# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

3. Docker container inspect [container -id]


```
"NetworkSettings": {
    "Bridge": "",
    "SandboxID": "0af1feada3ad307c1f1b36904ed758ed8b1e893753b98267bb4e8cd856f6fe3f",
    "HairpinMode": false,
    "LinkLocalIPv6Address": "",
    "LinkLocalIPv6PrefixLen": 0,
    "Ports": {},
    "SandboxKey": "/var/run/docker/netns/0af1feada3ad",
    "SecondaryIPAddresses": null,
    "SecondaryIPv6Addresses": null,
    "EndpointID": "",
    "Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "",
    "IPPrefixLen": 0,
    "IPv6Gateway": "",
    "MacAddress": "",
    "Networks": {
        "none": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": null,
            "NetworkID": "a54a8830a732b1de9ddeccb799fcb6e8ee9a0c6e153c14b8af38843a06067799",
            "EndpointID": "b0766ea220ab7bf033ec5dfcc8f60f548699060552d11cf516c9fea8644670db",
            "Gateway": "",
            "IPAddress": "",
            "IPPrefixLen": 0,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "",
            "DriverOpts": null
        }
    }
```

Docker network (multiple network mapping with a container)
[network connect , network disconnect]

- Bydefault one network will ve associated with container (bridge), we can add more network of only brige type to an container. We can't add multiple network on host and none network

```
anil@DESKTOP-0J00329:/$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
55ccc5c1c702   bridge    bridge    local
7c034408dd34   host      host      local
a54a8830a732   none      null      local
e355b9efce22   test      bridge    local
```

1. **Docker container run –-network=bridge –it ubuntu:14.04 bash**

```
anil@DESKTOP-0J00329:/$ docker container run --network=bridge -it ubuntu:14.04 bash
root@f8cdc693c678:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10 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:836 (836.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

2. **docker network connect test f8cdc693c678** [Add network test on existing cotainer]

```
root@f8cdc693c678:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 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:976 (976.0 B)  TX bytes:0 (0.0 B)

eth1      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:9 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:766 (766.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

3. **docker network disconnect test f8cdc693c678** [remove network]
4. **try to add other network**

```
anil@DESKTOP-0J00329:/$ docker network connect none f8cdc693c678
Error response from daemon: container cannot be connected to multiple networks with one of the networks in private (none) mode
anil@DESKTOP-0J00329:/$ docker network connect host f8cdc693c678
Error response from daemon: container cannot be disconnected from host network or connected to host network
```

*5.Registry Server (unsecure)

The Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images

Why use it

You should use the Registry if you want to:

- tightly control where your images are being stored
- fully own your images distribution pipeline
- integrate image storage and distribution tightly into your in-house development workflow

** The Registry is compatible with Docker engine **version 1.6.0 or higher**.

(unsecure registry server / pull and push only allowed via 127.0.0.1/8 or HTTPS url)

- **In general we are using docker.io to pull and push the image.**
- **We need to create custom Registry to push images and pull images.**

1. **Docker pull registry**

```
anil@DESKTOP-0J0O329:/$ docker pull docker.io/registry
Using default tag: latest
latest: Pulling from library/registry
e95f33c60a64: Pull complete
4d7f2300f040: Pull complete
35a7b7da3905: Pull complete
d656466e1fe8: Pull complete
b6cb731e4f93: Download complete
```

default path (optional)

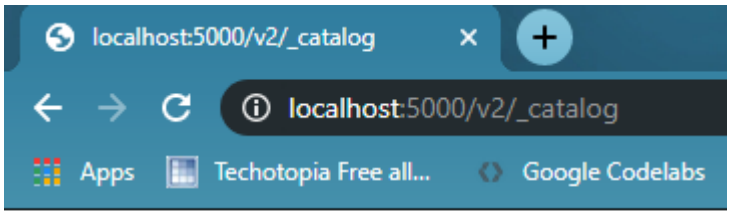
docker pull registry

```
anil@DESKTOP-0J0O329:/$ docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
ubuntu_imported_image   latest    8014e4d46765   8 days ago      201MB
mysql                   latest    8457e9155715   12 days ago     546MB
registry                latest    5c4008a25e05   2 weeks ago     26.2MB
```

2. Create simple_registry container from registry image with default volume
   docker container run -it -d -p 5000:5000 --name simple_registry registry

```
anil@DESKTOP-0J0O329:/$ docker container run -it -d -p 5000:5000 --name simple_registry registry
c6df3f70428cef885b042a011b0bfabede4ef0cd0452f9db9141fa1c166ebe03
```

3. Check images in repository of private registry (simple_resgisty)
   127.0.0.1:5000/v2/_catalog




```
{
    repositories: [ ]
}
```

4. [customize mount path as per storage location/directory]
   Docker container insptect [container id]

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "fb360848004cfa4b10dd3d16901be2e2c650b6b178072b29a22d8b9
        "Source": "/var/lib/docker/volumes/fb360848004cfa4b10dd3d16901be
        "Destination": "/var/lib/registry",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

directory path

location path: we can use amazon s3

5. Docker image tag ubuntu:21.04 [url path]:[port]/[image name]

default location : docker image tag hub.docker.com/ubuntu:21.04

```
anil@DESKTOP-0J0O329:/$ docker image tag ubuntu:21.04 127.0.0.1:5000/ubuntu:21.04
```

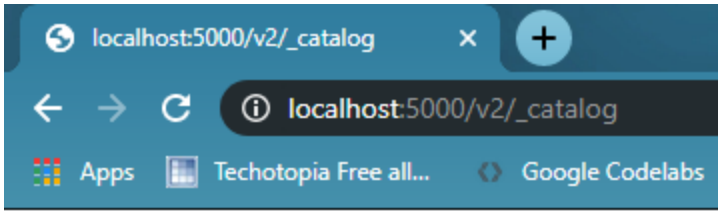
custome location with [image name]

```
anil@DESKTOP-0J0O329:/$ docker image rm -f ubuntu:21.04
127.0.0.1:5000/ubuntu   21.04     1fc773f9e714   7 weeks ago     80.7MB
ubuntu                  21.04     1fc773f9e714   7 weeks ago     80.7MB
```

6. Docker image push 127.0.0.1:5000/ubuntu:21.04 [push in local registry server]

```
anil@DESKTOP-0J0O329:/$ docker image push 127.0.0.1:5000/ubuntu:21.04
The push refers to repository [127.0.0.1:5000/ubuntu]
aa8011ef7554: Pushed
befc9ff11fa7: Pushed
```

7. check




```
{
  - repositories: [
        "ubuntu"
    ]
}
```

8. Pull docker image from local registry server

```
anil@DESKTOP-0J0O329:/$ docker image pull 127.0.0.1:5000/ubuntu:21.04
21.04: Pulling from ubuntu
486d08009c1b: Already exists
31e228808914: Already exists
7316b1e8087c: Already exists
Digest: sha256:c0f51f9c801886eb5b1e93af916ea0a2d145ace8c40474db957fe4e062a7120d
Status: Downloaded newer image for 127.0.0.1:5000/ubuntu:21.04
127.0.0.1:5000/ubuntu:21.04
anil@DESKTOP-0J0O329:/$ docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
ubuntu_imported_image   latest    8014e4d46765   8 days ago      201MB
mysql                   latest    8457e9155715   12 days ago     546MB
registry                latest    5c4008a25e05   2 weeks ago     26.2MB
nginx                   latest    35c43ace9216   3 weeks ago     133MB
myubuntu                1         5823492a6659   7 weeks ago     132MB
127.0.0.1:5000/ubuntu   21.04     1fc773f9e714   7 weeks ago     80.7MB
```

Docker Registry Server (unsecure server)

(allow other url to push and pull images , other than 127.0.0.1 and https)

1. **Check we can able to pull or push via other url path**

```
anil@DESKTOP-0J0O329:/$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.25.114  netmask 255.255.255.240  broadcast 192.168.25.127
```

```
anil@DESKTOP-0J0O329:/$ docker image tag ubuntu:14.04 192.168.25.114:5000/ubuntu:14.04
```

```
anil@DESKTOP-0J0O329:/$ docker image ls
```

```
192.168.25.114:5000/ubuntu   14.04     df043b4f0cf1   5 months ago    197MB
```

docker image push 192.168.25.144:5000/ubuntu:14.04

```
anil@DESKTOP-0J0O329:/$ docker image push 192.168.25.114:5000/ubuntu
Using default tag: latest
The push refers to repository [192.168.25.114:5000/ubuntu]
Get https://192.168.25.114:5000/v2/: http: server gave HTTP response to HTTPS client
```

2. Resolved the issue by adding a file `/etc/docker/daemon.json` with following content

```
Windows PowerShell    root@DESKTOP-0J0O329: /

{
        "insecure-registries" : [ "192.168.25.114:5000" ]
}
~
~
```

3. Restart docker
4. Docker image push 192.168.25.114:5000/ubuntu:14.04

```
root@DESKTOP-0J0O329:/# docker image push 192.168.25.114:5000/ubuntu:14.04
The push refers to repository [192.168.25.114:5000/ubuntu]
Get https://192.168.25.114:5000/v2/: dial tcp 192.168.25.114:5000: connect: connection refused
```

(it should work but ,here not working --- we will check latter for this issue)

B. Docker File

Docker can build images automatically by reading the instructions from a `Dockerfile`. A `Dockerfile` is a text document that contains all the commands a user could call on the command line to assemble an image. Using `docker build` users can create an automated build that executes several command-line instructions in succession.

1.Create Docker File (layed architecture)

1. Vi Dockerfile → create/open Dockerfile (name Dockerfile is default , docker will search Dockerfile for build from path)

   and press I (-- insert mode --)

```
root@DESKTOP-0J00329:/# vi Dockerfile
```

2. FROM ubuntu:16.04

```
FROM ubuntu:16.04

~
~
~
~
```

3. Press esc -> :w (enter to save) -> :q (enter to save and exit cell)
4. Docker image build –t [give any name of image] [path]

   docker image build -t myubuntu:1 . →(**.** Dockerfile is present in root directry, add tag for versioning)

```
root@DESKTOP-0J00329:/# docker image build -t myubuntu:1 .
[+] Building 52.6s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 56B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/ubuntu:16.04
```

5. Docker image ls

```
root@DESKTOP-0J00329:/# docker images
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
ubuntu_imported_image   latest    8014e4d46765   22 hours ago    201MB
nginx                   latest    35c43ace9216   2 weeks ago     133MB
myubuntu                1         5823492a6659   6 weeks ago     132MB
ubuntu                  21.04     1fc773f9e714   6 weeks ago     80.7MB
ubuntu                  latest    f63181f19b2f   6 weeks ago     72.9MB
hello-world             latest    bf756fb1ae65   14 months ago   13.3kB
```

6. Create and run docker container using created image

   Docker container –it myubuntu:1

```
root@DESKTOP-0J00329:/# docker container run -it myubuntu:1
root@4500d50ef78c:/#
```

2. Create Dockerfile (of OS ubuntu 16.04 and inatall software)

1. **Vi Dockerfile and press I (-- insert mode --)**

```
root@DESKTOP-0J0O329:/# vi Dockerfile
```

```
FROM ubuntu:16.04
RUN  apt-get update && apt-get install -y tree
RUN  touch /tmp/1.txt
RUN  touch /tmp/2.txt
RUN  touch /tmp/2.txt
~
~
~
```

os and version

update and install in single line
(it will create one layer)

create file in temp ,
each line will create new layer
while building image

2. **Press esc -> :q(enter to save and exit)**
3. **Docker image build –t myubuntu:2 .**

```
root@DESKTOP-0J0O329:/# docker image build -t myubuntu:2 .
[+] Building 58.3s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 170B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/ubuntu:16.04
 => CACHED [1/5] FROM docker.io/library/ubuntu:16.04@sha256:e74994b7a9ec8e2129cfc6a871f3236940006e
 => [2/5] RUN  apt-get update && apt-get install -y tree
 => [3/5] RUN  touch /tmp/1.txt
 => [4/5] RUN  touch /tmp/2.txt
 => [5/5] RUN  touch /tmp/2.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:c59320c3af9f8c12f0b1236f99f776f5ae67d3f1b2e1d16c73145c5ba2a41f22
 => => naming to docker.io/library/myubuntu:2
```

layer wise process

4. **Docker image ls**

```
root@DESKTOP-0J0O329:/# docker image ls
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
myubuntu                2         c59320c3af9f   5 minutes ago   163MB
ubuntu_imported_image   latest    8014e4d46765   23 hours ago    201MB
nginx                   latest    35c43ace9216   2 weeks ago     133MB
myubuntu                1         5823492a6659   6 weeks ago     132MB
ubuntu                  21.04     1fc773f9e714   6 weeks ago     80.7MB
ubuntu                  latest    f63181f19b2f   6 weeks ago     72.9MB
hello-world             latest    bf756fb1ae65   14 months ago   13.3kB
```

5. **Docker container run -it myubuntu:2 /bin/bash** (check tree and created file available or not)

```
root@f3e19cf75b07:/tmp# tree
.
|-- 1.txt
`-- 2.txt
```

3. Dockerfile (label, ,env ,run,workdir)

1. Vi Dockerfile and press I (-- insert mode --)

```
FROM ubuntu:16.04
LABEL name="Anil Gupta"
LABEL profile="Devops Engineer"
ENV   NAME Anil
ENV   Version version
RUN   pwd>/tmp/1stpwd.txt
RUN   cd /tmp/
RUN   pwd>/tmp/2ndpwd.txt
WORKDIR /tmp
~
~
~
```

docker image inspect [image name]

inside running container -->env

create file in /tmp

inside tmp

due to layered architechture
file will be created from root path :means inside tmp directory.

change working dir form root to /tmp

2. Press esc-> :q(save and exit cell)
3. Docker image build –t myubuntu:3 .

```
root@DESKTOP-0J0O329:/# docker image build -t myubuntu:3 .
[+] Building 6.6s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 231B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/ubuntu:16.04
 => CACHED [1/5] FROM docker.io/library/ubuntu:16.04@sha256:e74994b7a9ec8e2129cfc6a871f323694000
 => [2/5] RUN   pwd>/tmp/1stpwd.txt
 => [3/5] RUN   cd /tmp/
 => [4/5] RUN   pwd>/tmp/2ndpwd.txt
 => [5/5] WORKDIR /tmp
 => exporting to image
 => => exporting layers
 => => writing image sha256:aab7c9313d0fcd8007df703eafd6d59de132e827acddd592a41a615509136d8c
 => => naming to docker.io/library/myubuntu:3
```

4. Docker image ls

```
root@DESKTOP-0J0O329:/# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED              SIZE
myubuntu            3         aab7c9313d0f   About a minute ago   132MB
myubuntu            2         c59320c3af9f   About an hour ago    163MB
```

5. Docker image inspect myubuntu:3 (check level and env)

```
    "Labels": {
        "name": "Anil Gupta",
        "profile": "Devops Engineer"
    }
```

```
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "NAME=Anil",
    "Version=version"
],
```

6. Docker container run –it myubuntu:3 /bin/bash (check env and temp for files)

```
root@DESKTOP-0J0O329:/# docker container run -it myubuntu:3 /bin/bash
root@8666faf8f35a:/tmp# env
HOSTNAME=8666faf8f35a
TERM=xterm
NAME=Anil
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;0
4;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha
z=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.
```

4. Dockerfile (create user inside image, copy (copy only files) vs add (copy and extract file if tar file) , Add)

1. Vi Dockerfile and press I for (--insert mode--)

```
FROM ubuntu:16.04
LABEL name="Anil Gupta"
LABEL profile="Devops Engineer"
ENV   NAME anil
ENV   PASS paper
RUN   pwd>/tmp/1stpwd.txt
RUN   cd /tmp/
RUN   pwd>/tmp/2ndpwd.txt
WORKDIR /tmp
RUN pwd>/tmp/3rdpwd.txt
RUN apt-get update && apt-get install -y openssh-server && apt-get install -y python
RUN useradd -d /home/anil/ -g root -G  sudo -m -p $(echo "$PASS" | openssl passwd -1 -stdin) $NAME
RUN whoami>/tmp/1stwhoami.txt
USER $NAME
RUN whoami>/tmp/2ndwhoami.txt
RUN mkdir -p /tmp/project
COPY test /tmp/project/
~
~
```

env variable

install softwares

add user in image

chnage user

copy files to images

2. Press esc -> :wq (save and exit from cell)
3. Docker image build –t myubuntu:6 .

```
root@DESKTOP-0J00329:/# docker image build -t myubuntu:6 .
[+] Building 3.7s (17/17) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 555B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/ubuntu:16.04
 => [internal] load build context
 => => transferring context: 63B
 => [ 1/12] FROM docker.io/library/ubuntu:16.04@sha256:e74994b7a9ec8e2129cfc6a871f3236940006ed31
 => CACHED [ 2/12] RUN   pwd>/tmp/1stpwd.txt
 => CACHED [ 3/12] RUN   cd /tmp/
 => CACHED [ 4/12] RUN   pwd>/tmp/2ndpwd.txt
 => CACHED [ 5/12] WORKDIR /tmp
 => CACHED [ 6/12] RUN pwd>/tmp/3rdpwd.txt
 => CACHED [ 7/12] RUN apt-get update && apt-get install -y openssh-server && apt-get install -y
 => CACHED [ 8/12] RUN useradd -d /home/anil/ -g root -G  sudo -m -p $(echo "paper" | openssl pa
 => CACHED [ 9/12] RUN whoami>/tmp/1stwhoami.txt
 => CACHED [10/12] RUN whoami>/tmp/2ndwhoami.txt
 => [11/12] RUN mkdir -p /tmp/project
 => [12/12] COPY test /tmp/project/
 => exporting to image
 => => exporting layers
 => => writing image sha256:b36bcc2c6c12ebf19648699196ca368f45011921894cc9b9ad8ee6d43b973341
 => => naming to docker.io/library/myubuntu:6
```

4. Docker image ls

```
root@DESKTOP-0J00329:/# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED             SIZE
myubuntu            6         b36bcc2c6c12   24 seconds ago      240MB
```

5. Docker container run –it myubuntu:6 /bin/bash

```
anil@aeaab8479c53:/tmp$ ls
1stpwd.txt  1stwhoami.txt  2ndpwd.txt  2ndwhoami.txt  3rdpwd.txt  project
anil@aeaab8479c53:/tmp$ cat 1stwhoami.txt
root
anil@aeaab8479c53:/tmp$ cat 2ndwhoami.txt
anil
```

```
anil@aeaab8479c53:/tmp$ cd project
anil@aeaab8479c53:/tmp/project$ ls
1stTest.txt
anil@aeaab8479c53:/tmp/project$
```

5. Dockerfile (CMD, get particular cell after running container)

1. vi Dockerfile and press I for (--insert mode--)

```
FROM ubuntu:16.04
LABEL name="Anil Gupta"
LABEL profile="Devops Engineer"
ENV   NAME anil
ENV   PASS paper
RUN   pwd>/tmp/1stpwd.txt
RUN   cd /tmp/
RUN   pwd>/tmp/2ndpwd.txt
WORKDIR /tmp
RUN pwd>/tmp/3rdpwd.txt
RUN apt-get update && apt-get install -y openssh-server && apt-get install -y python
RUN useradd -d /home/anil/ -g root -G  sudo -m -p $(echo "$PASS" | openssl passwd -1 -stdin) $NAME
RUN whoami>/tmp/1stwhoami.txt
USER $NAME
RUN whoami>/tmp/2ndwhoami.txt
RUN mkdir -p /tmp/project
COPY test /tmp/project/
CMD  ["python"]
~
```

get python cell after container start , in a docker file one last CMD command will show if more than two will be there

2. Press esc -> :wq(save and exit)
3. Docker image build –t myubuntu:8 .
4. Docker container run –it myubuntu:8

```
root@DESKTOP-0J00329:/# docker container run -it myubuntu:8
Python 2.7.12 (default, Mar  1 2021, 11:38:31)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

6. Dockerfile (entry point -> set default entry point when container starts)

1. Vi Dockerfile and press I (-- insert mode --)

```
FROM ubuntu:16.04
LABEL name="Anil Gupta"
LABEL profile="Devops Engineer"
ENV   NAME anil
ENV   PASS paper
RUN   apt-get update
RUN   apt-get install -y tree python
ENTRYPOINT ["tree"]
CMD ["--help"]
```

excecute any sofware ,server,file as an enty point

excecute command based on entypoint

2. Press esc enter -> :wq (to exit cell)
3. Docker image build –t myubuntu:9 .

```
root@DESKTOP-0J00329:/# docker image build -t myubuntu:9 .
[+] Building 98.4s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 240B
 => [internal] load .dockerignore
 => => transferring context: 2B
```

4. Docker image ls
5. Docker container run myubuntu:9

```
root@DESKTOP-0J00329:/# docker container run myubuntu:9
usage: tree [-acdfghilnpqrstuvxACDFJQNSUX] [-H baseHREF] [-T title ]
        [-L level [-R]] [-P pattern] [-I pattern] [-o filename] [--version]
        [--help] [--inodes] [--device] [--noreport] [--nolinks] [--dirsfirst]
```

6. Docker container ls –a

```
root@DESKTOP-0J00329:/# docker container ls -a
CONTAINER ID   IMAGE              COMMAND                  CREATED
e3c10947fd5d   mysql              "docker-entrypoint.s…"   2 minutes ago
1050026d40f8   caf5e792464e       "tree --help"            53 minutes ago
d2f08358fff8   caf5e792464e       "tree --help"            54 minutes ago
1415fa24af86   e6bdf79ce6e3       "python"                 42 hours ago
a594a4c85377   e6bdf79ce6e3       "python"                 42 hours ago
984d2197394f   e6bdf79ce6e3       "/bin/bash"              42 hours ago
```

C. Docker Compose

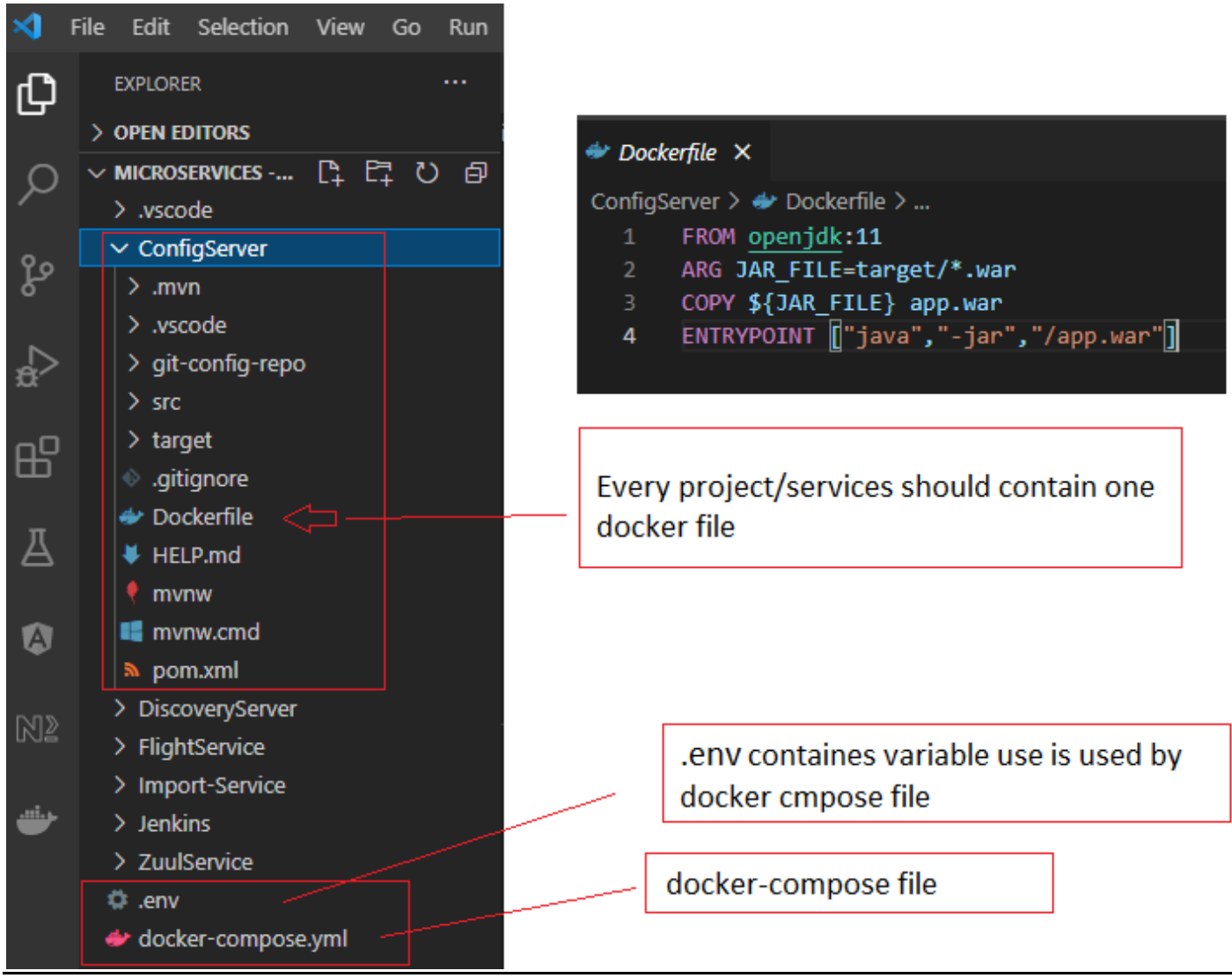
Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see the list of features.

Compose works in all environments: production, staging, development, testing, as well as CI workflows. You can learn more about each case in Common Use Cases.

Using Compose is basically a three-step process:

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Run `docker compose up` and the Docker compose command starts and runs your entire app. You can alternatively run `docker-compose up` using the docker-compose binary.

{ Microservice Project Architecture/ and package overview with docker }



Docker-compose.yml

docker-compose.yml

```yaml
version: '3'

services:

  springboot-configserver:
    image : springboot-configserver:1.0
    volumes:
      - springboot-volume:/usr/src/
    networks:
     springboot-network:
    ports :
      - 8888:8888
    build :
      context : ./configserver/
      dockerfile: Dockerfile
    # depends_on:
    # - mysql-standalone
    container_name: springboot-configserver
```

docker compose version

define list of service info

sevice identity , which contains several info like ,image ,volumes etc

image name

volume name

network name (declaration/variable)

port mapping with host server to container

docker file of configserver

container name

```yaml
networks:

  springboot-network:
    name: user-network
    driver: bridge
```

we can define all service network at bottom of docker compose file

network variable of particular service

network name and its driver type

Docker-compose with .env file

docker-compose.yml | .env

.env

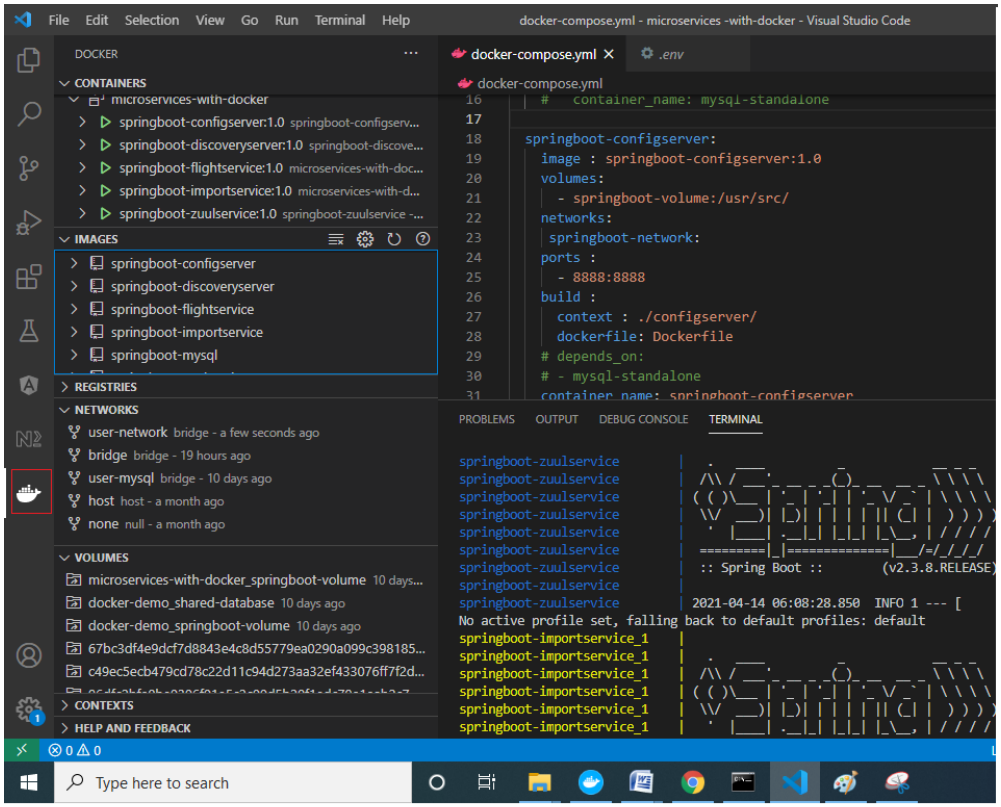
```
DB_NAME=test
DB_USER=root
DB_PWD=root
DB_ROOT_PWD=admin
```

.env file

```yaml
mysql-standalone:
  image: mysql:5.7
  volumes:
    - shared-database:/var/lib/mysql
  networks:
   mysql-newotk:

  environment:
  - MYSQL_DATABASE=${DB_NAME}
  - MYSQL_ROOT_PASSWORD=${DB_ROOT_PWD}
  - MYSQL_USER= ${DB_USER}
  - MYSQL_PASSWORD=${DB_PWD}
  container_name: mysql-standalone
```

use .env variable in docker-compose file

Install docker plugin in vs code and run docker-compose command to start containers



1. install docker plugin in .vs code to see all activities realted to docker as-

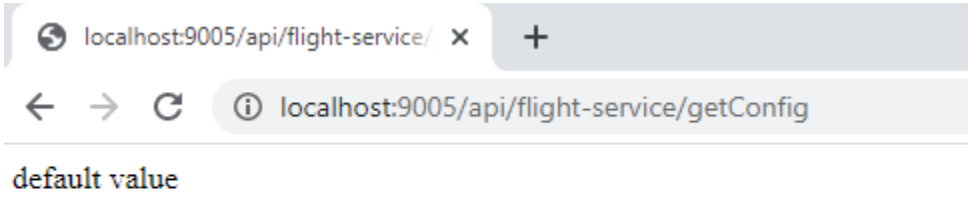
-> containers
->images
->registries
->network
->volumes

2. start docker
3. docker compose commnad

=> docker-compose up

Check server is running or not via browser (host machine based on port mapping)

➔ Below 9005 is related to zull api gateway which is running successfully



Without docker compose (steps to create container of individual service)

1. mvn package -DskipTests // mvn -f ./import-service/pom.xml package -DskipTests
2. docker build -t springboot-mysql:1.0 . // read docker file and crate image
3. docker container run -it -p 9002:9002 --network=user-mysql springboot-mysql:1.0
4. docker logs [container id]

Disable websecurity (or cors , add below code to crome properties)

"C:\Program Files\Google\Chrome\Application\chrome.exe" --disable-web-security --user-data-dir="C:/ChromeDevSession"

***Access via browser (check api is working of not)**
http://localhost:9002/api/flight-service/getImportService

Docker-compose commands :

docker-compose --help (Command

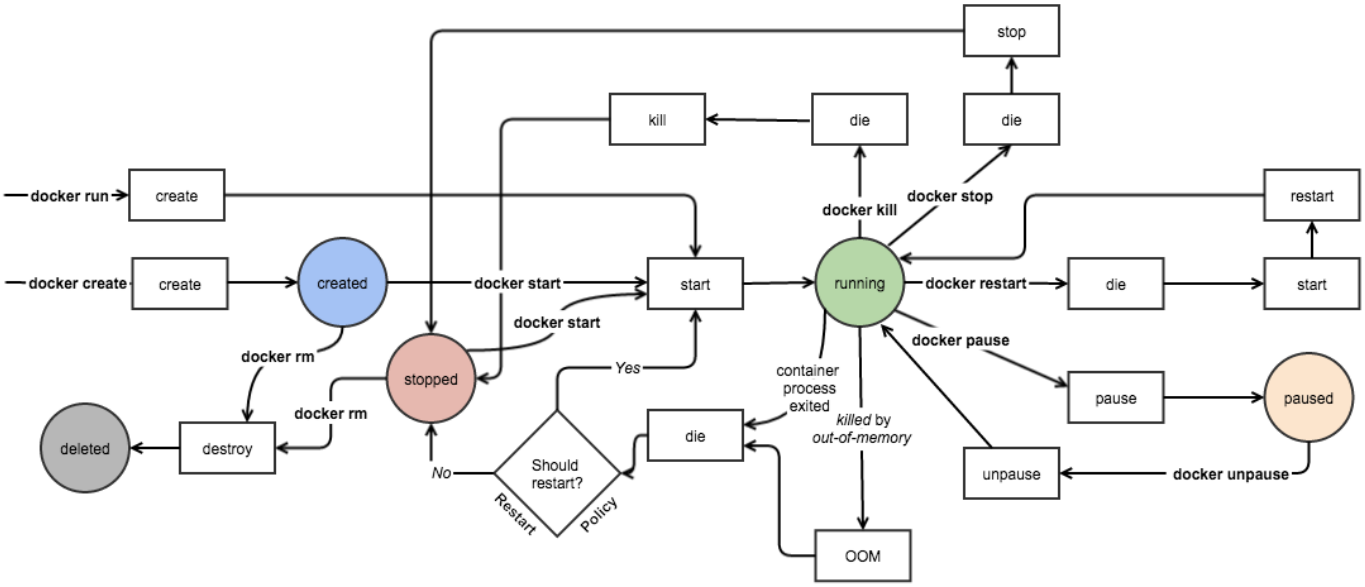
```
Commands:
  build              Build or rebuild services
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove resources
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show version information and quit
```

Example -- 1. docker-compose build (build images)
2. docker-compose create (create container only)
3. docker -compose up (create and start container)
4.docker-compose up --scale springboot-flightservice=2 springboot-importservice=2

(scale services -> for scalling we should remove port mapping from docker-compose file) ..etc

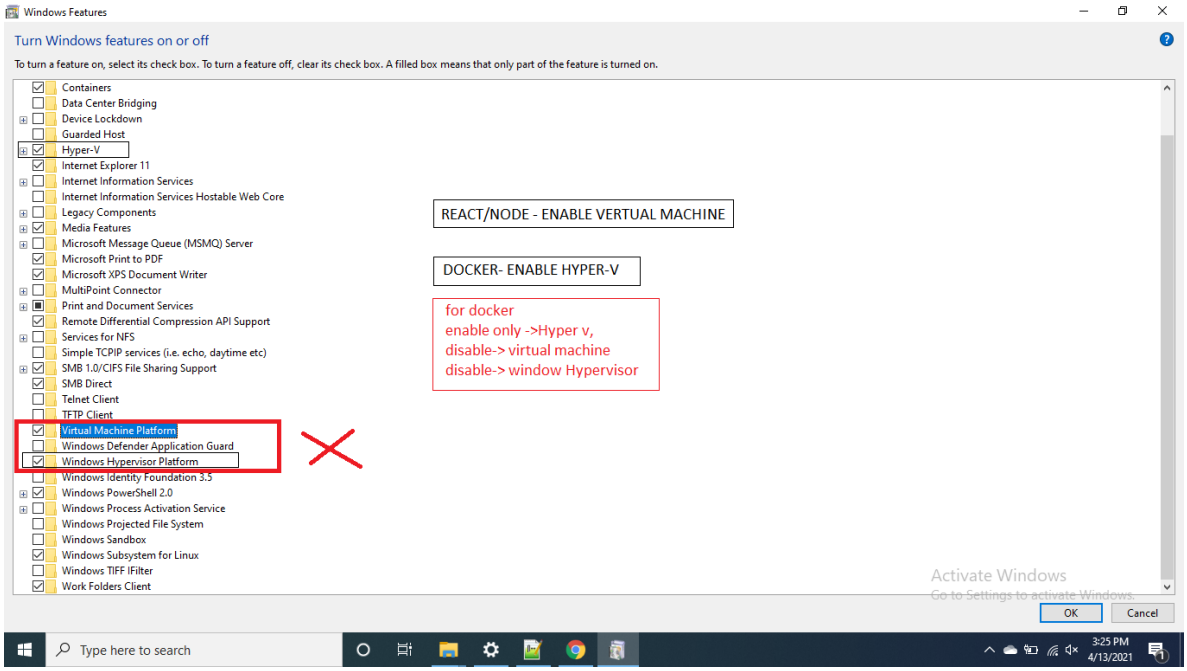
---

Docker life cycle :-



---

windown features for docker (configuration , Enable hyper -v)



Reference : -

1. https://docs.docker.com/
2. https://www.youtube.com/watch?v=ETBj0oxe81o&list=PL6XT0grm_Tfje2ySztzdhp0HmCjVj5P4z [gourav Sharma docker tutorial]

Thanks