<body bgcolor=red text=blue>

<marquee><H1> Welcome to Cloud </H1></ma

</body> pagename: index.html,

</html>

built → docker build -t webserver.

* Managning Docker Hub :-

→ It is a service provided by Docker for finding sharing container images. With your team.

→ Link your images to the GitHub/ Bit bucket repositories that can be built automatically based on web hook

→ public repositories:

User get access to free public repositories for storing and sharing images.

→ Any one can pull the docker pull command to download an image & run or build further images from it.

private Repositories:

→ They are just that are private.

→ users can choose a "subscription plan" for it

Docker HUB vs Docker Subscription (CLI):

DH:
- Shareable image, but not be private.
- No hassle of self-hosting.
- Free (except for a certain number of private

DS:

→ Integrated into your authentication service.
  ie AD/LDAP.

→ Deployed on our own infrastructure (or) cloud

→ Commericial

Note: By default Reporisitories pushed as public

AD → Active directory.

Docker HUB Enterprise (GUI)

→ It offers you is access to the S/W, access to upd
/patches/ security fixes, and supporting relating to
issues with the software.

→ the open-source Dockery repository image does
offer these services at this level

Step1: Hub. docker.com → Signin

Step2: create a Repository. → click on Repositor

rnraju Nameof images: Webserver → [account name + Rep name] tag

visibility :- ☑ public ☐ private

∴ rnraju/Webserver

→ Go to docker machine

project 1 :- Building Webserver.

Step1 :- Vi Dockerfile

Installing Apache Webserver.

FROM ubuntu : latest

RUN apt-get update -y

COPY index.html /var/WW/html → copy index.html file into Document root location

EXPOSE 80. → service start

CMD ["apachectl", "-D", " FOREGROUND"]

→ Save it

Step2: vi index.html [same code] → save it

Step3: docker built -t rnraju/webserver :

webserver: 2.4 .

tagname

image docker repository address (Id+name)

Step4: docker images

Step5: make a container [to check running (00) no

docker run -it -d -p 8000:80 --name

testserver → imagename

f759bf0f285c → imageId ..

step6 docker ps.

Instance publicIpv4 copy → security Group All TCP (security groups)

=> 54.164.35.206: 8000 "

Step7: Image push to docker Hub.

Step1: Login in to docker hub.

docker login. → usrname: ravinadh
passwd: kingravi@1

Step2:- docker push mraju/webser:24
→ Imagename

Step3: Goto docker hub → Image is displayed

: push. cmd

* 2nd project:-

Building docker image for NodeJS :-

Step1: Vi Dockerfile.

FROM node:alpine. → baseimage.

WORKDIR /usr/app
COPY ./ /usr/app. } copy Local files to container.

RUN npm install → dependiencies.

CMD ["npm", "start"] → default command

Step2: Create a package with .json [Node Json related]

vi package.json

→ "dependencies": {

"express": "*"

},

"scripts": {

"start": "node index.js" → save & quit.

}

}

Step3. create a index.js file

vi index.js

```
const express = require('express');
const app = express();
app.get('/', (req,res) => {
res.send('welcome To cloud');
});
app.listen(8080, () =>
Console.log('Listening port 8080');
});
```

→ sq

Step4: → tagname

docker build -t rnraju/webserver:nodejs .

Step5: docker images.

Step6 docker run -it -d -p 7000:8000

--name nodeJS cef131d7cff1

Step7:

docker ps.

Step8: Ipvu+7000 [Google]

Step9: upload image. in dH

docker push rnraju/webserver:nodejs

* Docker Commit :- It is used to create a new image from a containers changes.

→ docker pull Centos.

→ container name.

Step2: docker run -it -d --name myserver cent

docker ps.

Step3: To Enter in to Container inside

docker exec -it a3d557u bash

Steps: cd /opt.

touch aws azure gcp.

ls.

exit → container to saver.

$ → ./J #
↓
Newcontain

Steps: To create a image from the container images.

~~docker commit rnraju/Webserver : Centos.~~

CO.

CID

→ docker commit a3d557 rnraju/webserver : test server.

→ docker images.

→ docker exec -it rnraju/webserver : testserver

ls /opt.

→ make container [build]

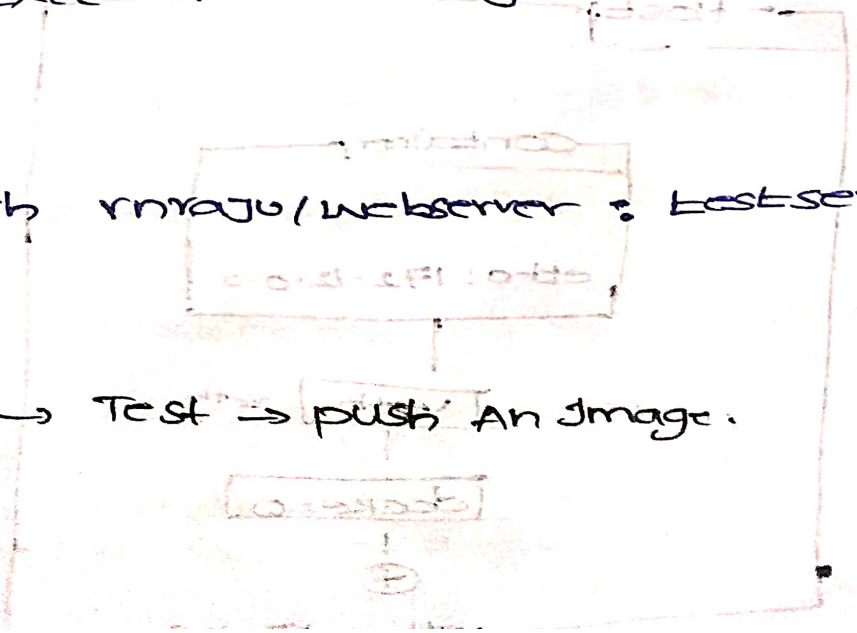
docker run -it -d conID

exec & test [test]

docker exec -it ImageId

push.

docker push rnraju/webserver : testserver.



Build → Test → push An Image.

29/12/22

Docker Networking:

→ It include support for networking containers through the use of network drivers.

→ To establish communication between Containers & outside world via the Docker host machine.

→ Docker support different types of Network drivers fit for a certain use cases.

N/W Drivers:

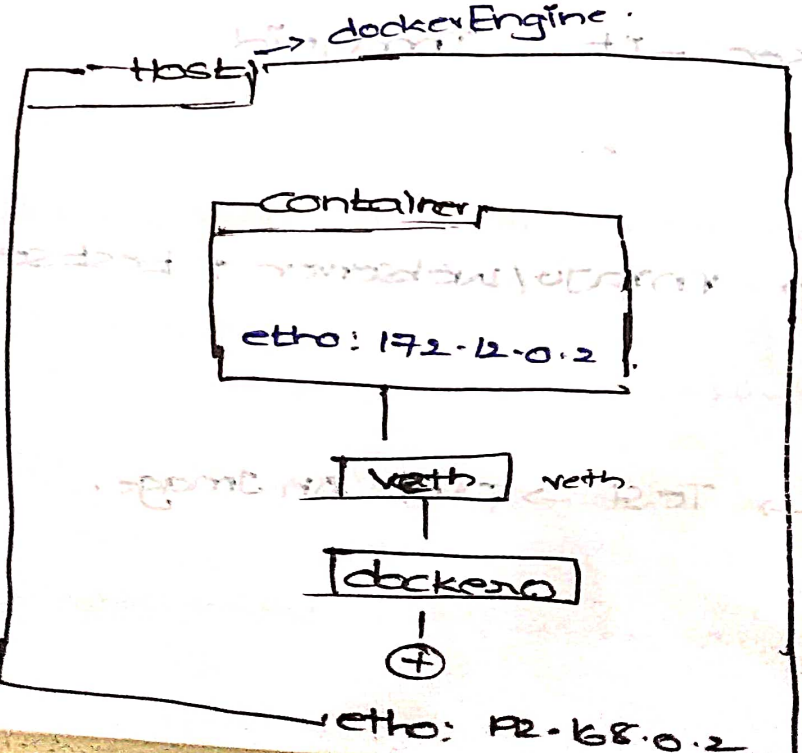
Bridge
Host
overlay
IPvlan
macvlan
None

Network plugins.

[Bridge, Host, None] ↓ supported by

Bridge N/W: A default Bridge Network (Bridge) is created automatically, when you start docker.

→ By default, Containers are running on default bridge



→ ifconfig → 172-17-0-1

create a container | subnet → portion in a N/W

→ To check the Network details → docker network ls.

→ To check bridge N/W details → docker network inspect bridge.

→ create a container in a Network.

→ container Name

docker run -it --name test1 alpine

ifconfig → 172-17-0-2. inetaddress.

→ Duplicate session, launch another container.

docker run -it --name test2 alpine

ifconfig → 172-17-0-3

*→ To check N/W activities alive (or) not ? → container talk automatically with each other

ping 172-17-0-2 → ctrl+c [stop]

exit.

→ To change container Host Name (by default it id value

docker run --rm -it --name=test3 --hostname

test3.example.com alpine sh

→ To check it :- hostname

cat /etc/hostname.

exit.

User defined bridge N/W :-

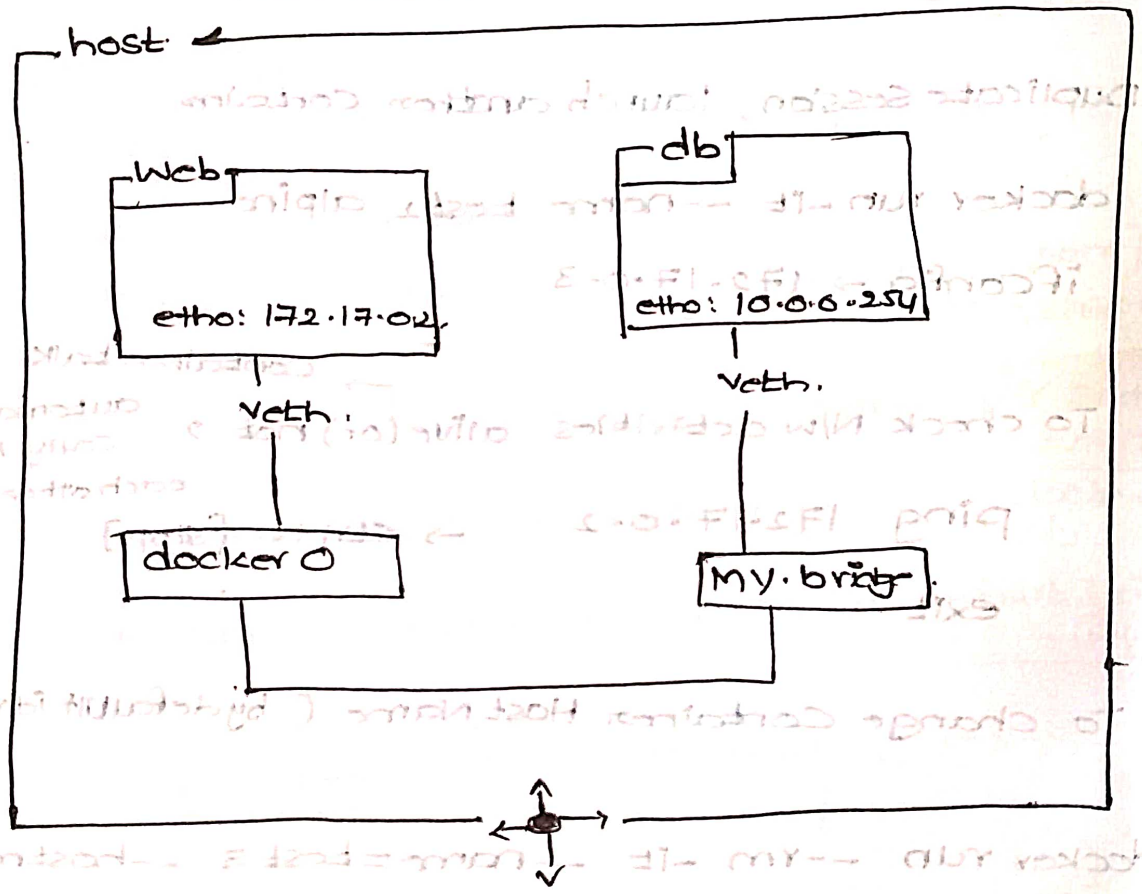
→ These Network are Superior to the default bri

→ It is usually used when your applications run in

standalone containers that need to communicate

→ These are best when you need "multiple containe

to communicate on the same Docker Host.



* available N/W → docker network ls.

→ How to create Network

docker network create my-bridge.

docker network ls.

details of N/W. docker network inspect my-bridge.

It provide N/W id. → 172-18.0.0716.

→ Create a Container on the my-bridge Network

Name of bridge

~~docker network my-bridge~~

→ docker run -it --name webserver --net my-bric

alpine.

→ ifconfig.

→ exit

→ create a user defined Network :-

docker network create productionteam-bridge.

--subnet 10.0.0.0/24 --gateway 10.0.0.1

gateway follows alw

1 series from subn

→ docker network ls.

→ N/W details → docker network inspect productio

team-bridge.

Create a container.

docker run -it --name Webserver1 --net.

productionteam-bridge. alpine.

Result. Ipaddress → 10.0.0.2

exit.

Host N/W

→ For Standalone containers, remove Network isolation b/w the Container and the docker Host and use the host Networking directly.

→ These are the best when the N/W stack should not be isolated from the docker host but you want other aspects of the container to be isolated.

* create a container on the Host N/W :-

→ docker run --rm -dit --network host --name my_nginx nginx

→ docker network ls.

2. Verify both Host machine and N/W Container machine

docker run -it --network host --name my_nginx alpine.

∴ Running on same (Address)

ifconfig → check docker host machine address.

3. None (Network): It completly disable the N/working stack on a container.

→ Usually used in conjunction with a custom N/W drivers. Note:- Not available for swarm services,

* Create a container on None N/W

docker run -it --network none --name myalpine alpine.

ifconfig.

exit.

* Delete a Network:

1. Before deleting a N/W, we must "stop" the running container.

→ docker ps.

docker stop containerID.

docker system prune -a

→ docker network ls. → bridge
host
None.

* To delete a particular N/W.

docker network rm NetworkID [bridge]

28/10/22

Manage Docker storage :-

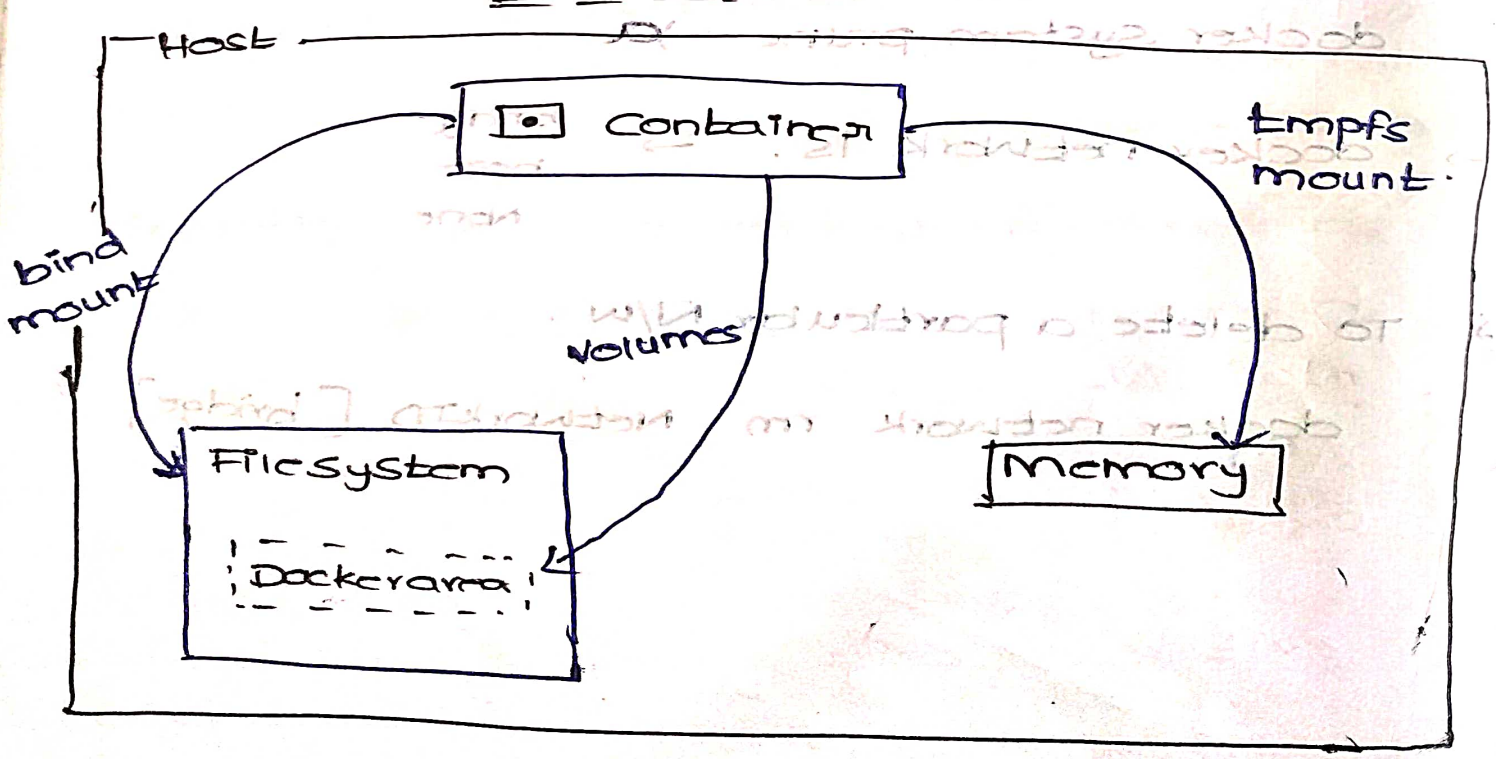
→ By default, all files created inside a container are stored on a "writable container layer"

→ That means :
- The data doesn't persist when that container no longer exists
- you can't easily move the data somewhere else

→ Docker has two options for containers to store files in the "Host machine persistently"
1. → volumes
2. → bind mounts.

→ Running Docker on Linux use a tmpfs mount and on Windows use a "named pipe"

Mounting Types.



1. Tmpfs mount :- A tmpfs mount is not persisted on disk, either on the "Docker host memory" or with in a Container.

→ It can be used by a Container during the lifetime of the Container.

→ Tmpfs mounts are never written to the "host file system"

docker run -it -d --name linux ubuntu
↳ name.

docker ps.

docker exec -it CID bash.

cd /opt.

touch aws azure gcp.

ls.

exit.

docker stop CID.

docker start CID.

docker exec -it CID ls /opt.

2. Volumes :-

→ Volumes are the best way to "persist data" in Docker.

→ These are stored in host file system which is managed by Docker (/var/lib/docker/volumes/ on linux) → default storage location.

→ Volumes support "volume drivers" which allows us to store data on remote host / cloud providers.

→ docker run -it -v /my-data (location to store data) --name linux ubuntu bash.

→ cd /opt.
→ cd
→ cd /my-data/
→ touch abc xyz.
→ ls

To check the volume:-

cd /var/lib/docker/volumes/
ls → 2files Idvalue 9f3ce----
3fbac----
cd Idvalue./
cd_data/
ls →abc xyz.

* Note: Delete a container and verify the volume data.

* multiple volumes for container:

→ docker run -it -v /my-data (V1) -v /data (V2).
ubuntu bash.
cd /my-data/
cd
cd /data/

→ Go to volumes [cd volumes] & verify.

cd /var/lib/docker/volumes/

ls

* Creating volumes :- [custom volume]

Syntax :- docker volume create [options] [volume]

check volume availability → docker volume ls.

→ create a volume :- docker volume create my-vol

docker volume ls.

output: Driver Volume Name

local my-vol

→ How to mount volume to a container?

→ mounting data folder

docker run -it -v my-vol:/my-data ubuntu bash

cd /my-data/

touch java php.

ls

exit.

Go to volumes :- cd /var/lib/docker/volumes/

and check ls.

cd my-vol/

ls → data

cd _data/

ls → Java php. (files)

* Bind mount :- Bind mounts may be stored anywhere on the host system.

→ A file (or) directory on the host machine is mount in to a container.

→ The file (or) directory is referenced by its "absolute path" on the host machine.

Note: Bind mounts have "Limited functionality" compared to volumes.

Example: /cloud : /aws (Host : container side)

→ docker run -it -v /cloud : /aws ubuntu /bin/bash

cd /aws.

touch a b c.

ls.

checking. cd /cloud/ } 2nd terminal

ls.

exit.

*** * How to mount "Readonly" Filesystem :-

docker run -it -v /cloud : /aws : ro ubuntu/bash (/bin)

cd /aws/

ls

touch -d [d - Readonly file]

* Volumes in a Dockerfile :-

vi Dockerfile

VOLUME ["/my-data"]

* Deleting volumes :-

docker volume ls.

→ Before removing volume first we need to stop the containers.

→ docker ps.
docker ps -a

→ docker Container rm CID1. CID2 CID3.

→ docker ps -a.

→ docker volume rm my-vol

→ docker volume ls.

→ docker volume prune. [Remove all]

→ docker volume ls.

volumes.

→ Easy backup and recovicks

→ To.

Bind mount

Named pipes:-

* A "npipe" mount can be used for communicati between the Docker host and a container.

→ Common use case is to run a third-party tool inside a container and connect to the Docker Engi API.

29/12/22

Docker Compose:-

→ It is a tool for defining and running multiple containers as a single service.

→ You use a YAML file to configure your application Servi

Compose IS Three Step process:-

Step1 :- Define your apps environment with a Dockerfile, so it can be reproduced anywhere.

Step2 :- Define the "services" that make up your app in docker-compose.yml so they can be run together in an isolated environment.

Step3: Run "docker compose up" and the Docker compose command starts and runs your entire app.

Filestructure of Docker Compose:

```
version : 'x'
services :
    web:
        build:
        ports:
          "5000:5000"
    volumes:
        . :/code.
    redis :
        image: redis.
```

Note:
Spaces are manadatory for yaml code.

* Installing Docker Compose:

1. Google → Install the compose standalone → on linux
download → copy command (paste) (With root permissi

Step2: Apply Executable permission to the standlone.

chmod +x /usr/local/bin/docker-compose

Step3: Create a symbolic link for a binary file.

ln -s /usr/local/bin/docker-compose /usr/bin/

↓ Link

docker-compose.

Step4: To check docker compose version.

docker-compose --version. [14.2]

* Create a compose file for running nginx Webs

1. Vi docker-compose.yml (nginx, redis container)

```
version : '3'
Services :
    Web :
      image : nginx
      ports : - 9090 : 80.

    database :
      image : redis.
```

* Note: [2 character Space need for yaml file]

:wq!

2. docker compose ps

3. docker compose up -d → create a container

docker compose ps. | docker ps.

4. publicIp + 9090 [verify the WebServer]

5. Delete a container. docker compose down.

docker ps.

Example-2 : vi docker-compose.yml

```
version : '3'

services :
  web :
    image : nginx
    ports :
      - 9090 : 80.

  database :
    image : redis.

  python :
    image : python.

  alpine :
    image : alpine.        : wq!
```

→ docker compose up -d

→ docker compose ps.

* To check the validity of a compose-file ?

docker-compose config

* To scale up and scale down services ?

cat docker-compose.yml

Scale up :

docker-compose up -d --scale database=5
(servicename)

docker compose ps.

Delete : docker Compose down